SOCIAL NETWORK ANALYSIS

Enrollment Number – 101327

Name of Student – Anuj Mittal

Name of Supervisor – Mr. Suman Saha

MAY-2014

Submitted in partial fulfillment of the Degree of

Bachelor of Technology

DEPARTMENT OF COMPUTER SCIENCE

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT

Table of Contents

<u>CERTIFICATE</u>

This is to certify that the work titled “**Social Network Analysis**“ submitted by “**Anuj Mittal”** in partial fulfillment for the award of degree of B.Tech Computer Science Engineering of Jaypee University of Information Technology, Waknaghat has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

____

(Signature of Supervisor)

Name of Supervisor **Mr Suman Saha**

Designation **Asst. Professor, Dept. of CSE and ICT**

Date ……………………..

ACKNOWLEDGEMENT

I express my sincere gratitude to my respected project supervisor Asst. Professor Mr. Suman Saha, Department of Computer Science And Engineering, Jaypee University of Information Technology, Waknaghat under whose supervision and guidance this work has been carried out. His whole hearted involvement, advice, support and constant encouragement throughout, have been responsible for carrying out this project work with confidence. I am also grateful to him for providing me with required infrastructural facilities that have been highly beneficial to me in undertaking the above mentioned project.

I am sincerely grateful to Brig. S.P. Ghrera, Professor and Head of Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat for providing all necessary facilities for the successful completion of my project.

I would also like to thank the laboratory staff of Department of Computer Science and Engineering for their timely help and assistance.

Signature of the student	……………………..
Name of Student	Anuj Mittal
Date	………………………

SUMMARY

Objective

To divide the network into various communities on the basis of characteristics

Description

As we know that data in real world is growing at a very fast pace. So it is very difficult to make analysis on this data which can be represented in the form of network. So to ease the computation or analysis of the complex network we divide the complete network into various communities on the basis of characteristics (i.e.   distance between the nodes, modularity of sub graph).

In this project i have studied three different algorithms for community detection in a network. These algorithms are based on different approaches. Firstly, Walktrap algorithm which works on agglomerative method in which edges are added to an initially empty network starting with the vertex pairs with highest similarity. Second, Multilevel Algorithm which works on modularity optimization approach. Modularity is the measure to check the quality of the community which is created by the adding or removing nodes from a community. Third, Girvan and Newman algorithm which follows divisive methodology which attempt to find the least similar connected pairs of vertices and then remove the edges between them. It divides the network into smaller and smaller components.

List of Figures

Chapter 1: INTRODUCTION

In 1967, social psychologist Stanley Milgram performed an experiment to solve an unresolved hypothesis circulating in those days. The hypothesis was called the small-world problem. The claim of the small-world phenomenon is that the world, is in a sense small, when viewed as a network of social acquaintances, could be reached through a network of friends in a only a few steps. Milgram asked a few hundred randomly selected people to send letters to a stock broker in Boston via intermediaries. They can send the letter to people they knew on first name basis. Among the letters that reached the destination correctly, the average path length was found to be six. This led to the phrase "six degrees of separation". This experiment laid the stage for algorithmic aspects this new and emerging science.

In order to make such a claim, instead of asking, "How small is our world", one could ask, "What would it take for any world to be small?" In other words, we want to construct a mathematical model of the world in which the individuals are represented as nodes and relationships are represented as edges. This allows analysis using tools of mathematics.

Imagine one has one hundred friends, each one of them also has hundred friends. So at one degree of separation one connects to one hundred people and at two degrees connects to one hundred times one hundred. Proceeding in a similar fashion, in five degrees he is connected to nine billion people. So if everyone has one hundred friends, then within six steps he can connect himself to the entire population.

But there is one important omission in this reasoning. Chances are that one will come up with many of the same people in one's friends' network. We tend to have groups of friends, each of which is like a community or 2 cluster based on shared experience, location, or interests, joined to each other by overlaps created when individuals in one group also belong to other groups. This is particularly relevant, because clustering breeds redundancy

As social networks gain prominence, the first obvious question that comes in observing these networks is: how to extract meaningful knowledge from these data? In seeking a response, the network structure proves to be of utmost importance. Identifying high-order structures within networks yields insights into their functional organization, which in turn contributes more knowledge while offering many possible actions, including marketing plans, recommendations and user interface adaptations. Community detection may become a more complicated task given that social networks can be structured on many different levels, yet communities reduce the complexity of a network's original graph in a substantial way, thus revealing its macro-structure.

A property that seems to be common to many networks is community structure, the division of network nodes into groups within which the network connections are dense, but between which they are sparser. The ability to find and analyze such groups can provide invaluable help in understanding and visualizing the structure of networks.

The study of community structure in networks has a long history. It is closely related to the ideas of graph partitioning in graph theory and computer science.

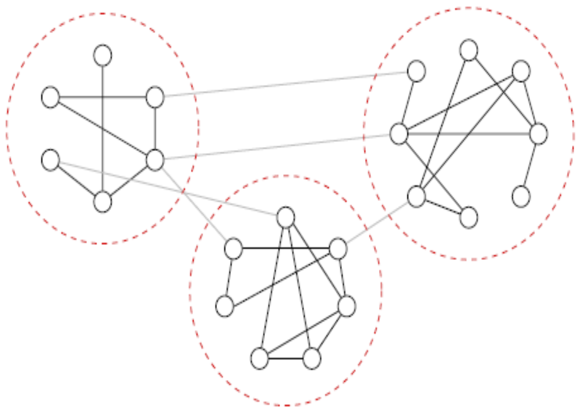


Fig 1: A small network with community structure. In this case there are three communities, denoted by the dashed circles, which have dense internal links but between which there are only a lower density of external links.

Graph partitioning is a problem that arises in, for example, parallel computing. Suppose we have a number n of intercommunicating computer processes, which we wish to distribute over a number g of computer processors. Processes do not necessarily need to communicate with all others, and the pattern of required communications can be represented by a graph or network in which the vertices represent processes and edges join process pairs that need to communicate. The problem is to allocate the processes to processors in such a way as roughly to balance the load on each processor, while at the same time minimizing the number of edges that run between processors, so that the amount of interprocessor communication (which is normally slow) is minimized.

In general, finding an exact solution to a partitioning task of this kind is believed to be an NP-complete problem, making it prohibitively difficult to solve for large graphs, but a wide variety of heuristic algorithms have been developed that give acceptably good solutions

A solution to the graph partitioning problem is however not particularly helpful for analyzing and understanding networks in general. If we merely want to find if and how a given network breaks down into communities, we probably don't know how many such communities there are going to be, and there is no reason why they should be roughly the same size. Furthermore, the number of inter-community edges needn't be strictly minimized either, since more such edges are admissible between large communities than between small ones.

1.1 Problem Statement

What I supposed to do?

As social networks gain prominence, the first obvious question that comes in mind in observing these networks is: how to extract meaningful knowledge from these data?

As we discussed earlier, the increasing complexity of the graph while analyzing. So finding the communities reduces the complexity of a network's original graph. So I have implemented an algorithm for community detection to solve the above mentioned problem.

1.2 Motivation

With the increase in communication bandwidth and the shrinking of distances, it has become increasingly possible for the formation and working together of various communities of interest with significant reduction in time and cost. Understanding the formation of such groups and sustenance of them has become a topic of great interest in recent times. The growth of Internet and computer mediated communications such as emails, newsgroups, online discussion forums, Internet Relay Chats have revolutionized the formation of such communities of interest. Weblogs and numerous social networking websites like Facebook, Orkut, and so on are a recent addition to the repertoire of computer mediated communications. Social Networking web services are internet applications that help connect friends, business partners or just about any one by allowing users to have profiles, blogs, interact with others, join or create communities and much more. This revolution in communication has shifted the techniques for understanding such dynamics from social and psychological perspectives to the domain of computer science. Social Network Analysis using such techniques has become a powerful tool for such analysis.

Chapter 2: LITERATURE REVIEW

2.1)Definitions:

2.1.1)Social Network Analysis

Social network analysis (SNA) is a set of research procedures for identifying structures in systems based on the relations among actors. Grounded in graph and system theories, this approach has proven to be a powerful tool for studying networks in physical and social world. SNA focuses on relations and ties in studying actor's behavior and attitudes. Thus the positions of actors within a network and the strength of ties between them become critically important. Social position can be evaluated by finding the centrality of a node identified through a number of connections among network members. Such measures are used to characterize degrees of influence, prominence and importance of certain members. Tie strength mostly involves closeness of bond. There is general agreement that strong ties contribute to intensive resource exchange and close communities, whereas weak ties provide integration of relatively separated social groups into larger social networks.

Properties of Social Networks

Properties that seem to be common to many networks: the small-world property, Right-skewed distributions, and network transitivity.

Small world effect is the average distance between vertices in a network scaling logarithmically with the total number n of vertices.

Right-skewed degree distribution is another property that many networks possess. The degree of a vertex in a network is the total number of other vertices to which it is attached, and it was found that vertices in a network possess low degree and a few vertices with high degree, the precise distribution follows exponential form.

A third property that many networks follow is network transitivity, which suggests that if two vertices that both are neighbors of the same third vertex have a more probability of also being neighbors of each other. This is similar to the fact that if two of your friends will have a higher probability of knowing each other than two people chosen at random from the population, on basis of their common acquaintance with you.

2.1.2) Unipartite and Bipartite graphs

A graph is a representation of a set of objects called vertices, some of which are connected by links. Object connections are depicted by links, also called edges. Such a mathematical structure may be referred to as a unipartite graph. A good example of this type of graph is the well-known Zakary Karate club .

A special case of this graph is known as the bipartite graph, i.e. whose vertices can be divided into two disjoint sets A and B such that the edges only connect one vertex in A to one in B, in considering that A and B are independent sets. Vertices of A are not connected to any other vertices within A, and the same applies for B. For example, let A be a set of individuals and B a set of photos showing these same individuals.

2.1.3) Hypergraph

A hypergraph H is a pair (V, E) where V = v1, v2, ..., vn is a non-empty (usually limited) set and E = E1, E2, ..., Em is a family of not empty subsets of V. The elements of V are the vertices of H. The elements of E are the edges (also called hyperedges) of H. A set of social communities can be viewed as a hypergraph whose vertices are the individuals and whose hyperedges are the communities.

In the field of community detection seek to partition individuals into communities, i.e. non-intersecting hyperedges.

2.1.4)Galois lattice

Freeman was the first to use Galois lattices in order to represent network data. The underlying assumption is that individuals sharing the same subset of properties define a community. The approach adopted consists of the following: objects, attributes and the set of relations between objects and attributes

This set of relations can then be represented by a binary bi-adjacency matrix, whereby objects o are the columns, attributes a are the rows and a “1” is placed at the cell corresponding to a pair (oi, aj) if oi possesses aj . A maximum subset of objects that contain a subset of attributes is defined as a “concept”, i.e. a group of objects for which the addition or removal of an attribute changes its constitution. All objects of a concept then form the “extent” and all attributes of a concept give rise to the “intent”. A partial order is applied to concepts and serves to establish a hierarchy. According to the definition of Galois hierarchies, an object can appear in all the concepts where it can share the same set of attributes with other objects belonging to other concepts.

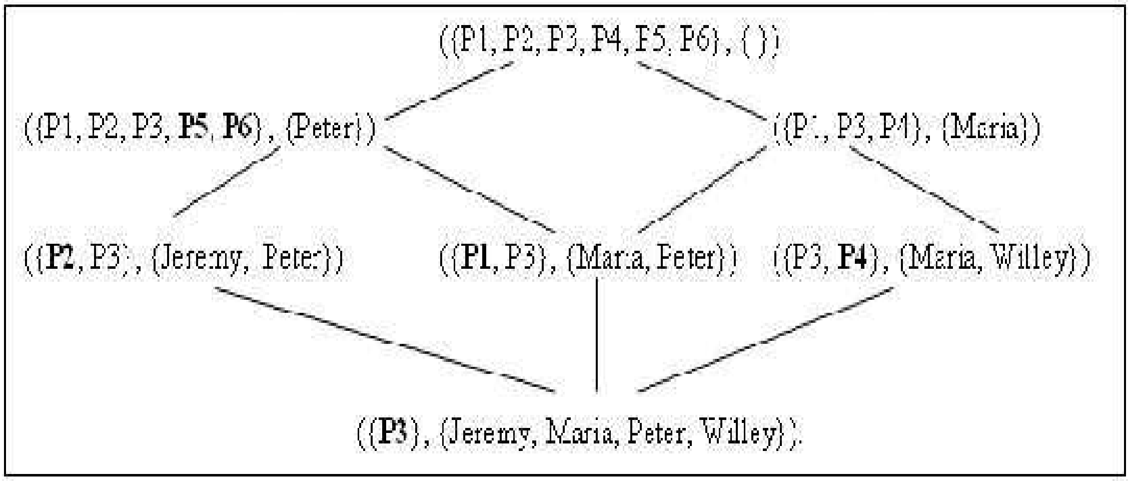

({P1, P2, P3, P4, P5, P6}, { })
({P1, P2, P3, P5, P6}, {Peter})
({P1, P3, P4}, {Maria})
({P2, P3}, {Jeremy, Peter})
({P1, P3}, {Maria, Peter})
({P3, P4}, {Maria, Willey})
({P3}, {Jeremy, Maria, Peter, Willey})

Fig 2: Example of a Galois lattice in which several individuals are sharing several photos.

2.1.5) Modularity

Modularity is a measure of strength of sub graph of a network into communities. Networks with high modularity will have a compact structure among the nodes of the same community but sparse connections between nodes in different communities.
Modularity measures the capacity of a given graph partition to yield the densest groups.

Modularity of a sub graph can be calculated as below

$$M=1/(2m) * \sum Aij-(ki*kj)/(2m))\ \delta(ci,cj)$$

If ci=cj then $\delta(ci,cj) =1$
Else $\delta(ci,cj) =0$

$m=1/2 \sum ij\ Aij$
Aij – weight of edge between node i and node j
ki – sum of weight of edges attached to vertex i
Ci – community

This formula has mainly been used in order to measure the ability of a community detection algorithm to obtain a satisfactory partition of a given graph.

2.2) Algorithms for Community Detection

2.2.1) Walktrap Algorithm

This approach is based on the following:

Random walks on a graph tend to get "trapped" into densely connected parts corresponding to communities.

In discrete random walk process on the graph G, at each time step a walker is on a vertex and moves to a vertex chosen randomly and uniformly among its neighbors. At each step, the transition probability from vertex i to vertex j is Pij = Aij/d(i). The probability of going from i to j through a random walk of length t is (Pt)ij .

It satisfies two properties of the random walk process

Property 1: When the length t of a random walk starting at vertex i tends towards infinity, the probability of being on a vertex j only depends on the degree of vertex j.

Property 2: The probabilities of going from i to j and from j to i through a random walk of a fixed length t have a ratio that only depends on the degrees d(i) and d(j)

In order to group the vertices into communities, we will use distance r as a parameter between the vertices that captures the community structure of the graph. This distance must be large if the two vertices are in different communities, and on the contrary if they are in the same community it must be small.

To compare two vertices i and j using these data, we must notice that:

- If two vertices i and j are in the same community, the probability Pt ij will surely be high. But the fact that Pt ij is high does not necessarily imply that i and j are in the same community.
- The probability Pt ij is influenced by the degree d(j) because the walker has higher probability to go to high degree vertices.
- Two vertices of a same community tend to "see" all the other vertices in the same way. Thus if i and j are in the same community, then $P^t ik = P^t jk$

The distance between two nodes can be found out by the help of following formula

$$rij = (\sqrt{\sum}(P^t ik - P^t jk)^2)/d(k)$$

rij is the distance between the node i and node j

$P^t ij$ is the probability of going from 1 node to other in t steps

d(i)= number of adjacent neighbors

The probability can be found out by given formula

Pij = Aij/d(i)

Aij = 0 or 1 depending on whether the edge is present between the nodes

d(i)= number of adjacent neighbors

Algorithm

Start from a partition P1 = {{v}, v □ V } of the graph into n communities reduced to a single vertex. We first compute the distances between all adjacent vertices. Then this partition evolves by repeating the following operations. At each step k:

- Choose two communities C1 and C2 in Pk(partition k) according to the distance between the communities
- Merge these two communities into a new community C3 = C1 ∪ C2 and create the new partition: Pk+1 = (Pk \ {C1,C2}) ∪ {C3}
- update the distances between adjacent communities

After n − 1 steps, the algorithm finishes and we obtain Pn = {V }. Each step defines a partition Pk of the graph into communities, which gives a hierarchical structure of communities. This structure is a tree in which the leaves correspond to the vertices and each internal node is associated to a merging of communities in the algorithm: it corresponds to a community composed of the union of the communities corresponding to its children.

Choosing the communities to merge

We will only merge adjacent communities (having at least an edge between them). Moreover it ensures that each community is connected.

Two communities will merge according to Ward's method. At each step k, we merge the two communities that minimize the mean σ_k of the squared distances between each vertex and its community.

$$\sigma_k = 1/n \sum_{C \in Pk} \sum_{i \in C} r^2_{ic}$$

Evaluating the quality of a partition

Quality of partition can be measured using the given formulae

$$Q(P) = \sum_{C \in P} e_C - a^2_C$$

Where,

e_C is fraction of edges inside community C

a_C is fraction of edges bound to community C

The best partition is then considered to be the one that maximizes Q.

Complexity of walktrap algorithm is $O(mn^2)$ where m is number of edges and n number of vertices.

2.2.2) Multilevel Algorithm

This algorithm is based on modularity optimization in short time and that unfolds a complete hierarchical community structure for the network. The quality of the partitions detected is very good, as measured by the so-called modularity.

This algorithm is based on following two phases that are repeated iteratively:

i) <u>Modularity is optimized by allowing only local changes of communities</u>

A different community is assigned to each node of the network. So, in this initial partition there are as many communities as there are nodes. Then, for each node i the neighbours j of i is choosen and we evaluate the gain of modularity that would take place by removing i from its community and by placing it in the community of j. The node i is then placed in the community for which this gain is maximum. but only if this gain is positive. If no positive gain is possible, i stays in its original community. This process is applied repeatedly and sequentially for all nodes until no further improvement can be achieved.

This first phase stops when a local maxima of the modularity is attained, i.e., when no individual move can improve the modularity. The ordering of choosing the nodes can influence the computation time.

The gain in modularity Δ Q obtained by moving an isolated node i into a community C can easily be computed by:

$$\Delta Q = \left[\left(\frac{\sum in + k_{i,in}}{2m}\right) - \left(\frac{\sum tot + k_i}{2m}\right)^2 \right] - \left[\left(\frac{\sum in}{2m}\right) - \left(\frac{\sum in}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2 \right]$$

Where ∑in is the sum of the weights of the links inside C, ∑tot is the sum of the weights of the links incident to nodes in C, ki is the sum of the weights of the links incident to node i, k i,in is the sum of the weights of the links from i to nodes in C and m is the sum of the weights of all the links in the network.

A similar expression is used in order to evaluate the change of modularity when i is removed from its community. One therefore evaluates the change of modularity by removing i from its community and then by moving it into a neighboring community.

ii) Found communities are aggregated in order to build a new network

The weights of the links between the new nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. Links between nodes of the same community lead to self-loops for this community in the new network.

Once this second phase is completed, it is then possible to reapply the first phase of the algorithm to the resulting weighted network and to iterate. It is denoted by ”pass” a combination of these two phases. By construction, the number of meta-communities decreases at each pass, and as a consequence most of the computing time is used in the first pass. The passes are iterated until there are no more changes and a maximum of modularity is attained.

Advantages

1. First, its steps are intuitive and easy to implement, and the outcome is unsupervised
2. The algorithm is extremely fast, i.e., computer simulations on large ad-hoc modular networks suggest that its complexity is linear on typical and sparse data. As the number of communities decreases drastically after just a few passes so that most of the running time is concentrated on the first iterations.

But this algorithm fails to identify communities smaller than a certain scale, thereby inducing a resolution limit on the community detected by a pure modularity optimization approach.

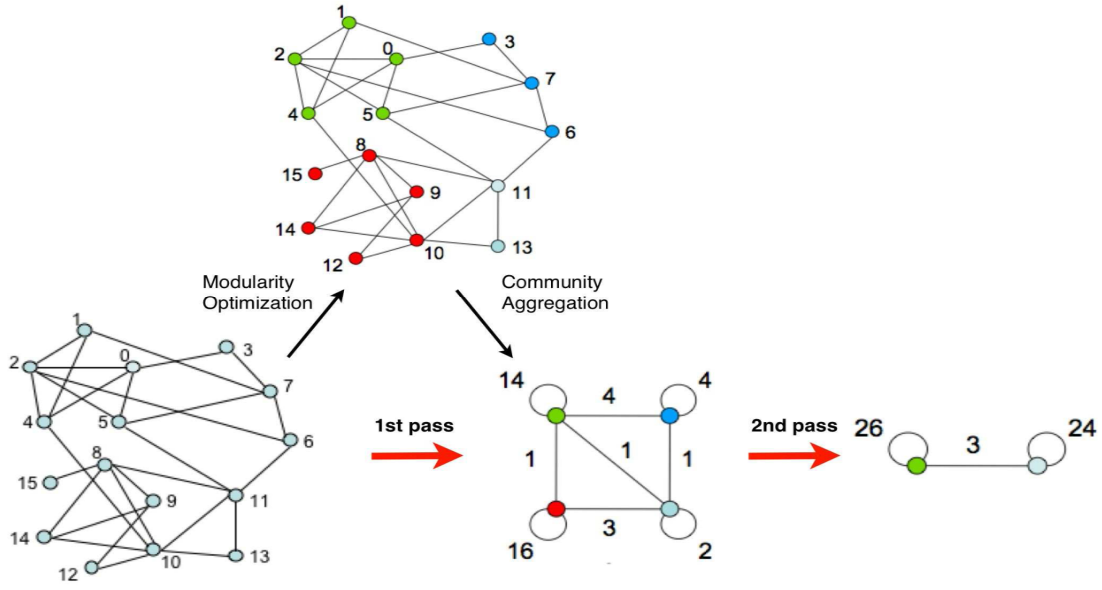


Fig 3: Visualization of steps of multilevel algorithm.

2.2.3) Girvan and Newman

Algorithms fall into two broad classes, agglomerative and divisive, depending on whether they focus on the addition or removal of edges to or from the network.

In an agglomerative method, similarities are calculated by one method or another between vertex pairs, and edges are then added to an initially empty network (n vertices with no edges) starting with the vertex pairs with highest similarity. The procedure can be halted at any point, and the resulting components in the network are taken to be the communities. Agglomerative methods based on a wide variety of similarity measures have been applied to different networks. Some networks have natural similarity metrics built in.

For example, in the widely studied network of collaborations between film actors in which two actors are connected if they have appeared in the same film, one could quantify similarity by how many films actors have appeared in together.

Agglomerative methods have their problems. One concern is that they fail with some frequency to find the correct communities in networks were the community structure is known, which makes it difficult to place much trust in them in other cases. Another is their tendency to find only the cores of communities and leave out the periphery. The core nodes in a community often have strong similarity, and hence are connected early in the agglomerative process, but peripheral nodes that have no strong similarity to others tend to get neglected. There are a number of peripheral nodes whose community membership is obvious to the eye—in most cases they have only a single link to a specific community— but agglomerative methods often fail to place such nodes correctly.

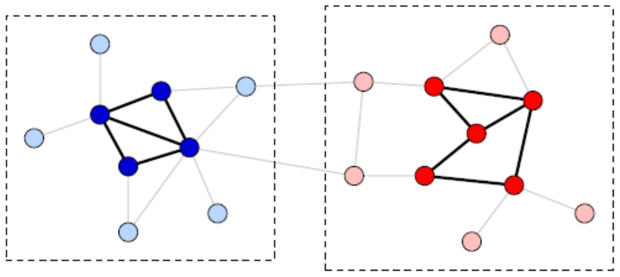


FIG 4: Agglomerative clustering methods are typically good at discovering the strongly linked cores of communities (bold vertices and edges) but tend to leave out peripheral vertices, even when most of them clearly belong to one community or another.

In a divisive method, we start with the network of interest and attempt to find the least similar connected pairs of vertices and then remove the edges between them. By doing this repeatedly, we divide the network into smaller and smaller components, and again we can stop the process at any stage and take the components at that stage to be the network communities. Rather than looking for the most weakly connected vertex pairs, our approach will be to look for the edges in the network that are most "between" other vertices, meaning that the edge is, in some sense, responsible for connecting many pairs of others. Such edges need not be weak at all in the similarity sense.

This algorithm belongs to the category of divisive algorithms. Its underlying principle calls for removing the edges that connect different communities. In the algorithm, several measures of edge centrality are computed, in particular the so-called intermediate centrality, whereby edges are selected by estimating the level of edge importance based on these measures. As an illustration, intermediate centrality is defined as the number of shortest paths using the edge under analysis.

Features

This Girvan and Newman algorithm shares two definitive features:

1) They involve iterative removal of edges from the network to split it into communities, the edges removed being identified using one of a number of possible "betweenness" measures and these measures are recalculated after each removal.

2) The inclusion of a "recalculation step" in the algorithm

If Girvan and Newman were to perform a standard divisive clustering based on edge betweenness then they would calculate the edge betweenness for all edges in the network and then remove edges in decreasing order of betweenness.

However, once the first edge in the network is removed in this algorithm, the betweenness values for the remaining edges will no longer reflect the network as it now is. This can give rise to unwanted behaviors. For example, if two communities are joined by two edges, but, for one reason or another, most paths between the two flow along just one of those edges, then that edge will have a high betweenness score and the other will not. An algorithm that calculated betweennesses only once and then removed edges in betweenness order would remove the first edge early in the course of its operation, but the second might not get removed until much later. Thus the obvious division of the network into two parts might not be discovered by the algorithm. In the worst case the two parts themselves might be individually broken up before the division between the two is made. The solution is that they simply recalculate betweenness measure after the removal of each edge. This certainly adds to the computational effort of performing the calculation, but its effect on the results is so desirable that they consider the price worth paying.

Algorithm

Calculate the betweenness for all edges in the network.

1. Remove the edge with the highest betweenness.
2. Recalculate betweenness for all edges affected by the removal.
3. Repeat from step 2 until no edges remain.

The procedure of link removal ends when the modularity of the resulting partition reaches a maximum.

The recalculation step is the most important feature of the algorithm, as far as getting satisfactory results is concerned. This measure appears to work well and is the quickest to calculate, it can be calculated for all edges in time O(mn), where m is the number of edges in the graph and n is the number of vertices.

The betweenness between the two nodes can be calculated by using the formula

$C_B(v) = \sum_{s,t \in V} \sigma(s,t|e)/\sigma(s,t)$

s is the source node

t is the destination node

$\sigma(s,t)$ is the number of shortest paths between pairs of nodes

$\sigma(s,t|e)$ is the number of shortest paths between pairs of nodes that pass through the link

2.3) Applications

Communities in a network might represent real social groupings, perhaps by interest or background; communities in a citation network might represent related papers on a single topic; communities in a metabolic network might represent cycles and other functional groupings; communities on the web might represent pages on related topics; hidden communities might represent potential suspicious activity. Being able to identify these communities could help us understand and exploit these networks more effectively.

Medical science: Suppose we have found formula to cure a particular type of infection. So we can apply community detection on the database of the patients to group together patients having same type of symptoms. We can give same cure to each of these patients.

Recommender systems development: In this people having common school, interests, friends and many more can be grouped together in the same community. This community result must be used by recommender system while recommending friends or pages to others.

Knowledge Management and Collaboration: SNAs can help locate expertise, seed new communities of practice, develop cross-functional knowledge-sharing, and improve strategic decision-making across leadership teams.

Team-building: SNAs can contribute to the creation of innovative teams and facilitate post-merger integration. SNAs can reveal, for example, which individuals are most likely to be exposed to new ideas.

Human Resources: SNAs can identify and monitor the effects of workforce diversity, on-boarding and retention, and leadership development. For instance, an SNA can reveal whether or not mentors are creating relationships between mentees and other employees.

Strategy: SNAs can support industry ecosystem analysis as well as partnerships and alliances. They can pinpoint which firms are linked to critical industry players and which are not.

Chapter 3: Simulation

3.1) Rstudio

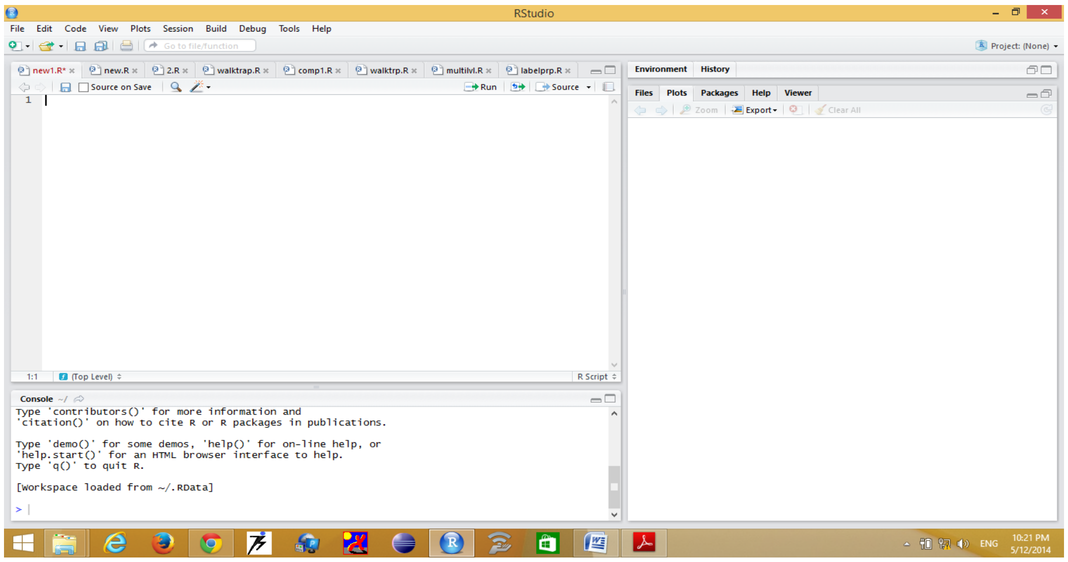
RStudio IDE is a powerful and productive user interface for R, a programming language for statistical computing and graphics.. It's free and open source.

It is a GNU project which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

Some of its features include:

1) Customizable workbench with all of the tools required to work with R in one place (console, source, plots, workspace, help, history, etc.).
2) Syntax highlighting editor with code completion.
3) Execute code directly from the source editor (line, selection, or file).
4) Runs on all major platforms (Windows, Mac, and Linux) and can also be run as a server, enabling multiple users to access the RStudio IDE using a web browser.

It supports file with .r extension. R provides a wide variety of statistical (linear and nonlinear modeling, classical statistical tests, time-series analysis, classification, clustering) and graphical techniques.


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Project: (None)
new1.R* new.R 2.R walktrap.R comp1.R walktrp.R multilvl.R labelprp.R
Source on Save
Run
Source
1:1 (Top Level) R Script
Console ~/
Environment History
Files Plots Packages Help Viewer
Zoom Export Clear All
10:21 PM 5/12/2014

```
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

>
```

Fig 5: RStudio

3.2) Input

Zachary graph:

Over the course of two years in the early 1970s, Wayne Zachary observed social interactions between the members of a karate club at an American university. He constructed networks of ties between members of the club based on their social interactions both within the club and away from it. By chance, a dispute arose during the course of his study between the club's administrator and its principal karate teacher over whether to raise club fees, and as a result the club eventually split in two, forming two smaller clubs, centered around the administrator and the teacher.

It is an undirected and unweighted graph. It consists of 34 vertices and 78 edges.

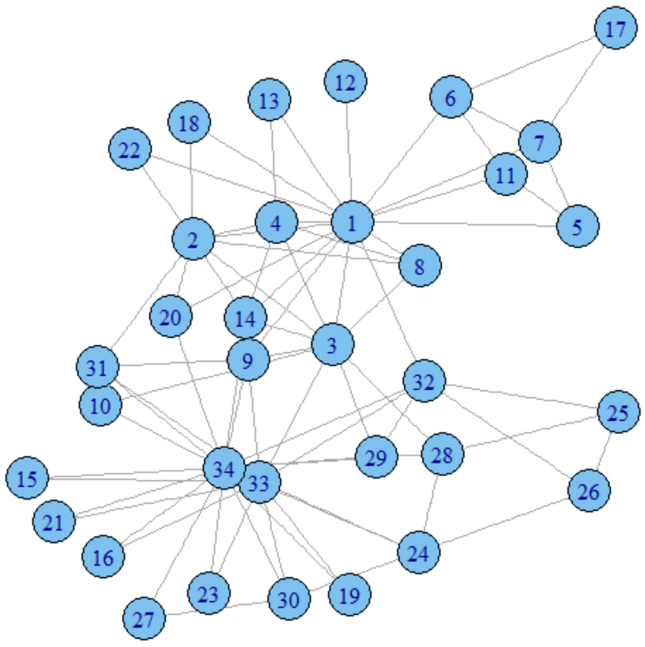


Fig 6: A network structure extracted from Zachary's observations before the split.

3.3) Output

Walktrap

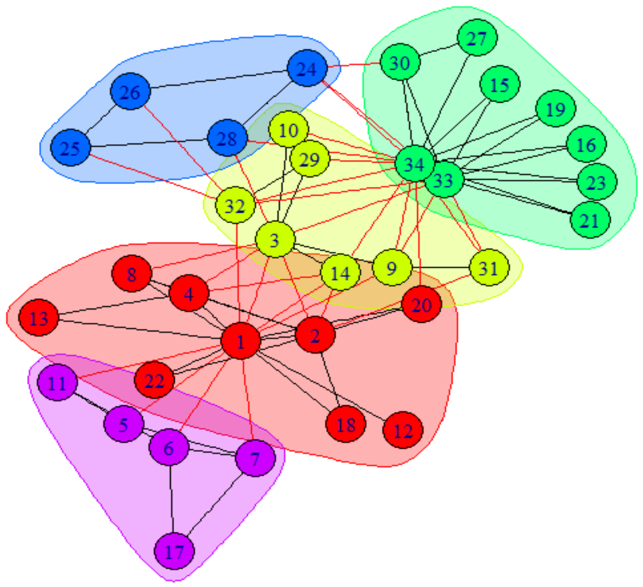


Fig 7: Output of walktrap algorithm using RStudio with Zachary graph as input

- Modularity in case of walktrap = 0.3532216

Multilevel

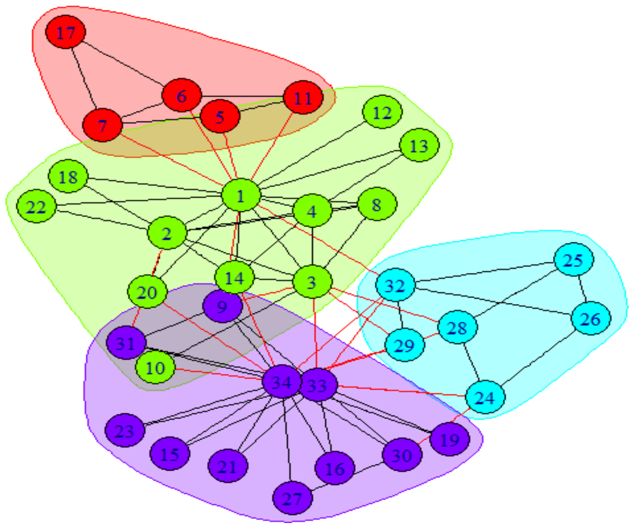


Fig 8: Output of multilevel algorithm using RStudio with Zachary graph as input

- Modularity in case of multilevel= 0.4188034

Girvan and Newman

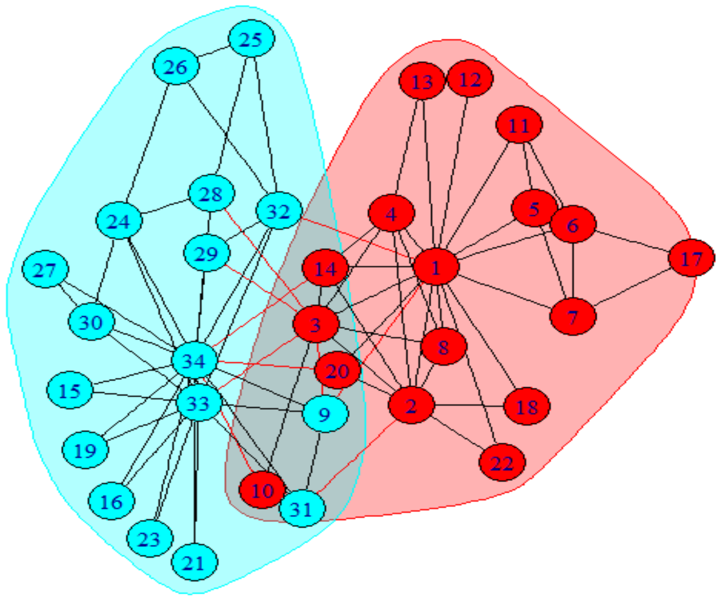


Fig 9: Output of Girvan and Newman algorithm using RStudio with Zachary graph as input

- Modularity in case of Girvan and Newman= 0.3717949

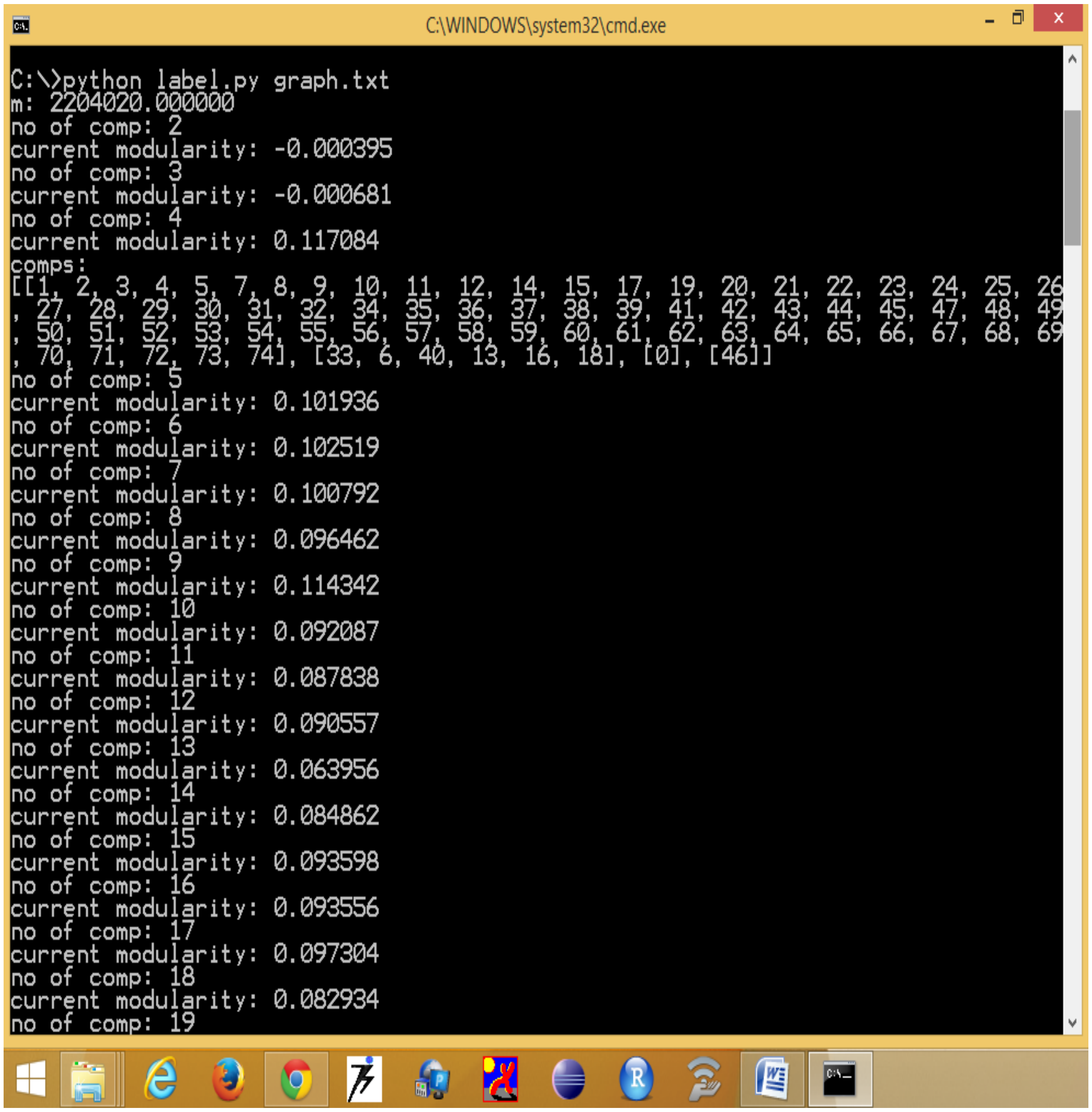
More is the modularity, better will the strength of the community and more compact or highly dense will be the structure.

Chapter 4: CODE IMPLEMENTATION

4.1) Code

```
import networkx as nx

import math

import numpy

import csv

import random as rand

import sys


def buildG(G, file1, delimiter_):

    reader = csv.reader(open(file1), delimiter=delimiter_)

    for line in reader:

        if float(line[2]) != 0.0:

            G.add_edge(int(line[0]),int(line[1]),weight=float(line[2]))

def edgbtw(G):

    init_ncomp = nx.number_connected_components(G)

    ncomp = init_ncomp

    while ncomp <= init_ncomp:

        bw = nx.edge_betweenness_centrality(G)

        max_ = 0.0

        for k, v in bw.iteritems():

            _BW = float(v)/float(G[k[0]][k[1]]['weight'])

            if _BW >= max_:

                max_ = _BW
        for k, v in bw.iteritems():
            if float(v)/float(G[k[0]][k[1]]['weight']) == max_:
                G.remove_edge(k[0],k[1])
        ncomp = nx.number_connected_components(G)
def GetModularity(G, deg_, m_):
    New_A = nx.adj_matrix(G)
    New_deg = {}
    New_deg = UpdateDeg(New_A)
    comps = nx.connected_components(G)
    print 'no of comp: %d' % len(comps)
    Modu = 0
    for c in comps:
        EWC = 0
        RE = 0
        for u in c:
            EWC += New_deg[u]
            RE += deg_[u]
        Modu += ( float(EWC) - float(RE*RE)/float(2*m_) )
    Modu = Modu/float(2*m_)
    return Modu


def UpdateDeg(A):
    deg_ = {}
    n = len(A)

    for i in range(n):

        deg = 0.0

        for j in range(n):

            deg += A[i,j]

        deg_[i] = deg

    return deg_

def newmang(G, Orig_deg, m_):

       BestQ = 0.0

    Q = 0.0

    while True:

        edgbtw(G)

        Q = GetModularity(G, Orig_deg, m_);

        print "current modularity: %f" % Q

        if Q > BestQ:

            BestQ = Q

            Bestcomps = nx.connected_components(G)

            print "comps:"

            print Bestcomps

        if G.number_of_edges() == 0:

            break

    if BestQ > 0.0:

        print "Best Q: %f" % BestQ
        print Bestcomps
    else:
        print "Best Q: %f" % BestQ
def main(argv):
    if len(argv) < 2:
        sys.stderr.write("Usage: %s <input graph>\n" % (argv[0],))
        return 1
    graph_fn = argv[1]
    G = nx.Graph()
    buildG(G, graph_fn, ',')
   n = G.number_of_nodes()
    A = nx.adj_matrix(G)
    m_ = 0.0
    for i in range(0,n):
        for j in range(0,n):
            m_ += A[i,j]
    m_ = m_/2.0
    print "m: %f" % m_
  Orig_deg = {}
    Orig_deg = UpdateDeg(A)
    newmang(G, Orig_deg, m_)
if __name__ == "__main__":
    sys.exit(main(sys.argv))
```

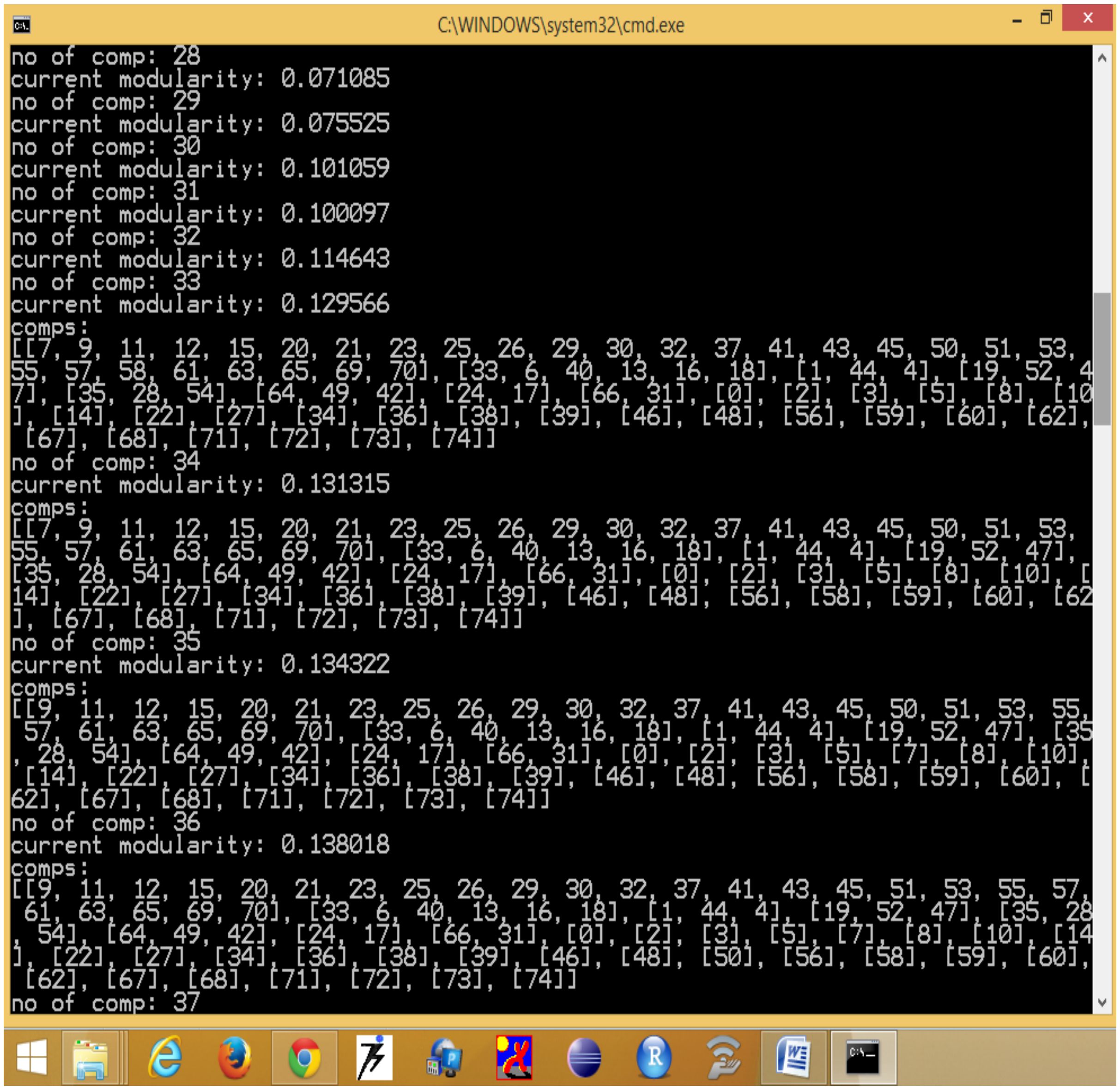
4.2) Output:



C:\WINDOWS\system32\cmd.exe

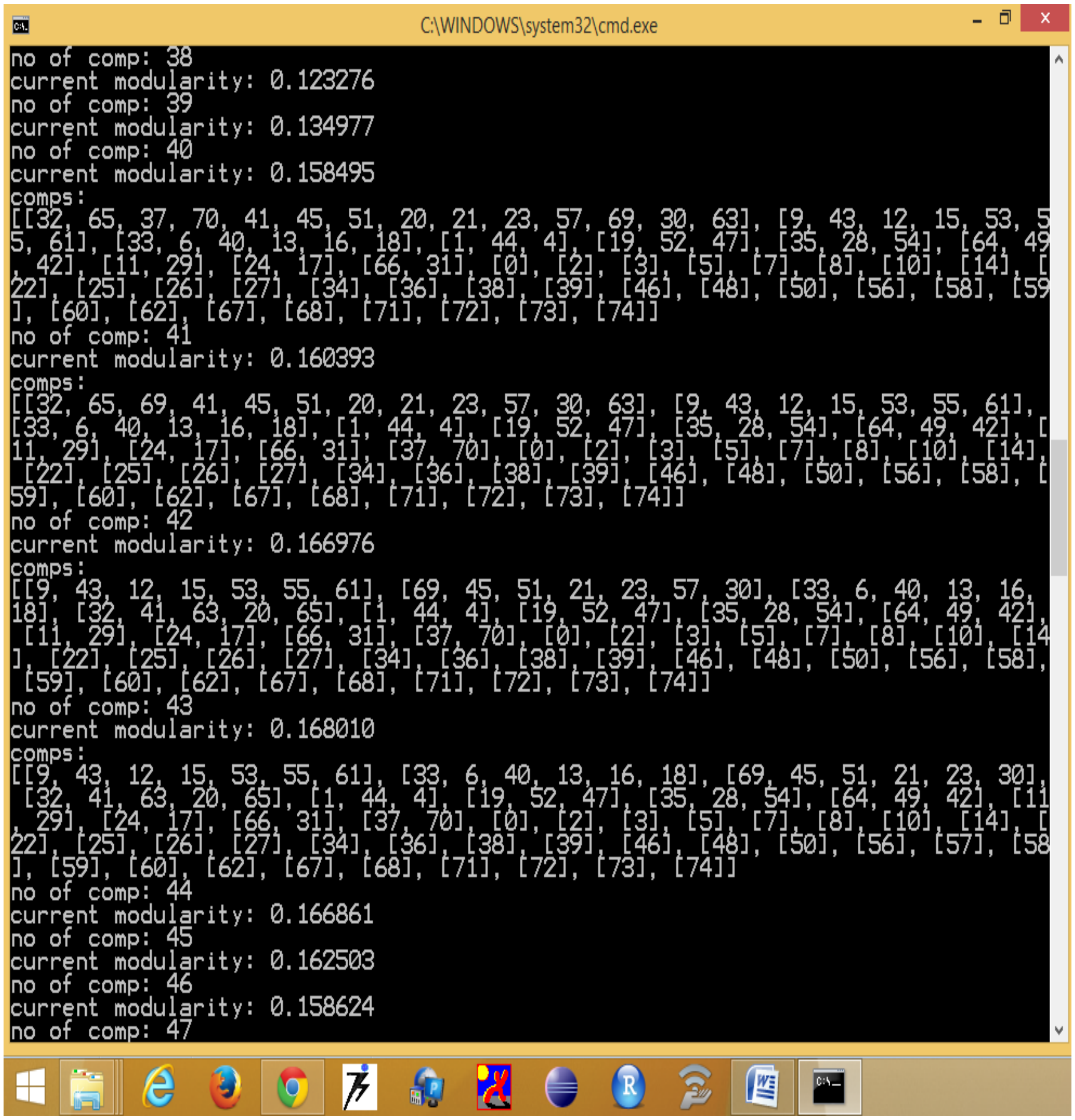
```
C:\>python label.py graph.txt
m: 2204020.000000
no of comp: 2
current modularity: -0.000395
no of comp: 3
current modularity: -0.000681
no of comp: 4
current modularity: 0.117084
comps:
[[1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 12, 14, 15, 17, 19, 20, 21, 22, 23, 24, 25, 26
, 27, 28, 29, 30, 31, 32, 34, 35, 36, 37, 38, 39, 41, 42, 43, 44, 45, 47, 48, 49
, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69
, 70, 71, 72, 73, 74], [33, 6, 40, 13, 16, 18], [0], [46]]
no of comp: 5
current modularity: 0.101936
no of comp: 6
current modularity: 0.102519
no of comp: 7
current modularity: 0.100792
no of comp: 8
current modularity: 0.096462
no of comp: 9
current modularity: 0.114342
no of comp: 10
current modularity: 0.092087
no of comp: 11
current modularity: 0.087838
no of comp: 12
current modularity: 0.090557
no of comp: 13
current modularity: 0.063956
no of comp: 14
current modularity: 0.084862
no of comp: 15
current modularity: 0.093598
no of comp: 16
current modularity: 0.093556
no of comp: 17
current modularity: 0.097304
no of comp: 18
current modularity: 0.082934
no of comp: 19
```

Fig 10 a: Output of Girvan and Newman implementation



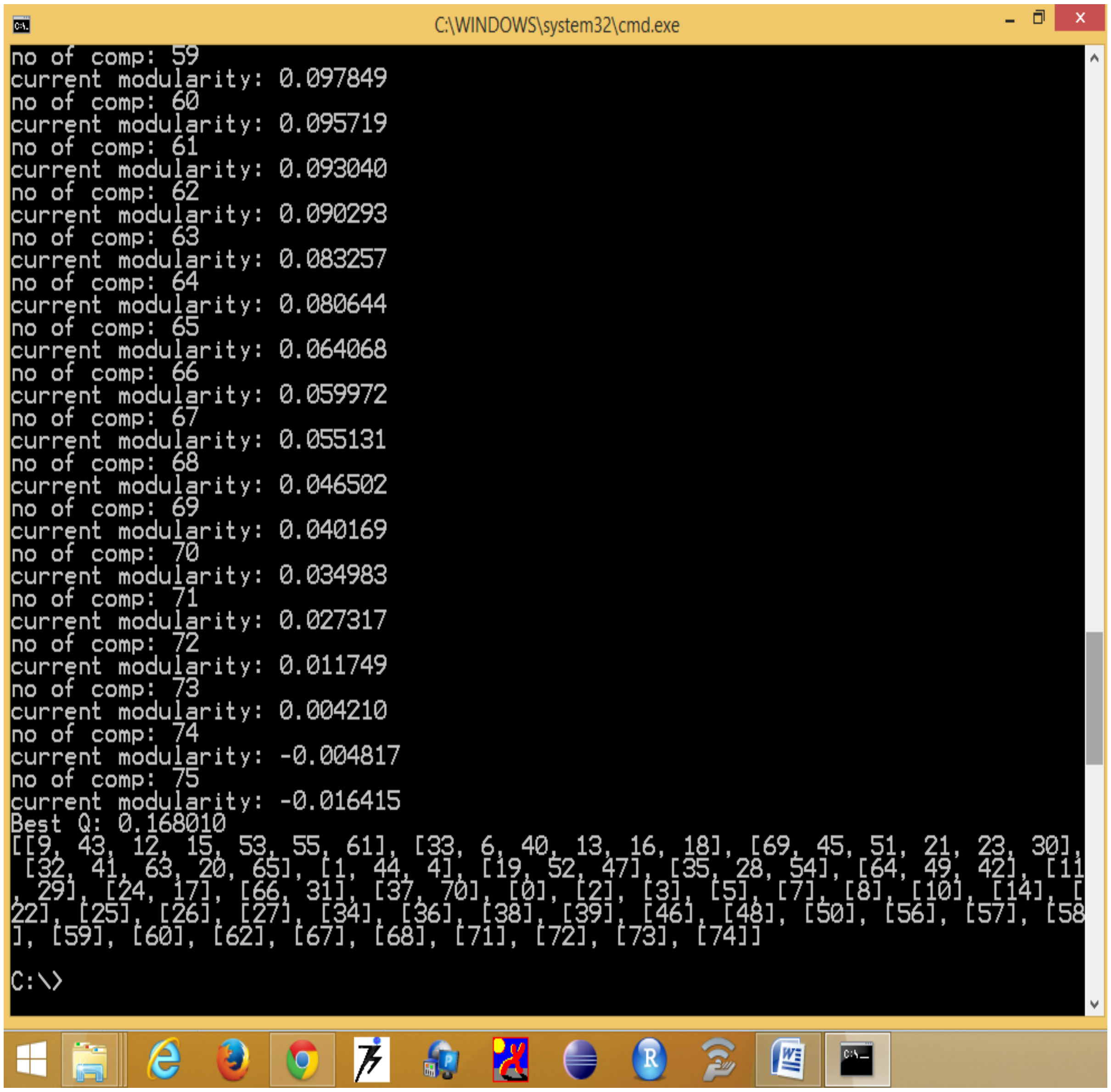
```
no of comp: 28
current modularity: 0.071085
no of comp: 29
current modularity: 0.075525
no of comp: 30
current modularity: 0.101059
no of comp: 31
current modularity: 0.100097
no of comp: 32
current modularity: 0.114643
no of comp: 33
current modularity: 0.129566
comps:
[[7, 9, 11, 12, 15, 20, 21, 23, 25, 26, 29, 30, 32, 37, 41, 43, 45, 50, 51, 53,
55, 57, 58, 61, 63, 65, 69, 70], [33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 4
7], [35, 28, 54], [64, 49, 42], [24, 17], [66, 31], [0], [2], [3], [5], [8], [10
], [14], [22], [27], [34], [36], [38], [39], [46], [48], [56], [59], [60], [62],
 [67], [68], [71], [72], [73], [74]]
no of comp: 34
current modularity: 0.131315
comps:
[[7, 9, 11, 12, 15, 20, 21, 23, 25, 26, 29, 30, 32, 37, 41, 43, 45, 50, 51, 53,
55, 57, 61, 63, 65, 69, 70], [33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 47],
[35, 28, 54], [64, 49, 42], [24, 17], [66, 31], [0], [2], [3], [5], [8], [10], [
14], [22], [27], [34], [36], [38], [39], [46], [48], [56], [58], [59], [60], [62
], [67], [68], [71], [72], [73], [74]]
no of comp: 35
current modularity: 0.134322
comps:
[[9, 11, 12, 15, 20, 21, 23, 25, 26, 29, 30, 32, 37, 41, 43, 45, 50, 51, 53, 55,
 57, 61, 63, 65, 69, 70], [33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 47], [35
, 28, 54], [64, 49, 42], [24, 17], [66, 31], [0], [2], [3], [5], [7], [8], [10],
 [14], [22], [27], [34], [36], [38], [39], [46], [48], [56], [58], [59], [60], [
62], [67], [68], [71], [72], [73], [74]]
no of comp: 36
current modularity: 0.138018
comps:
[[9, 11, 12, 15, 20, 21, 23, 25, 26, 29, 30, 32, 37, 41, 43, 45, 51, 53, 55, 57,
 61, 63, 65, 69, 70], [33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 47], [35, 28
, 54], [64, 49, 42], [24, 17], [66, 31], [0], [2], [3], [5], [7], [8], [10], [14
], [22], [27], [34], [36], [38], [39], [46], [48], [50], [56], [58], [59], [60],
 [62], [67], [68], [71], [72], [73], [74]]
no of comp: 37
```

Fig 10 b: Output of Girvan and Newman implementation



```
no of comp: 38
current modularity: 0.123276
no of comp: 39
current modularity: 0.134977
no of comp: 40
current modularity: 0.158495
comps:
[[32, 65, 37, 70, 41, 45, 51, 20, 21, 23, 57, 69, 30, 63], [9, 43, 12, 15, 53, 5
5, 61], [33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 47], [35, 28, 54], [64, 49
, 42], [11, 29], [24, 17], [66, 31], [0], [2], [3], [5], [7], [8], [10], [14], [
22], [25], [26], [27], [34], [36], [38], [39], [46], [48], [50], [56], [58], [59
], [60], [62], [67], [68], [71], [72], [73], [74]]
no of comp: 41
current modularity: 0.160393
comps:
[[32, 65, 69, 41, 45, 51, 20, 21, 23, 57, 30, 63], [9, 43, 12, 15, 53, 55, 61],
[33, 6, 40, 13, 16, 18], [1, 44, 4], [19, 52, 47], [35, 28, 54], [64, 49, 42], [
11, 29], [24, 17], [66, 31], [37, 70], [0], [2], [3], [5], [7], [8], [10], [14],
 [22], [25], [26], [27], [34], [36], [38], [39], [46], [48], [50], [56], [58], [
59], [60], [62], [67], [68], [71], [72], [73], [74]]
no of comp: 42
current modularity: 0.166976
comps:
[[9, 43, 12, 15, 53, 55, 61], [69, 45, 51, 21, 23, 57, 30], [33, 6, 40, 13, 16,
18], [32, 41, 63, 20, 65], [1, 44, 4], [19, 52, 47], [35, 28, 54], [64, 49, 42],
 [11, 29], [24, 17], [66, 31], [37, 70], [0], [2], [3], [5], [7], [8], [10], [14
], [22], [25], [26], [27], [34], [36], [38], [39], [46], [48], [50], [56], [58],
 [59], [60], [62], [67], [68], [71], [72], [73], [74]]
no of comp: 43
current modularity: 0.168010
comps:
[[9, 43, 12, 15, 53, 55, 61], [33, 6, 40, 13, 16, 18], [69, 45, 51, 21, 23, 30],
 [32, 41, 63, 20, 65], [1, 44, 4], [19, 52, 47], [35, 28, 54], [64, 49, 42], [11
, 29], [24, 17], [66, 31], [37, 70], [0], [2], [3], [5], [7], [8], [10], [14], [
22], [25], [26], [27], [34], [36], [38], [39], [46], [48], [50], [56], [57], [58
], [59], [60], [62], [67], [68], [71], [72], [73], [74]]
no of comp: 44
current modularity: 0.166861
no of comp: 45
current modularity: 0.162503
no of comp: 46
current modularity: 0.158624
no of comp: 47
```

Fig 10 c: Output of Girvan and Newman implementation



```
no of comp: 59
current modularity: 0.097849
no of comp: 60
current modularity: 0.095719
no of comp: 61
current modularity: 0.093040
no of comp: 62
current modularity: 0.090293
no of comp: 63
current modularity: 0.083257
no of comp: 64
current modularity: 0.080644
no of comp: 65
current modularity: 0.064068
no of comp: 66
current modularity: 0.059972
no of comp: 67
current modularity: 0.055131
no of comp: 68
current modularity: 0.046502
no of comp: 69
current modularity: 0.040169
no of comp: 70
current modularity: 0.034983
no of comp: 71
current modularity: 0.027317
no of comp: 72
current modularity: 0.011749
no of comp: 73
current modularity: 0.004210
no of comp: 74
current modularity: -0.004817
no of comp: 75
current modularity: -0.016415
Best Q: 0.168010
[[9, 43, 12, 15, 53, 55, 61], [33, 6, 40, 13, 16, 18], [69, 45, 51, 21, 23, 30],
 [32, 41, 63, 20, 65], [1, 44, 4], [19, 52, 47], [35, 28, 54], [64, 49, 42], [11
, 29], [24, 17], [66, 31], [37, 70], [0], [2], [3], [5], [7], [8], [10], [14], [
22], [25], [26], [27], [34], [36], [38], [39], [46], [48], [50], [56], [57], [58
], [59], [60], [62], [67], [68], [71], [72], [73], [74]]

C:\>
```

Fig 10 d: Output of Girvan and Newman implementation

CONCLUSION

From all the three algorithms i.e. Walktrap, Multilevel, Girvan and Newman it can be concluded that if we want to maximize the modularity of the communities then we should use Multilevel algorithm for community detection. It has also one more advantage of linear complexity. But this algorithm fails to identify communities smaller than a certain scale.

In my opinion, Girvan and Newman is the better algorithm because of many reasons. Firstly, it uses divisive approach which is mainly focused on finding the least similar connected pairs of vertices and then remove the edges between them. By doing this repeatedly, we divide the network into smaller and smaller components. Secondly, it has complexity of O(mn) where m represents edges and n represents nodes. Third, it is able to identify all the possible communities. Fourth, it can optimally run with billions of nodes as it reduces the network into smaller ones at every iteration.

REFERENCES:

Books:

1. Introduction to social network methods by Robert A. Hanneman and Mark Riddle
2. Social Network Analysis: Methods and Applications by Stanley Wasserman , Katherine Faust
3. The SAGE Handbook of Social Network Analysis by John Scott, Peter J. Carrington

Research Papers:

1. Pascal Pons and Matthieu Latapy "Computing communities in large networks using random walks"
2. Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte1, and Etienne Lefebvre " Fast unfolding of communities in large networks"
3. M. E. J. Newman1, 2 and M. Girvan2," Finding and evaluating community structure in networks"
4. Santo Fortunato " Community detection in graphs"
5. Michel Planti´e and Michel Crampes " Survey on Social Community Detection"
6. Sergi Lozano , Jordi Duch and Alex Arenas "Community detection in a large social dataset of European Projects "
7. Jiyang Chen, Osmar R. Zaiane, Randy Goebel "Detecting Communities in Social Networks using Max-Min Modularity"

Web Pages:

1. http://rdatamining.wordpress.com
2. http://www.academia.edu/239537
3. http://www.cyclismo.org/tutorial/R
4. https://docs.python.org/2/tutorial
5. http://www.stanford.edu/~messing/RforSNA.html