Scalable Community Detection and Centrality Algorithms for Network Analysis

A THESIS

submitted by

VISHNU SANKAR M

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

NOVEMBER 2014

THESIS CERTIFICATE

This is to certify that the thesis titled **Scalable Community Detection and Centrality Algorithms for Network Analysis**, submitted by **Vishnu Sankar M**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B Ravindran
Research Guide
Associate Professor
Dept. of CSE
IIT-Madras, 600 036

Place: Chennai

Date: April 1, 2015

ACKNOWLEDGEMENTS

I take this opportunity to thank my research guide Dr. B. Ravindran for his friendly guidance and inputs during the course of my research. He organized weekly lab meetings which served as a forum to be aware of the ongoing research in the lab.

I thank Ericcson for funding my research and Mr.Shivashankar of Ericsson for helping me getting a patent out of it.

I thank my colleague Swarnim with whom I collaborated on the topic “comparison of different scoring functions". This served as the base, from which my research led me to come up with a new scoring function.

Lastly, I thank my family and friends for supporting me throughout my research.

ABSTRACT

KEYWORDS: Social Networks ; Communities ; Scoring functions; Modularity; Hadoop; Game Theory; Centrality

Communication has become a lot easier with the advent of easy and cheap means of reaching people across the globe. This has allowed the development of large networked communities and, with the technology available to track them, has opened up the study of social networks at unprecedented scales. This has necessitated the scaling up of various network analysis algorithms that have been proposed earlier in the literature. While some algorithms can be readily adapted to large networks, in many cases the adaptation is not trivial.

Real-world networks typically exhibit non-uniform edge densities with there being a higher concentration of edges within modules or communities. Various scoring functions have been proposed to quantify the quality of such communities. The popular scoring functions suffer from certain limitations. This thesis identifies the necessary features that a scoring function should incorporate in order to characterize good community structure and propose a new scoring function, CEIL (Community detection using External and Internal scores in Large networks), which conforms closely with the characterization. It also demonstrates experimentally the superiority of CEIL score over the existing scoring functions. Modularity, a very popular scoring function, exhibits a resolution limit, i.e., one cannot find communities that are much smaller in size compared to the size of the network. In many real world networks, community size does not grow in proportion to the network size. This implies that resolution limit is a serious prob-

lem in large networks. Still modularity is very popular since it offers many advantages such as fast algorithms for maximizing the score, and non-trivial community structures corresponding to the maxima. This thesis shows analytically that the CEIL score does not suffer from resolution limit. It also modifies the Louvain method, one of the fastest greedy algorithms for maximizing modularity, to maximize the CEIL score. The modified algorithm, known as CEIL algorithm, gives the expected communities in synthetic networks as opposed to maximizing modularity. The community labels given by CEIL algorithm matches closely with the ground-truth community labels in real world networks. It is on par with Louvain method in computation time and hence scales well to large networks.

This thesis also explores the scaling up of a class of node centrality algorithms based on cooperative game theory which were proposed earlier as efficient alternatives to traditional measure of information diffusion centrality. It presents the distributed versions of these algorithms in a Map-Reduce framework, currently the most popular distributed computing paradigm. The scaling behavior of the distributed version of algorithms on large synthetic networks is demonstrated empirically thereby establishing the utility of these methods in settings such as online social networks.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS **i**

ABSTRACT **ii**

LIST OF TABLES **vi**

LIST OF FIGURES **ix**

1 Introduction **1**

1.1 Need for a new community detection algorithm 2

1.2 Need for parallelizing game theoretic centrality algorithms 3

1.3 Contribution of the thesis 3

1.4 Outline of the thesis 4

2 Background and Related Work **5**

2.1 Related algorithms to find communities 5

2.2 Related measures of centrality 10

3 Introduction and Analysis of CEIL Score **11**

3.1 Introduction of CEIL score 11

3.2 Analysis of CEIL score 21

3.3 Comparison of several scoring functions 27

4 CEIL Algorithm to Find Communities **35**

4.1 CEIL Algorithm 35

4.2 Empirical validation of CEIL algorithm 39

5 Parallelization of Centrality Algorithms **47**

5.1 Map-Reduce and Hadoop 47

5.2 Parallelization of game theoretic centrality algorithms 51

5.3 Experimental Results 68

6 Conclusions and Future Work **76**

A Results of perturbation experiments **77**

LIST OF TABLES

3.1 Features that are captured by different scoring functions 21

3.2 Networks with ground-truth communities 28

3.3 Absolute difference in Z score between the large and small perturbation. Best scores are bolded. 32

3.4 Spearman's rank correlation coefficient for density 33

3.5 Spearman's rank correlation coefficient for separability 33

4.1 Rand index 45

4.2 Running Time 45

5.1 Running time of Game 1 on E-R graphs of different densities in seconds 73

5.2 Running time of Game 2 on E-R graphs of different densities in seconds 73

5.3 Running time of Game 3 on E-R graphs of different densities in seconds 73

5.4 Running time of Game 4 on E-R graphs of different densities in seconds 74

5.5 Running time of Game 5 on E-R graphs of different densities in seconds 74

5.6 Running time of all the game theoretic algorithms on Barabasi Albert graphs of density 0.1 in seconds 74

5.7 Running time of game 1, game 2 and game 3 algorithms in seconds on a 10 million edge network 75

5.8 Running time of game 1, game 2 and game 3 algorithms in seconds on a 1 million edge network 75

LIST OF FIGURES

3.1 Communities differing only in the parameter 'Number of nodes forming the community' 12

3.2 Communities differing only in the parameter 'Number of intra community edges present in the community' 12

3.3 Communities differing only in the parameter 'Number of inter community edges incident on the community' 12

3.4 An example network with 5 communities 14

3.5 An example network with 2 communities 19

3.6 Outline of a network with atleast 3 communities 23

3.7 Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under NodeSwap perturbation. . . . 30

3.8 Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Random perturbation. 30

3.9 Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Expand perturbation. 31

3.10 Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Shrink perturbation. 31

4.1 The network on the left side is an example network with 16 nodes. At the end of the first phase, which is the maximization of CEIL score, four communities are formed. In the diagram at the top, they are marked by four different colors. The second phase is to construct the induced graph using the labels of the first phase. It reduced the graph to 4 nodes. The numbers inside the parenthesis are the properties of the nodes - Number of intra community edges, Sum of degree of all nodes and Number of nodes of the community in the original graph which the node represents in the induced graph. One pass represents one iteration of the algorithm. 37

4.2 The second pass reduces the graph to two nodes. After this, merging of nodes decreases the score given by CEIL score to the network. So, the algorithm stops. 38

4.3 Circle of Cliques 41

4.4 Two Pair of Cliques 42

5.1 Map-Reduce model with 3 mappers and 2 reducers 48

5.2 An example network which is unweighted 49

5.3 An example network which is weighted 50

5.4 Result of cascade experiment in collaboration network 71

A.1 Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under NodeSwap perturbation. 77

A.2 Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Random perturbation. 78

A.3 Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Expand perturbation. 78

A.4 Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Shrink perturbation. 79

A.5 Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under NodeSwap perturbation. 79

A.6 Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Random perturbation. 80

A.7 Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Expand perturbation. 80

A.8 Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Shrink perturbation. 81

A.9 Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under NodeSwap perturbation. 81

A.10 Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Random perturbation. 82

A.11 Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Expand perturbation. 82

A.12 Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Shrink perturbation. 83

CHAPTER 1

Introduction

Interactions between people in social media, telephone calls between phone users, collaboration between scientists and several other social interactions have significant correlation with the behavior of people. Decisions taken by people are influenced by their social interactions. Such interactions are frequently modeled using a graph. In social domain, people form nodes and the interactions between them form edges [15]. In telecom domain, phone users form nodes and calls made by them form edges [17]. In all of these networks, finding groups of nodes which have similar characteristics and finding important nodes in the network have been among the top research interests.

Easy connectivity across the globe has resulted in a large number of interactions between people. The possibility of collection, storage and processing of this huge volume of data has allowed the construction and study of large networks. This has necessitated the improvements in the algorithms that are used to analyze the properties of the network. Most of the algorithms are inherently non scalable. This thesis considers two major areas of network analysis namely community detection and centrality and improves over the existing algorithms in those two areas.

Many real world networks exhibit the property of community structure [20], i.e., nodes in the network can be partitioned into communities such that more edges are present between nodes belonging to the same community than between the nodes belonging to different communities. This structural property has attracted researchers as the communities in networks typically correspond to some real world property of the network. For instance, communities in a collaboration network correspond to research area of the members forming that community.

Finding important nodes/edges in the network is one of the chief challenges addressed by social network analysis. The measure of importance of nodes/edges, known as centrality, varies depending on the context. Generally, a centrality measure which is optimal in one of the context will be sub-optimal in a different context. This structural property is significant in applications such as viral marketing where the initial set of influencers are crucial for the success, modeling the epidemic spread and identifying critical nodes in power systems where the failure of these critical nodes may create a cascading failure.

1.1 Need for a new community detection algorithm

Most of the community detection algorithms reported in the literature do not scale well. The Louvain method [2] is the fastest known algorithm to find communities. But modularity [22], the objective function which the Louvain method optimizes, suffers from the problem of resolution limit, i.e., one cannot detect communities which are much smaller in size when compared to the size of the network using modularity. For instance, consider two communities which are connected to each other by a single edge. Also, each of them are connected to rest of the network by a single edge. Modularity maximization cannot find communities which are of size less than $\sqrt{\frac{L}{2}}$ in this case where L represents the total number of edges in the network. In many real world networks, community size does not grow in proportion to the network size [13]. This implies that resolution limit is a serious problem in large networks. Conductance is another popular scoring function used to measure the quality of communities. But, optimization of conductance will always give one community which is the graph itself and hence it is used mainly as a relative comparative measure and not as an objective function. Surprise is another scoring function which does not suffer from resolution limit but is computationally intensive. Label propagation algorithm proposed by Raghavan et al. is simple and each iteration

takes little time. But the number of iterations is prohibitively large. Clique percolation technique is another popular algorithm but is mainly used to find the overlapping communities. The iterative ensemble method proposed by Ovelgonne and Schulz finds natural communities but is computationally very expensive. Hence, there is a need for a newer community detection algorithm which is free from resolution limit, yet easy to compute and scalable.

1.2 Need for parallelizing game theoretic centrality algorithms

The traditional centrality measures including betweenness centrality, closeness centrality and eigen vector centrality measures fail to identify the collective importance of several nodes. Game theoretic centrality measures proposed in [19], [14] overcome this problem and have polynomial time complexity. Hadoop is a open source implementation of the widely used map-reduce parallelization framework. Though these algorithms are inherently parallelizable, hadoop is not the ideal platform for parallelizing graph algorithms. Since hadoop is widely used, we have used it to parallelize five of the game theoretic centrality measures which has allowed us to have a better scaling behavior in these algorithms.

1.3 Contribution of the thesis

The contributions of this thesis are :

- Identification of the necessary features that a good scoring function should incorporate in order to find the relative ordering of communities in a network.

- Introduction of a new scoring function which captures all the features of communities and a theoretical proof that this scoring function does not suffer from resolution limit.
- A good alternative algorithm to Louvain method for finding communities in large networks.
- Parallelization of the polynomial time game-theoretic algorithms for finding centralities of nodes in large networks.

1.4 Outline of the thesis

Rest of this thesis is organized as follows:

- Chapter 2 introduces the related community detection algorithms and the existing centrality measures.
- Chapter 3 characterizes the necessary features that are required for a good scoring function, proposes a new scoring function which confirms to the characterization and compares it with the existing scoring functions.
- Chapter 4 presents an algorithm to find communities and applies it on synthetic and real world networks and compares it's performance with other algorithms.
- Chapter 5 talks about the parallelization of the game theoretic centrality algorithms using hadoop.
- Chapter 6 gives a summary of the work done.

CHAPTER 2

Background and Related Work

In this chapter, we discuss about the various techniques which are used to find communities in networks and the existing measures of centrality.

2.1 Related algorithms to find communities

Existing approaches to find communities

Several approaches are existing to find communities in a network. We describe most of the widely used methods in this section.

Divisive Algorithms : Inter community edges are the edges which go from a node belonging to one community to another node which belongs to a different community. Divisive algorithms identifies such edges and removes them one by one. The algorithm can either have a stopping criteria or remove all the edges and then construct a dendogram using the order of removal of edges.

Agglomeration Algorithms : In agglomeration algorithms, each node in the network is assumed to be of individual community i.e., we will have as many communities as the number of nodes in the network. Every community is merged with the neighboring communities based on a criteria. The merging of communities is stopped once the stopping criteria is reached.

Random Walk : In random walk based methods, similarity between vertices are calculated based on the probability of a random walker choosing that path. This probability

will generally be high for vertices which are closer than the ones farther apart. The similarity score is then used to find communities by either divisive or agglomeration technique.

Spectral Methods : Spectral methods transform the adjacency matrix of the network into a suitable form. Then, it uses the values of the eigen vectors of this matrix to find communities.

Optimization of an objective function : Scoring functions came into existence to evaluate the goodness of the communities given by various community detection algorithms. Scoring functions quantifies the quality of communities i.e., they differentiate good and bad communities by assigning them a score. A partition of the network having the best score according to a scoring function is the best partition of the network. Finding the partition which gets the best score according to scoring functions among all such partitions is an NP-hard problem. So, this class of algorithms find communities by greedy optimization of the scoring functions.

Existing algorithms to find communities

Girvan and Newman proposed a divisive algorithm which uses betweenness score to find the inter community edges [21]. Betweenness score for an edge is calculated based on the fraction of the shortest path between all pairs of nodes in the network that contains this edge. Inter community edges will have high betweenness score as it will be the part of many shortest paths in the network. So, in this algorithm, the edge with the highest betweenness score is removed. Then, the betweenness scores of the edges affected due to the removal of this edge are recalculated. This process is repeated until there is no edge remaining in the graph and the order of removal of edges is used to built the dendogram. This method suffers from high computational cost. The algorithm runs

with a worst case time complexity of O(m^2n) where m is the number of edges in the network and n is the number of nodes in the network. This algorithm is parallelizable and the parallel version [33] has an improved running time.

Newman proposed another algorithm which greedily optimizes a scoring function named moudlarity [22]. The exact maximization of modularity is hard [4]. It falls into the general category of agglomerative hierarchical clustering. Initially, all the nodes in the network are considered to be a community on its own. Then, the pairs of communities are merged which will give a greatest increase or smallest decrease to the modularity score of the network. Then, the order of merging of communities is used to built the dendogram. Partition at every level of this dendogram gives different number of communities. But, the algorithm selects the partition that gives the maximal value of modularity score. This running time of this algorithm is O($(n + m)n$). The increase in speed of this algorithm over the betweenness algorithm can be partly attributed to the faster calculation of change in the modularity value whenever two communities are merged.

A randomized greedy algorithm whose aim was to improve the running time of the plain greedy algorithm without much decline in the performance was proposed by Ovelgonne and Geyer-Schulz [24]. This is also an agglomerative hierarchical clustering wherein one best pair of nodes are merged after every iteration. The randomized greedy algorithm chooses k random nodes from the network and searches for best pair of nodes to be merged only from among the random nodes and their neighbors and not from all pair of nodes in the network like the plain greedy algorithm. This method improves the running time of the algorithm and as well makes it stochastic.

Ovelgonne and Geyer-Schulz also proposed an ensemble method to find communities using the randomized greedy algorithm [24]. Initially, a set S of k partitions of the same network G is obtained by running the randomized greedy algorithm k times.

Then, the partition $\hat{P}$ is formed which is the maximal overlap of all the partitions in set S. An induced graph $\hat{G}$ is formed from the partition $\hat{P}$. A final algorithm is used on this induced graph to find the communities of $\hat{G}$ which is then projected back on to G. Note that the initial set S can be constructed either by running any stochastic algorithm k times or by running k deterministic algorithms or by any combination of both which makes this a generic framework. They also proposed an iterative ensemble method to improve the performance of this algorithm by repeating the steps before the final algorithm is run [25].

A very fast method to maximize modularity was introduced by Blondel et al. [2]. This method uses a simple heuristic to maximize modularity. This method consists of two phases. Initially, all the nodes in the network are assigned to its own community. Then, every node in the network is considered in a sequential order and are merged with their neighbors such that it gives a maximal increase to the modularity score of the partition. This process is repeated until the modularity value reaches a local maxima. In the second phase, an induced graph is constructed using the partition obtained from the first phase. This induced graph can be given as input to the first phase and thus the two phases are iterated until the modularity value of the partition reaches a maximal value.

Modularity maximization to find communities can also be done using spectral methods [23]. Let A be the adjacency matrix of the network, D be the diagonal matrix with degree of nodes in the diagonal and $L = D^{-frac12}AD^{-frac12}$ be the normalized Laplacian matrix. The eigen vector x corresponding to the second largest eigen value λ is considered. The network is partitioned into two groups based on the signs of the elements of x. This process is repeated and a dendogram is built.

Despite several methods were developed to find communities by maximizing modularity, optimization of modularity was proven to suffer from resolution limit [8]. Attempts were made to circumvent resolution limit based on the concept of multi-resolution

[1]. But, Lancichinetti and Fortunato proved that multi-resolution suffers from two opposite co-existing problems [12]. At low resolution, small sub graphs will be merged. At high resolution, large sub graphs will get split. Since the resolution has to be either low or high at a time, multi-resolution algorithms will always suffer from at least one of the two problems.

Tanmoy et al. proposed a novel approach to find communities by taking into consideration of not only the number of inter community edges going out of a community but also the distribution of inter community edges to all the neighboring communities [5]. This method is proved empirically to produce a better community structure than modularity maximization algorithms.

Label propagation method was proposed by Raghavan et al. [29]. In this method, all the nodes are assigned an unique label at the initial step. Then, every node in the network is considered in a random order and it is assigned the label which majority of its neighbors have. This process is repeated until all the nodes in the network gets a label which at least half of its neighbors have.

Pons and Latapy proposed a random walk based method to find communities [27]. They find the similarity of nodes based on the random walk and then use this knowledge to build a hierarchical agglomeration algorithm.

Palla et al. proposed a clique percolation technique which is mainly used to find overlapping communities in networks [26]. Two k-cliques are said to be adjacent if they share k-1 nodes between them. The maximal union of such k-cliques that are adjacent to each other is defined as a community. Though this method is computationally expensive, it uncovers the natural overlap between communities as a node is allowed to part of any number of communities.

2.2 Related measures of centrality

The popular centrality measures are degree centrality, closeness centrality, betweenness centrality and page rank centrality.

Degree centrality of a node is the number of links incident upon that node. It is useful in the context of finding the single node which gets affected by the diffusion of any information in the network. It follows from the fact that the node with high degree centrality has the chance of getting affected from many number of sources.

Closeness centrality of a node is the sum of the inverse of the shortest distance from the node to every other node in the network. In applications such as package delivery, a node with high closeness centrality can be considered as the central point.

Betweenness centrality of an edge is the fraction of shortest paths between all pairs of nodes in the network that passes through this edge. A node with high betweenness centrality will typically be the one which acts as a bridge between many pairs of nodes.

Page rank centrality of a node depends on the number and the quality of the neighbors who have links to the node. One of the popular application of page rank centrality is finding the relevant page from the web. Initially, all the web pages are assumed to have an equal page rank value. Then, page rank value of every web page is divided equally to each of the outgoing links from that web page. Page rank value of a web page after the above step is the sum of the values obtained from all the incoming links to that web page. This process is repeated iteratively until the page rank values of all the web pages do not change by a large margin in any two subsequent iterations.

CHAPTER 3

Introduction and Analysis of CEIL Score

In this chapter, we characterize the features of a good scoring function, propose a new scoring which confirms to our characterization and compare it with the existing scoring functions. The intuition behind a community is that it should be well connected internally and well separated from rest of the network, i.e., it should have more number of intra community edges and less number of inter community edges. A scoring function is the mathematical formulation of the quality of a community.

3.1 Introduction of CEIL score

In this section, we introduce our new scoring function which we call as CEIL(Community detection using External and Internal scores in Large networks).

Characterization of a good scoring function

A community is characterized by the following three features.

- Number of nodes forming the community.
- Number of intra community edges present in the community.
- Number of inter community edges incident on the community.

A scoring function should incorporate all these three features of the communities because numerous examples can be constructed such that there are two communities in

(a) Community A (b) Community B

Figure 3.1: Communities differing only in the parameter ‘Number of nodes forming the community’

(a) Community A (b) Community B

Figure 3.2: Communities differing only in the parameter ‘Number of intra community edges present in the community’

(a) Community A (b) Community B

Figure 3.3: Communities differing only in the parameter ‘Number of inter community edges incident on the community’

the same network which differ in only one of the above three features in which case the third feature becomes necessary to distinguish between those two communities. For example, consider a scoring function which do not include the feature ‘number of nodes forming the community’. It will give same score to a community A in a network having 10 intra community edges, 2 inter community edges and 5 nodes and to a community B

in the same network having 10 intra community edges, 2 inter community edges and 10 nodes. But community A has higher internal density than community B and should be given higher score than community B. Similarly, other two features are also essential in assigning scores to communities in order to find the relative order of communities in a network.

Notations

$G -$ A simple, unweighted network.

$N -$ Number of nodes in G.

$E -$ Number of edges in G.

$L -$ Set of edges in G.

$S -$ Set of all communities in G.

$C -$ Number of communities in G.

$s -$ One of the communities in S.

$a_s -$ Number of intra community edges in s.

$b_s -$ Number of inter community edges incident on s.

$n_s -$ Number of nodes forming the community s.

$\xi -$ Community link matrix for communities in G.

Existing scoring functions

Few of the widely used scoring functions are conductance, triad participation ratio and modularity.

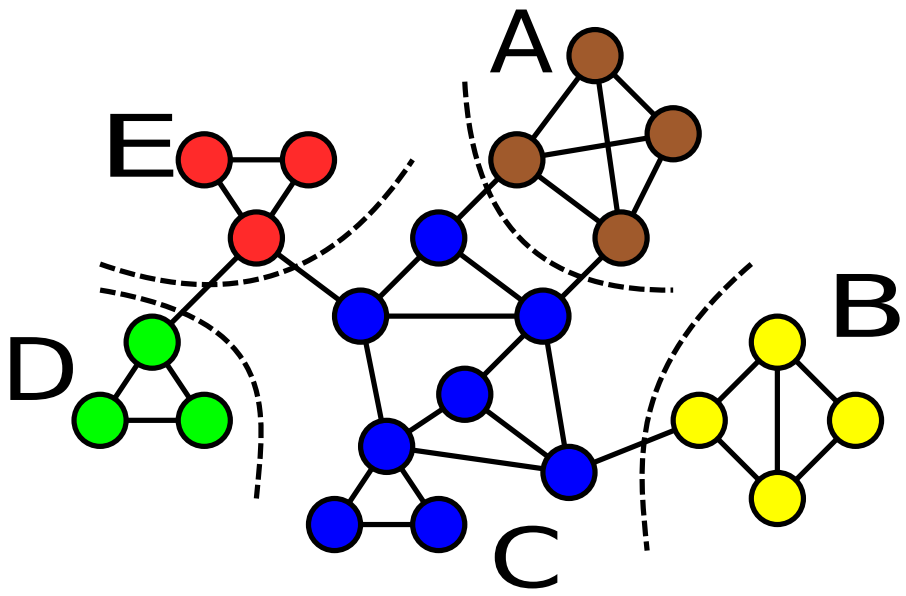

Figure 3.4: An example network with 5 communities

Definition 1. *Conductance is given by the ratio of number of inter community edges to the total degree of all nodes in the community [31].*

$$\text{Conductance}(s) = \frac{b_s}{2a_s + b_s}$$

Conductance ranges from 0 to 1 with 0 representing the good community.

In Figure 3.4,

For community A,

a_s = 6

b_s = 2

Conductance(A) = $\frac{2}{14}$ = 0.1428

For community B,

a_s = 5

b_s = 1

Conductance(B) = $\frac{1}{11}$ = 0.0909

Definition 2. *Triad Participation Ratio is given by the fraction of nodes in the community that are participating in at least one triad.*

$$\text{Triad Participation Ratio}(s) = \frac{|\{u : u \in s, \{(v,w) : v,w \in s, (u,v) \in L, (u,w) \in L, (v,w) \in L\} \neq \varnothing\}|}{n_s}$$

Triad participation ratio ranges from 0 to 1 with 1 representing the good community. In Figure 3.4,

For community A,

Number of nodes participating in at least one triad = 4

Total number of nodes = 4

TPR(A) = $\frac{4}{4}$ = 1

For community B,

Number of nodes participating in at least one triad = 3

Total number of nodes = 4

TPR(B) = $\frac{3}{4}$ = 0.75

Definition 3. *Modularity is given by the difference between the actual number of intra community edges present in the community and the expected number of intra community edges that would be present in the community if the edges in the network are distributed randomly with identical degree distribution [22].*

$$\text{Modularity}(s) = \frac{a_s}{E} - \left(\frac{2a_s + b_s}{2E}\right)^2$$

Modularity ranges from -0.5 to 1 with 1 representing the good community. A modularity value of 0 indicates that the edges in the network are uniformly distributed and there is

no community structure in the network.

In Figure 3.4,

For community A,

a_s = 6

b_s = 2

E = 34

Modularity(A) = $\frac{6}{34} - \left(\frac{14}{68}\right)^2$ = 0.1340

For community B,

a_s = 5

b_s = 1

E = 34

Modularity(B) = $\frac{5}{34} - \left(\frac{11}{68}\right)^2$ = 0.1208

We find that none of these existing scoring functions take into account all the three necessary features to characterize the communities. Modularity and conductance do not consider the parameter 'number of nodes forming the community' while Triad Participation Ratio does not consider the parameter 'number of inter community edges incident on the community'. So, we propose a new scoring function characterizing the communities by taking into account all the necessary features enabling us to rank the communities in a network better than the existing scoring functions.

CEIL score

A community is expected to be well connected within itself as well as well separated from rest of the network. We define internal score to model the connectedness within the community and external score to model its separability from rest of the network.

A community is said to be well connected internally, if a large fraction of pair of

nodes belonging to the community are connected, i.e., internal well connectedness of a community increases with the increase in number of intra community edges.

Definition 4. *Internal score of a community is the internal edge density of the community.*

$$\text{Internal Score}(s) = \begin{cases} \frac{a_s}{\binom{n_s}{2}} & \text{if } n_s \geq 2 \\ 0 & \text{if } n_s = 1 \end{cases}$$

Internal score ranges from 0 to 1. It takes the value of 0 when there are no intra community edges in the community and takes the value of 1 when every node in the community is connected to every other node in the community.

A community is said to be well separated from rest of the network, if the number of connections between nodes belonging to the community and to those belonging to other communities is less, i.e., external well separability increases with decrease in the number of inter community edges.

Definition 5. *External score of a community is the fraction of total number of edges that are intra community edges in the community.*

$$\text{External Score}(s) = \frac{a_s}{a_s + b_s}$$

External score ranges from 0 to 1. It takes the value of 0 when every edge which is incident on the community is an inter community edge and takes the value of 1 when every edge which is incident on the community is an intra community edge.

CEIL score of a community is the score given to one community in the network. A community structure is said to be good, if it is well connected within itself and is well separated from rest of the network, i.e., if it gets a high value in internal and external

scores. So, we chose the geometric mean of the internal and external score as community score. Since we are interested in the relative order of communities and not on the absolute score a community gets, we chose product over geometric mean.

Definition 6. *CEIL score of a community is product of internal and external score of the community.*

$$\text{CEIL Score}(s) = \text{Internal Score}(s) \times \text{External Score}(s) \tag{3.1}$$

CEIL score of a community ranges from 0 to 1. It takes the value of 0 when the community has no intra community edge and takes the value of 1 when the community is a clique as well as it has no inter community edge. A high community score represents a good community as it can be obtained only when both the internal and external scores are high.

A network typically has many communities.

Definition 7. *CEIL score of a network is the weighted sum of scores of all the communities in the network.*

$$\text{CEIL Score}(G, S) = \sum_{s \in S} \frac{n_s}{n} \times \text{Community Score}(s) \tag{3.2}$$

CEIL score of a network ranges from 0 to 1 in a simple, unweighted network.

Lemma 1. *Computational complexity of CEIL score is $O(E + C)$ and it can be approximated as $O(E)$.*

Proof. Computationally, CEIL score has the same asymptotic complexity as that of modularity and conductance. We need to calculate an additional parameter for each

community namely the number of nodes belonging to the community. A community link matrix ξ is a $C \times C$ matrix where the entries represent the number of edges going from one community to another. The diagonal elements of ξ represent the intra community edges while non-diagonal elements represent the inter community edges in the network. It takes $O(E)$ computation to compute ξ from a labeled network and $O(C)$ computation to compute the CEIL score of the network from the community link matrix. The additional parameter, 'number of nodes forming the community' can be calculated along with the calculation of the community link matrix. So, the computation of the network score takes $O(E + C)$ which can be approximated as $O(E)$ as the number of communities in most of the networks is very small when compared to the number of edges in the network. □

Example calculation of CEIL score

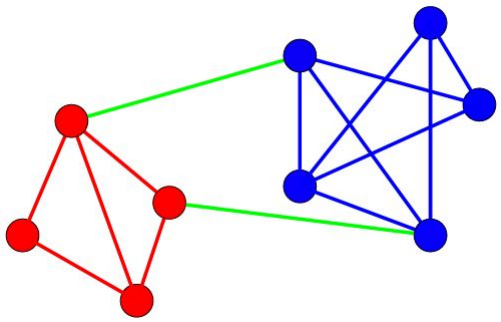

Figure 3.5: An example network with 2 communities

In Figure 3.5, all dots represent nodes and all lines represent edges of the network. This network consists of two communities - community 1 and community 2. Red dots belong to community 1 while blue dots belong to community 2. Red edges are intra community edges of community 1 while blue edges are intra community edges of community 2. Green edges are the inter community edges between community 1 and

community 2. Now,

- Number of nodes in the network(N) = 9
- Number of intra community edges in community 1(a_1) = 5
- Number of inter community edges in community 1(b_1) = 2
- Number of nodes in community 1(n_1) = 4
- Number of intra community edges in community 2(a_2) = 8
- Number of inter community edges in community 2(b_2) = 2
- Number of nodes in community 2(n_2) = 5

$$\begin{aligned}\text{CEIL score of community 1} &= \left(\frac{a_1}{\binom{n_1}{2}}\right)\frac{a_1}{a_1+b_1}\\ &= \left(\frac{5}{6}\right)\left(\frac{5}{7}\right)\\ &= 0.5952\end{aligned}$$

$$\begin{aligned}\text{CEIL score of community 2} &= \left(\frac{a_2}{\binom{n_2}{2}}\right)\frac{a_2}{a_2+b_2}\\ &= \left(\frac{8}{10}\right)\left(\frac{8}{10}\right)\\ &= 0.64\end{aligned}$$

$$
\begin{aligned}
\text{CEIL score of the network} &= \sum_{s \in S} \frac{n_s}{n} \times \text{CEIL Score}(s) \\
&= \left(\frac{4}{9} * 0.5952\right) + \left(\frac{5}{9} * 0.64\right) \\
&= 0.2645 + 0.36 \\
&= 0.6245
\end{aligned}
$$

Table 3.1: Features that are captured by different scoring functions

Scoring Functions	Intra community edges	Inter community edges	Number of nodes	Scalable	Optimizable	No Resolution limit
CEIL score	✓	✓	✓	✓	✓	✓
Modularity	✓	✓	✗	✓	✓	✗
Conductance	✓	✓	✗	✓	✗	✓
TPR	✓	✗	✓	✗	✓	✓

In Table 3.1, we compare the various desirable features of the different community scoring functions. We note that CEIL score have all the features mentioned in Table 3.1 while other scoring functions do not have all the features.

3.2 Analysis of CEIL score

In this section, we analyze the nature of CEIL score and show that it does not suffer from resolution limit.

Resolution Limit

Resolution limit is the inability of an algorithm to detect communities which are much smaller in size when compared to the size of the network, i.e, there exists a lower limit on the size of the detectable communities as a function of the network size. Modularity maximization suffers from resolution limit [8].

Theorem 2. *CEIL score does not suffer from resolution limit.*

Proof. Consider the same network which was used to prove the resolution limit of modularity maximization. The network consists of two communities - 1 and 2. They are connected to each other as well as to the rest of the network as in Figure 3.6. Let n_1 and a_1 be the number of nodes and the number of intra community edges of community 1. Similarly, let n_2 and a_2 be the number of nodes and the number of intra community edges of community 2. In order to express the number of inter community edges of both the communities in terms of their respective intra community edges, we consider four positive constants x_1, y_1, x_2 and y_2. Let x_1a_1 represent the number of inter community edges going from community 1 to community 2 and y_1a_1 represent the number of inter community edges going from community 1 to rest of the network. Similarly, let x_2a_2 represent the number of inter community edges going from community 2 to community 1 and y_2a_2 represent the number of inter community edges going from community 2 to rest of the network. Let N represent total number of nodes in the network.

We will consider two different partitions of this network. Partition A, in which 1 and 2 are considered as two different communities and Partition B, in which 1 and 2 are considered as a single community. The partition of the rest of the network can be done in anyway but identical in partitions A and B. Let N_A and N_B be the network scores of partitions A and B respectively. Let N_0 be the network score of rest of the network and is same for both the partitions A and B since the partition of rest of the network is

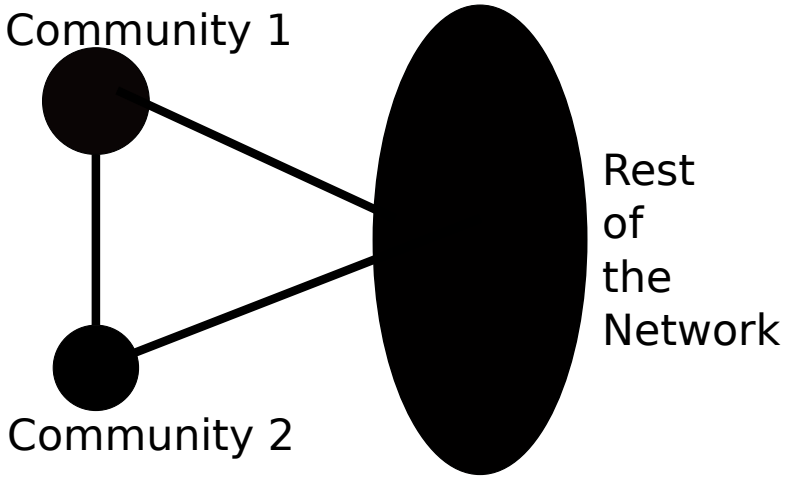

Figure 3.6: Outline of a network with atleast 3 communities

identical in both the cases.

Partition A : 1 and 2 are two different communities in partition A. The properties of community 1 are,

$$\begin{aligned} \text{Intra community edges} &= a_1 \\ \text{Inter community edges} &= x_1 a_1 + y_1 a_1 \\ \text{Number of nodes} &= n_1 \end{aligned}$$

$$\text{CEIL score(1)} = \left(\frac{a_1}{a_1 + x_1 a_1 + y_1 a_1}\right)\left(\frac{a_1}{\binom{n_1}{2}}\right)$$

Similarly,

$$\text{CEIL score(2)} = \left(\frac{a_2}{a_2 + x_2 a_2 + y_2 a_2}\right)\left(\frac{a_2}{\binom{n_2}{2}}\right)$$

CEIL score of the network for partition A is given by,

$$N_A = N_0 + \frac{n_1}{N}\left(\frac{a_1}{a_1 + x_1 a_1 + y_1 a_1}\right)\left(\frac{a_1}{\binom{n_1}{2}}\right) + \frac{n_2}{N}\left(\frac{a_2}{a_2 + x_2 a_2 + y_2 a_2}\right)\left(\frac{a_2}{\binom{n_2}{2}}\right)$$

Partition B : 1 and 2 are considered as a single community. The properties of that community are,

$$\begin{aligned}\text{Intra community edges} &= a_1 + x_1 a_1 + a_2 \\ \text{Inter community edges} &= y_1 a_1 + y_2 a_2 \\ \text{Number of nodes} &= n_1 + n_2\end{aligned}$$

$$\text{CEIL score(community)} = \left(\frac{a_1 + x_1 a_1 + a_2}{a_1 + x_1 a_1 + a_2 + y_1 a_1 + y_2 a_2}\right)\left(\frac{a_1 + x_1 a_1 + a_2}{\binom{(n_1+n_2)}{2}}\right)$$

CEIL score of the network for partition B is given by,

$$N_B = N_0 + \frac{n_1 + n_2}{N}\left(\frac{a_1 + x_1 a_1 + a_2}{a_1 + x_1 a_1 + a_2 + y_1 a_1 + y_2 a_2}\right)\left(\frac{a_1 + x_1 a_1 + a_2}{\binom{(n_1+n_2)}{2}}\right)$$

Comparison of partition A and B : For 1 and 2 to be communities, N_A should be greater than N_B, i.e.,

$$\left(\frac{n_1 a_1^2}{(a_1 + x_1 a_1 + y_1 a_1)\binom{n_1}{2}}\right) + \left(\frac{n_2 a_2^2}{(a_2 + x_2 a_2 + y_2 a_2)\binom{n_2}{2}}\right) > \left(\frac{(n_1 + n_2)(a_1 + x_1 a_1 + a_2)^2}{(a_1 + x_1 a_1 + a_2 + y_1 a_1 + y_2 b_2)\binom{(n_1+n_2)}{2}}\right) \quad (3.3)$$

Note that the above inequality does not depend on any parameter which is related to size of the network. This means that CEIL score does not suffer from resolution limit. □

A similar analysis for modularity has led to the inequality having local parameters on one side and number of edges in the network on the other side [8] i.e., modularity maximization takes the decision to split a community into two or not based on the

relationship between the local parameters of the community and the number of edges in the network.

Nature of CEIL score

We assume, for simplicity in calculations, that communities 1 and 2 in Figure 3.6 are similar, i.e, they have equal number of nodes, equal number of intra community edges and equal number of inter community edges. So, $n_1 = n_2 = n, x_1 = x_2 = x$ and $y_1 = y_2 = y$. This makes (3.3) as,

$$\left(\frac{na^2}{(a+xa+ya)\binom{n}{2}}\right)+\left(\frac{na^2}{(a+xa+ya)\binom{n}{2}}\right) > \left(\frac{(n+n)(a+xa+a)^2}{(a+xa+a+ya+ya)\binom{(n+n)}{2}}\right)$$

On simplification and rearrangement we get,

$$2 \times \frac{2n-1}{n-1} > \frac{(x+2)^2(x+y+1)}{x+2y+2}$$

The term $\frac{2n-1}{n-1}$ takes the minimum value of 2 at $n = \infty$. This makes the expression to be,

$$\frac{(x+2)^2(x+y+1)}{x+2y+2} < 4 \tag{3.4}$$

We will consider several cases for different values of the total number of inter community edges. Our analysis of these cases are in line with the method which was used to prove the resolution limit modularity [8]. However, we differ from them by finding the distribution of inter community edges of the communities 1 and 2 for which they will be found as two different communities. This is because the worst case scenario $x = 2$, $y = 0$ considered to prove the resolution limit in modularity actually connects community 1 and 2 enough to make them into a single community. Also, none of the existing

structural definitions of community discuss the distribution of inter community edges. Hence, we find the constraints to be put in the distribution of inter community edges for which CEIL score is free from resolution limit.

Trivial cases : An isolated community, i.e., one with no inter community edges (x=0 and y=0), would be the strongest community. The expression (3.4) always holds under this condition. This means that the CEIL algorithm will find communities 1 and 2 as two different communities as opposed to combining them together into a single community in the trivial case when both the communities have no inter community edges.

We connect both the communities by a single edge and also connect each of the community to rest of the network by a single edge. This makes both the community to have two inter community edges. Now, $x = \frac{1}{a}$ and $y = \frac{1}{a}$. The expression (3.4) becomes,

$$\frac{\left(\frac{1}{a}+2\right)^2\left(\frac{2}{a}+1\right)}{\frac{3}{a}+2} < 4$$

The above equation holds for all $a \geq 2$. a needs to be positive as it represents the number of intra community edges in the community and $a = 1$ represents a poor community structure as it is make the community to have 1 intra community edge and 2 inter community edges. This means that CEIL score is free from the resolution limit in this case.

Community in strong sense : One of the definitions of a community in strong sense is that the number of inter community edges should not be more than the number of intra community edges. In particular, we consider the extreme case where the number of inter community edges is equal to the number of intra community edges. This makes $x + y = 1$. Upon substituting this and simplifying, expression (3.4) becomes,

$$x^2 + 6x - 4 < 0$$

Considering that x is positive, range of x is [0,0.6]. This means that CEIL algorithm will find communities 1 and 2 to be two different communities if, not more than 60% of the inter community edges go to a single community which is of equal size.

Community in weak sense : A community in weak sense is one in which the number of inter community edges is not more than twice the number of intra community edges [28]. We consider the extreme case where the number of inter community edges is exactly twice the number of intra community edges. This makes $x + y = 2$. Upon substituting this and simplifying, expression (3.4) becomes,

$$3x^2 + 16x - 12 < 0$$

Considering that x is positive, range of x is [0,$\frac{2}{3}$]. This means that CEIL algorithm will find communities 1 and 2 to be two different communities if, not more than 33% of the inter community edges go to a single community which is of equal size.

In the above analysis, we have assumed that the communities 1 and 2 in Figure 3.6 are equal in the number of nodes, the number of intra community edges and the number of inter community edges. They can differ in any or all of these parameters. In these cases, the above constraints will also vary.

3.3 Comparison of several scoring functions

In this section, we compare CEIL score with other popular scoring functions. Yang and Leskovec [32] compared the performance of several scoring functions using perturbation experiments and have reported that Conductance and Triad Participation Ratio are the best performers. Modularity is the most widely used definition. So, we compare CEIL score against these three scoring functions. In addition to the perturbation

experiment, we also compare the scoring functions using a few community goodness metrics.

Datasets

We used four ground truth community datasets from Stanford large network data. The statistics of the networks are listed in Table 3.2.

Table 3.2: Networks with ground-truth communities

Networks	Nodes	Edges	Number of communities
LiveJournal	3,997,962	34,681,189	5000
Youtube	1,134,890	2,987,624	5000
DBLP	317,080	1,049,866	5000
Amazon	334,863	925,872	5000

Perturbation Experiment

Perturbation experiments were introduced for comparative study of different scoring functions in [32]. In these experiments, ground-truth communities are perturbed using few perturbation techniques to degrade their quality. The scores given by various scoring functions for the ground-truth as well as the perturbed communities are then compared. A good scoring function is expected not only to give high scores for ground-truth communities and is also expected to give low scores for perturbed communities.

We consider a ground-truth community s and do the following perturbations on the community.

- NODESWAP - We randomly select an inter community edge (u, v) and swap the memberships of u and v i.e., we remove a node which is part of community s and add a node which is not a part of community s. This perturbation technique preserves the size of the community but affects the fringe of the community.

- RANDOM - We randomly select a node in community s and swap it's membership with another node in the network which belongs to a different community. This perturbation technique also preserves the size of community s. However, it affects the community more than the NODESWAP perturbation.
- EXPAND - We randomly select an edge (u, v) such that $u \in s$ and $v \notin s$. We add the node v to community s. This perturbation increases the size of community s.
- SHRINK - We randomly select an edge (u, v) such that $u \in s$ and $v \notin s$. We remove the node u from community s. This perturbation decreases the size of community s.

Let f be a scoring function and $f(s)$ be the community score of the community s under f. Let s be a ground-truth community and $h(s, p)$ be the perturbed community obtained by perturbing s using the perturbation technique h for a perturbation intensity of p. The perturbation intensity p is the fraction of nodes in the community that gets affected by our perturbation techniques. We measure $f(h(s, p))$ by taking the average over 20 trials. Z-score, which measures the difference of scores in units of standard deviation is given by,

$$Z(f, h, p) = \frac{E_s[f(s) - f(h(s, p))]}{\sqrt{Var_s[f(h(s, p))]}}$$

where $E_s[.]$,$Var_s[.]$ are respectively the mean and variance over communities s. Note that we negate the value of conductance because it gives low score to ground-truth communities and high score to perturbed communities.

We measure the Z-score of the four scoring functions for several perturbation intensities. At small perturbation intensities, Z-score is expected to be low indicating the robustness of the definition. At high perturbation intensities, Z-score is expected to be high indicating the reactivity of the definition. Table 3.3 shows the average of absolute difference of Z-score between small and large perturbation across the 3 networks we have considered : $Z(f, h, 0.4) - Z(f, h, 0.05)$.

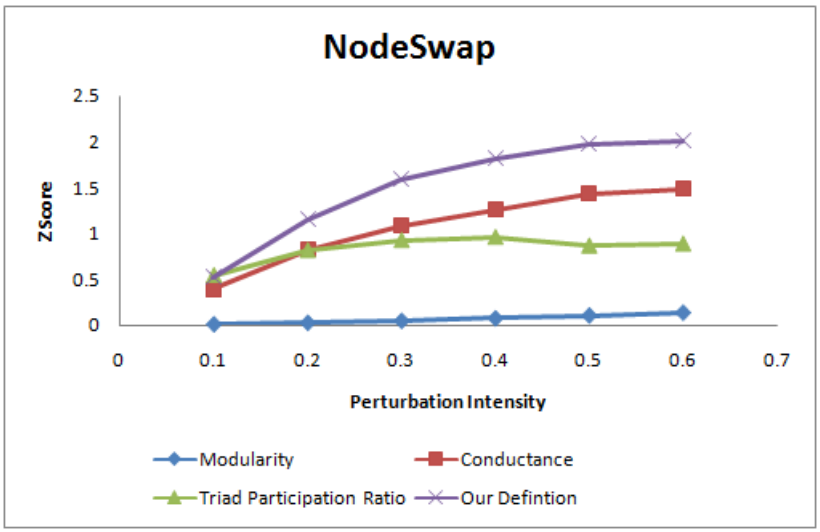

Figure 3.7: Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under NodeSwap perturbation.

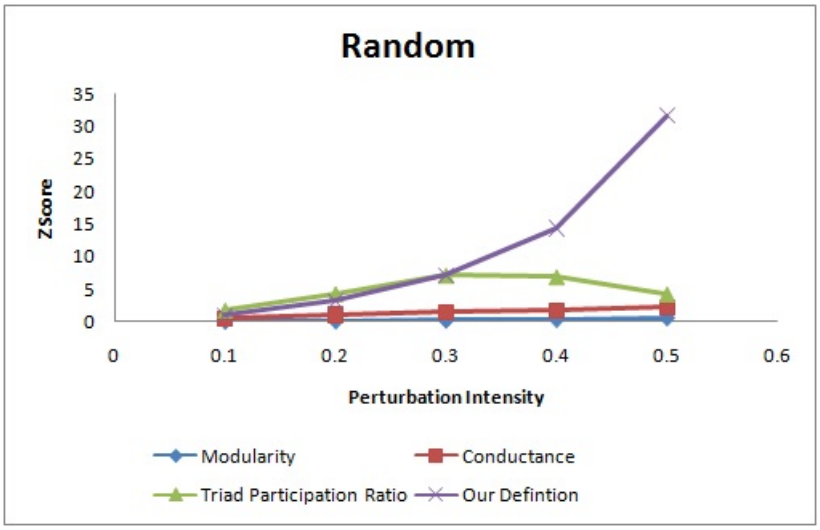

Figure 3.8: Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Random perturbation.

Figure 3.7, Figure 3.8, Figure 3.9 and Figure 3.10 shows the plot of Z-score as a function of perturbation intensity in the LiveJournal network. For the plots in other networks, refer appendix A. CEIL score performs significantly better in RANDOM and considerably better in NODESWAP and EXPAND. We obtained similar performance in

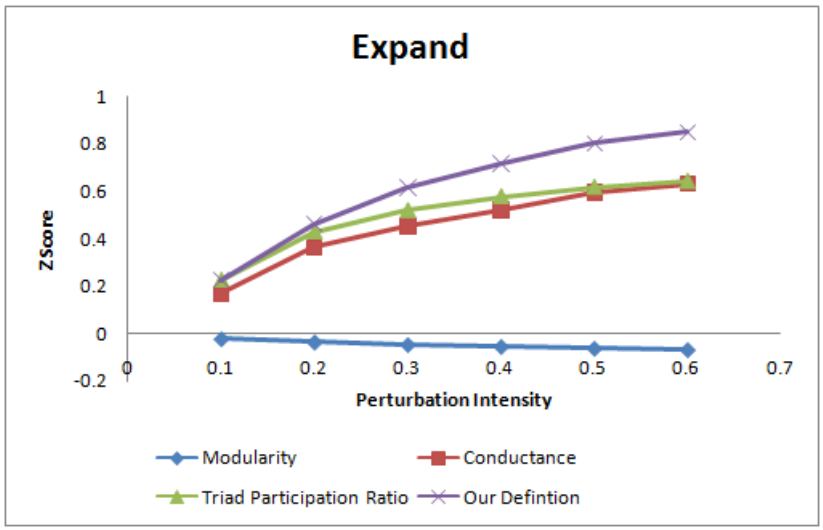

Figure 3.9: Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Expand perturbation.

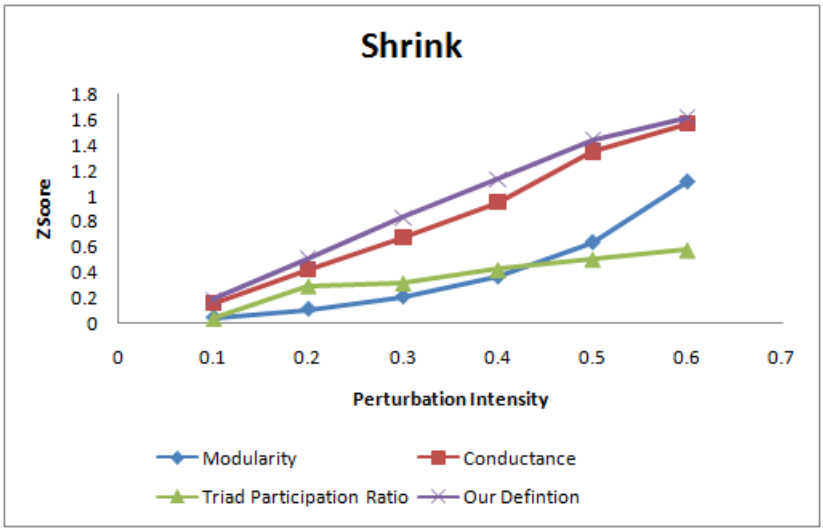

Figure 3.10: Z-score given by various scoring functions as a function of perturbation intensity in LiveJournal network under Shrink perturbation.

Youtube and DBLP datasets also. In SHRINK, we obtained mixed results as CEIL score does better in LiveJournal network while modularity does better in Youtube network and conductance does better in DBLP network.

Table 3.3 shows that CEIL score performs well under all the different perturbations

Table 3.3: Absolute difference in Z score between the large and small perturbation. Best scores are bolded.

Definitions	NodeSwap	Random	Expand	Shrink
Modularity	0.2123	0.2512	0.0350	**0.9994**
Conductance	1.1308	1.4024	0.3876	0.7101
TPR	1.1134	4.7466	0.2189	0.5555
CEIL Score	**2.7035**	**12.4463**	**0.8545**	0.8490

unlike other scoring functions. The reason for the poor performance of other scoring functions is that they do not include at least one of the three features necessary to characterize a community in their definition. We also note that modularity is not suitable for EXPAND and SHRINK perturbations since these perturbations affect the size of the community. The score given by modularity to a community depends on the size of the network in which the community is present. EXPAND perturbation affects it adversely as it increases the relative size of the community to size of the network and hence it performs poorly under EXPAND perturbation. On the other hand, SHRINK perturbation affects it favorably and hence it performs better under SHRINK perturbation.

Community Goodness Metrics

A community structure possesses the following two characteristics.

- Nodes in the community should be well connected to each other.
- A Community should be well separated from rest of the network.

Internal density and separability respectively are the goodness metrics which captures these two characteristics [9] [32]. These community goodness metrics are not community scoring functions. According to [32], “community scoring function quantifies how community-like a set is, while a goodness metric in an axiomatic way quantifies a desirable property of a community". Let s be a community, a_s be the number of intra

community edges in the community, b_s be the number of inter community edges incident on the community and n_s be the number of nodes forming the community. Then,

$$\text{Internal density}(s) = \frac{a_s}{n_s\left(n_s - 1\right)}$$

$$\text{Separability}(s) = \frac{a_s}{b_s}$$

We rank the ground-truth communities based on the density, separability as well as the score given by the scoring functions. We measure the correlation of the ranks given by the scoring functions and the ranks obtained through density and separability. We use spearman's rank correlation coefficient to find the correlation between the ranks [16]. It ranges from +1 to -1. A value of +1 indicate the highest positive correlation and a value of -1 indicate the highest negative correlation. Let the n raw scores X_i and Y_i be converted to ranks x_i and y_i respectively. Then,

$$\text{Spearman's rank correlation coefficient, } \rho = \frac{\sum_i \left(x_i - \overline{x}\right)\left(y_i - \overline{y}\right)}{\sqrt{\sum_i \left(x_i - \overline{x}\right)^2 \left(y_i - \overline{y}\right)^2}}$$

Table 3.4: Spearman's rank correlation coefficient for density

Networks	LiveJournal	Youtube	DBLP	Amazon
Modularity	-0.3751	-0.9017	-0.2313	-0.9070
Conductance	0.1963	0.5762	0.1736	-0.5676
TPR	0.4386	-0.5124	0.4052	0.4714
CEIL score	**0.5363**	**0.8279**	**0.7034**	**0.9474**

Table 3.5: Spearman's rank correlation coefficient for separability

Networks	LiveJournal	Youtube	DBLP	Amazon
Modularity	0.0600	-0.4854	0.0687	0.5791
Conductance	**1.0000**	**1.0000**	**1.0000**	**1.0000**
TPR	-0.0482	-0.4782	-0.0240	-0.3891
CEIL score	0.9002	0.9192	0.7954	-0.3513

From Table 3.4 and Table 3.5, we have the following conclusions. Modularity do not correlate well with both density and separability except in Amazon network where it has a reasonably good correlation in separability. Conductance absolutely correlates with separability but it doesn't correlate with density except in Youtube network where it is the second best. Triad participation ratio overall has the second best correlation with density but no correlation with separability. CEIL score has the highest correlation with density and has the second highest correlation with separability except in Amazon network. In the amazon network, communities with high internal density have low separability and vice versa. This is the reason for the negative correlation of CEIL score with separability in Amazon network. This correlation experiment also shows that the poor correlation of conductance with density is due to the fact that it does not consider the 'number of nodes forming the community'. Similarly, the poor correlation of triad participation ratio with separability is due to the fact that it do not consider the 'number of inter community edges incident on the community'. Since CEIL score takes into account all the features, it correlates well with both the goodness measures.

CHAPTER 4

CEIL Algorithm to Find Communities

In this chapter, we introduce the CEIL algorithm and compare it with other algorithms experimentally.

4.1 CEIL Algorithm

A greedy approach to find communities by efficiently maximizing an objective function is already proposed in [2]. Since it is one of the fastest known heuristic, we use the same method to maximize CEIL score. The algorithm has two phases. In the first phase, we assign each node to its own community. Then, we consider every node in the network in a sequential manner, remove it from its original community and add it either to the community of one of its neighbors or back to the original community, whichever will result in a greater increase in the CEIL score of the network. The newer properties of a community when a node n is added to the community is calculated as,

$$a_s = a_s + \text{intra}_n + \text{incident}_{n,s}$$
$$\text{deg}_s = \text{deg}_s + \text{deg}_n$$
$$n_s = n_s + n_n$$

where intra_n is the number of intra community edges in the community represented by node n, $\text{incident}_{n,s}$ is the sum of weights of the edges incident from node n to community s, deg_s is the sum of degree of all nodes in the community s, deg_n is the sum of

degree of all nodes in the community represented by node n and n_n is the number of nodes in the community represented by node n.

With the updated a_s, $\deg_s$ and n_s, the newer score and hence the increase is calculated. In a similar way, the decrease in score of a community when a node is removed from it is calculated. We repeat this process iteratively until there is no increase in the score given by the scoring function. At this time, the scoring function will reach its local maxima and the first phase ends.

Algorithm 1 Pseudocode of CEIL algorithm

Input: A graph G = (V,E)
Output: Label - A map from node to community
while True **do**
    for all v $\in$ V **do**
        Label[v] = v
    end for
    while Modified **do**
        Modified = False
        for all v $\in$ V **do**
            PreviousLabel = Label[v]
            Label[v] = Label of it's own or any of it's neighbors whichever gives a greater increase to the CEIL score.
            if PreviousLabel $\neq$ Label[v] **then**
                Modified = True
            end if
        end for
    end while
    if New CEIL Score $>$ Previous CEIL Score **then**
        $\hat{G}$ = An induced graph where nodes are communities of G.
        G = $\hat{G}$
    else
        Terminate the algorithm.
    end if
end while

In the second phase, we construct an induced graph of the network by using the community labels of nodes obtained from the first phase. Each community in the first

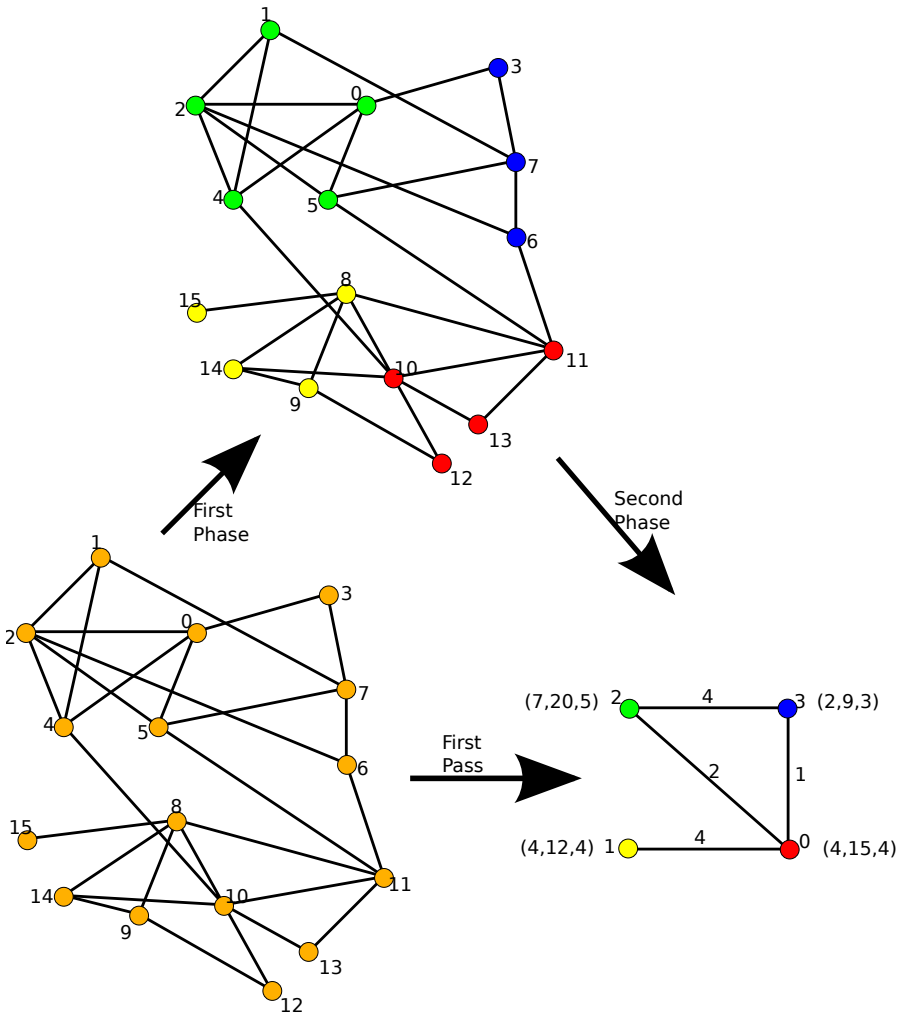

Figure 4.1: The network on the left side is an example network with 16 nodes. At the end of the first phase, which is the maximization of CEIL score, four communities are formed. In the diagram at the top, they are marked by four different colors. The second phase is to construct the induced graph using the labels of the first phase. It reduced the graph to 4 nodes. The numbers inside the parenthesis are the properties of the nodes - Number of intra community edges, Sum of degree of all nodes and Number of nodes of the community in the original graph which the node represents in the induced graph. One pass represents one iteration of the algorithm.

phase is represented by a node in the induced graph. Number of nodes in the community, sum of degree of all nodes in the community and the number of intra community edges

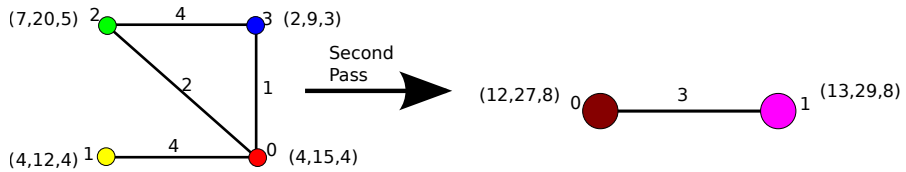

Figure 4.2: The second pass reduces the graph to two nodes. After this, merging of nodes decreases the score given by CEIL score to the network. So, the algorithm stops.

are all preserved in the induced graph by associating them with the respective node. Weight of an edge between two nodes in the induced graph is equal to the sum of weights of all edges between those two communities in the original graph. The second phase ends after the construction of the induced graph. We keep track of the scores of all the communities, i.e., update the scores of the communities as and when a change (addition or deletion of nodes) is made to the community. This will help in faster calculation of the increase or decrease to the community score whenever a change is made to the community.

The induced graph obtained as output of the second phase is given as the input to the first phase. The two phases are thus iterated until there is no increase in the score. At this time, it will reach a maximum value. Figure 4.1 and Figure 4.2 pictorially represents the working of CEIL algorithm.

In weighted networks, the number of edges is calculated as the sum of weights on all the edges. CEIL score of the network takes the low value of 0 but the high value is dependent on weights on the edges of the network. Nevertheless, the relative ordering of communities in a network will not get affected and so CEIL algorithm can be used in weighted networks also. By calculating only the edges going out from the nodes belonging to a community while calculating the number of intra and inter community edges, CEIL algorithm can be extended to directed graphs also. CEIL algorithm cannot be applied without modifications to find overlapping communities. But, CEIL score can

still be used to rank the overlapping communities.

4.2 Empirical validation of CEIL algorithm

One of our objectives is to develop a community detection algorithm that can scale to large networks. Hence, in this chapter, we restrict the comparison of CEIL algorithm to representative algorithms that exhibit good scaling behavior. Louvain method is a fast algorithm [2] which is used to find communities in large networks. Label propagation algorithm is simple and each iteration takes little time. But the number of iterations is prohibitively large. 95% of the nodes in the network happen to agree with the labels of atleast half of it's neighbors in 5 iterations in typical networks [29]. So, we have stopped the label propagation algorithm as soon as 95% of the nodes agree with the labels of atleast half of it's neighbors.

The experiments in this section are designed to show that CEIL algorithm finds,

- communities of different sizes.
- different number of communities.
- communities of varying size in the same network.
- communities in networks with different edge densities.
- communities in real world networks that matches closely with the ground-truth communities.

Evaluation Measures

We use Rand index [30] to compare the labels generated by the algorithms with the ground-truth labels. Rand Index is given by,

$$\text{Rand Index} = \frac{a+b}{a+b+c+d}$$

Rand Index considers all the $\binom{n}{2}$ possible pairs of n nodes of a network. a is the number of pairs where both the nodes belong to same community in ground-truth network as well as in the labels generated by the algorithm, b is the number of pairs where both the nodes belong to different community in ground-truth label as well as in the labels generated by the algorithm, c is the number of pairs where both the nodes belong to same community in ground-truth label but belongs to different community in the labels generated by the algorithm and d is the number of pairs where both the nodes belong to different community in ground-truth label but belongs to same community in the labels generated by the algorithm.

We define E_{res} to be the errors due to the classification of a pair of nodes into the same community which are actually in different communities in the ground-truth.

$$E_{res} = \frac{d}{a+b+c+d}$$

Apart from rand index, there are several other measures with which we can measure the overlap between ground-truth communities and the communities given by algorithms. We have chosen RAND index as the comparison metric since it penalizes incorrect assignments and rewards correct assignments. Further, Rand index is more forgiving when you report more communities than in the ground truth. Few of the notable measures are as below.

Normalized Mutual Information [6] measures the mutual dependence of two random variables. It is given by,

$$\text{NMI}(X, Y) = \frac{I(X, Y)}{\sqrt{H(X)H(Y)}}$$

The normalization variables $H(X)$ and $H(Y)$ are entropies of random variables X and Y respectively.

F-measure or F-score is another evaluation measure which is harmonic mean of precision and recall. It is given by,

$$\text{F-score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Experimental demonstration of resolution limit

In [8], synthetic networks with specific structural properties were used to prove the resolution limit in modularity. We use similar networks to show that CEIL algorithm finds the expected communities as opposed to modularity maximization.

Circle of cliques :

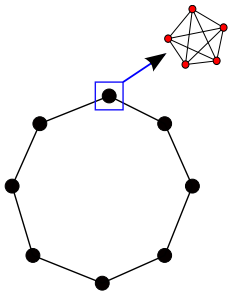

Figure 4.3: Circle of Cliques

Several equal sized cliques are arranged in a circle. Each clique is then connected to the neighbors on either side by an edge. The intuitive number of communities in

this network is the number of cliques with each clique being a community. Figure 4.3 shows an example network of circle of cliques. Each dot correspond to a clique. The line between any two dots is a single edge connecting the corresponding cliques.

- Consider such a network of 30 cliques with each clique having 5 nodes. CEIL algorithm gave 30 communities with each clique being a separate community. Louvain method was shown to give only 15 communities with two adjacent cliques belonging to a single community [8].
- To show that CEIL algorithm finds communities in networks irrespective of the number of communities, we repeated this experiment with 300, 3000 and 30000 cliques of size 5. In all the cases, CEIL algorithm was able to find each of the cliques as a separate community.
- To show that CEIL algorithm finds communities irrespective of the size, we kept the number of communities in this network to a fixed value of 10 and generated this network with size of cliques as 50, 500 and 1000. In all these networks, CEIL algorithm was able to find the correct communities.

Two Pair of cliques :

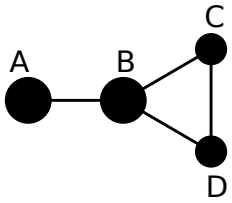

Figure 4.4: Two Pair of Cliques

To show that CEIL algorithm finds communities in networks where the communities differ in size, we generated the two pair of cliques network. Two pair of cliques network consists of a pair of big sized cliques and a pair of small sized cliques. In Figure 4.4, A and B are cliques of size 20, i.e., 20 nodes and 190 edges while C and D are cliques of size 5. The line between any two dots is the single edge connecting two cliques. CEIL algorithm gave 4 communities with each clique being a community. Louvain method was shown to give only 3 communities [8]. They are A, B and a third community

encompassing C and D. We kept the size of the two small cliques as constant at a size of 5 and generated three networks with size of the big cliques as 200, 2000 and 5000. In each of these cases, CEIL algorithm is able to find the four cliques as four different communities.

Note that we can construct numerous such examples where modularity maximization fails due to the resolution limit while maximization of CEIL score does not fail. The above experiments also show that CEIL algorithm will be able to identify communities irrespective of their size and number provided that a strong community structure is present in the network.

Four Community Network

To show that CEIL algorithm performs well on graphs with different densities, we generated a synthetic network consisting of four communities [6] where the density of the networks can be varied. The nodes belonging to a community are linked with probability P_{in} to nodes belonging to the same community and are linked with probability P_{out} to the nodes belonging to other communities. The probabilities P_{in} and P_{out} are chosen such that the average degree of nodes in the network is fixed to a certain value and thereby the density of edges in the network is fixed. By varying the probabilities P_{in} and P_{out}, the average number of links a node has to the nodes belonging to the same community, Z_{in}, and the average number of links a node has to the nodes belonging to the other communities, Z_{out}, can be controlled. As the value of Z_{out} increases, it is difficult for the algorithms to identify the communities.

We generated a network consisting of 128 nodes having 4 communities with 32 nodes for each communities as described in the benchmark proposed by Danon et al. [6]. We varied the density of edges in this network by controlling the sum of Z_{in} and Z_{out}.

The density of edges in the resulted networks are 25.19%, 12.59%, and 6.29%. In the network with density 25.19%, we were able to recover the 4 ground-truth communities for all $Z_{in} > Z_{out}$. In the network with density 12.59%, we were able to identify the communities fully when Z_{out} was 0 and 1. We obtained a rand index of 0.9922 and 0.93 when Z_{out} was 2 and 3. In the network with density 6.29%, we were able to obtain a rand index of 0.9506 when Z_{out} was 0. We note that the ability to recover the communities goes down as the density of edges in the network decreases. This is because we put a tighter constraint on what a community is. When Z_{out} was zero, the external score will be 1 but the internal score will be very less due to the lesser density of edges. This leads to some nodes being excluded from the community. We note that Louvain method was able to recover the communities even in the network with density of 6.29% when Z_{out} was zero. But, it is debatable that at this density whether any community structure exists or not.

Real World Graphs

To show that CEIL algorithm matches the ground-truth communities in real networks, we ran CEIL algorithm, Louvain method and Label Propagation algorithm in real world graphs and generated community labels for all the nodes in the networks. Few nodes in the network have no ground-truth label while few others have multiple ground-truth labels. We have only considered the nodes which have exactly one ground-truth community label for calculating the rand index. Essentially, the nodes which we consider are the core of the ground-truth communities. This process has resulted in many small communities. So, we have removed the communities which have less than 3 nodes. Table 4.1 shows the rand index.

From Table 4.1, we see that CEIL algorithm captures the ground-truth communities better than Louvain method. Though the label propagation algorithm gives a better

Table 4.1: Rand index

Networks	Louvain method	Label Propagation(95%)	CEIL algorithm
Youtube	0.8957	0.6915	0.9959
DBLP	0.9702	0.9818	0.9828
Amazon	0.9910	0.9953	0.9938

Table 4.2: Running Time

Networks	Louvain method	Label Propagation(95%)	CEIL algorithm
Youtube	321.25s	30.03s	395.25s
DBLP	134.39s	571.69s	77.77s
Amazon	81.17s	10.76s	80.68s

match to ground-truth communities in Amazon network, it's performs poorly in Youtube network. Since we have stopped this algorithm earlier, there is no guarantee for the performance of this algorithm on all the networks.

The errors E_{res} for Louvain method are 0.1026, 0.0145 and 0.0087 on Youtube, DBLP and Amazon networks while they are only 0.000015, 0.000002 and 0.00 for CEIL algorithm. Since we are not considering the nodes having multiple labels for the calculation of rand index, this type of mis-classification occurs only due to the resolution limit of the modularity.

We also note from Table 4.2 that the running time of CEIL algorithm is on par with the Louvain method. Only difference between Louvain method and CEIL algorithm is the scoring function which is maximized. The computation of both the scoring functions using the pre-computed parameters requires O(1) time. So, the empirical running time of both the algorithms is the same. But both the algorithms generate different number and size of communities depending on the topology of the networks. This is the reason for small differences in the running time of the algorithms. Label propagation has a much better running time in two of the networks. This is due to the reason that we have stopped the algorithm as soon as the 95% of the nodes in the network agree with the

labels of atleast half of their neighbors. It takes several hours in many networks for 100% of the nodes to agree with the labels of atleast half of their neighbors. Since we are considering only the core members of each of the community, all the algorithms give a higher value of rand index.

CHAPTER 5

Parallelization of Centrality Algorithms

Game theoretic centrality algorithms model the importance of nodes when combined with other nodes in the network [18], [19]. This makes it suitable to be applied in the context of information diffusion. Polynomial time algorithms to compute game theoretic centrality were introduced by Aadithya et al. [14]. In this chapter, we discuss the parallelization techniques which we had used to parallelize these game-theoretic centrality algorithms.

5.1 Map-Reduce and Hadoop

Map-Reduce : Map-reduce [7] is a parallel programming model for data intensive applications. Every map-reduce task starts with a map phase. The input and output to this phase are key-value pairs. One key-value pair is read at a time and one or many key-value pairs are written in the output.

$$(Key1, Value1) \rightarrow (list(Key2, Value2))$$

An optional combiner can be used at the end of the map phase to reduce the size of the output of the map phase. This reduces the amount of data to be transferred over the network which otherwise would be a bottleneck. This phase is followed by the sort and shuffle phase. The output of the map (or combiner) is sorted and then are sent to the respective reducers based on a partition algorithm. The default partition algorithm just

shuffles the map output to reducers but it can can be overridden by a custom partitioning algorithm. Once the sort and shuffle phase ends, the reduce phase begins. The input to this phase is the key and the list of values corresponding to that key. For each key, one or many key-value pairs are written into the output by the reducer.

$$(Key2, list(Value2)) \rightarrow (list(Key3, Value3))$$

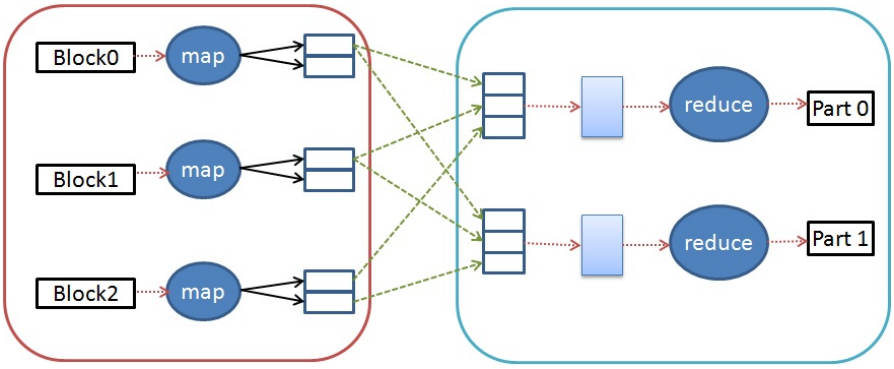

Figure 5.1: Map-Reduce model with 3 mappers and 2 reducers

Hadoop : Hadoop[10] is an open source implementation of the map-reduce programming model. It also encompasses the Hadoop distributed file system(HDFS) [3]. Hadoop follows a master slave architecture where the master(Job Tracker) takes up a job, assigns part of the tasks to each of the slaves(Task Tracker) and tracks the progress of the job from the reports of the slaves to the master. Generally, slaves are the data nodes wherein the input data is stored and processed. But, master can also act as a data node and do computation. The input file is broken into several chunks of equal size(except the last chunk) and are distributed among the data nodes. Each block or a chunk is replicated and are stored at different machines providing fault tolerance. Every machine can run any number of map or reduce tasks at a time. This framework is used to parallelize the five centrality algorithms proposed in [14] to run them on big data.

Challenges in parallelization : In general, large graphs are represented in the edge-

list format. In this format, each line of the input file is an edge. For example, edge AB is represented by "A [space] B" where A and B are the nodes forming the edge.

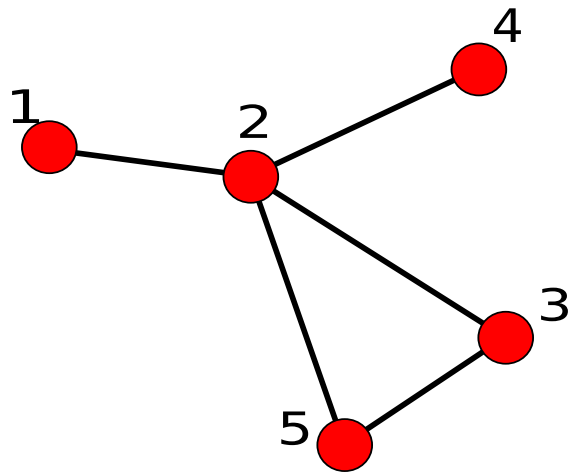

Figure 5.2: An example network which is unweighted

Input format for the graph represented in Figure 5.2 could be:

1 2

2 3

2 4

2 5

3 5

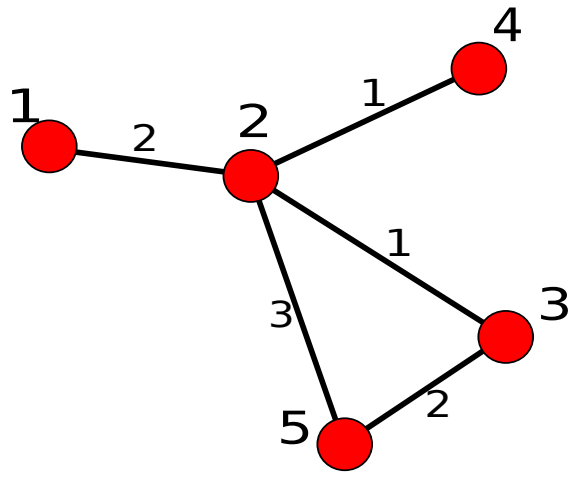

Figure 5.3: An example network which is weighted

Input format for the graph represented in Figure 5.3 could be:

1 2

1 2

2 3

2 4

2 5

2 5

5 2

3 5

5 3

Challenges: The challenges from programming perspective are,

1. The input file is partitioned into equal chunks and the chunks are distributed across the machines in the cluster. So, the information related to a particular node may be stored partially in many machines. For example, edges "1 2" and "1 3"

may be present in two different machines. In this case, we say that the information about node 1 is partially available in each of the two machines.

2. The mapper reads the input chunk (or an input file) in a sequential manner such that it reads only one line at a time. In our case, since each line corresponds to an edge, we say that a mapper processes one edge at a time. Essentially, a mapper will have knowledge only about the edge which is getting processing at that time and will not have any information about the other edges in the network. Similarly, a reducer processes one key at a time and will have information about only that particular key and its values at a time.

3. The number of map-reduce phases have to be less. As the number of map-reduce phases reduce, amount of computation decreases and hence there will be a decrease in the running time of the algorithm.

4. The amount of intermediate data, i.e., the output of map has to be less. This intermediate data have to be shuffled and sorted and have to be sent to the reducer through the network. If the intermediate data is large, it will become a bottleneck and the algorithm slows down.

5.2 Parallelization of game theoretic centrality algorithms

In this section, we describe the five game theoretic centrality algorithms proposed in [14] and the techniques to parallelize them. The class of centrality measures proposed in [14] are defined using cooperative games on networks. The following definitions describe the general methodology used for characterizing these measures.

Definition: A *cooperative game* is characterized by a set of players N and a payoff function $\nu : 2^N \rightarrow \mathcal{R}$, that assigns a real value to any subset of players, or *coalition*.

Definition: The set of nodes influenced by a coalition is known as the *fringe* of the coalition. By changing the definition of a fringe we can mode different influence propagation mechanisms.

Definition: The *Shapley value* of a node in a cooperative game is the expected marginal contribution of the node to any coalition. This is determined by computing the expected increase in the value of a randomly chosen coalition when this node is added to it.

The definitions of fringe in the different network games used in [14] capture different models of information/influence propagation in a network. The Shapely value of a node can then be used as a centrality measure in the context of information diffusion. For more details of the approach, please refer to [14].

An Example: Degree Centrality

Degree centrality is the easiest of the centralities in terms of computation. The degree centrality of a node is defined as the degree of the node. Let the input file be in the edgelist format. The degree centrality can be calculated in one map-reduce phase as in Algorithm 2.

The input file gets partitioned into several chunks and every machine in the cluster will have few chunks of this file. In the map phase, mapper is started in every machine and each mapper processes one chunk of the input file. Each chunk comprises a portion of the edgelist and all the chunks together forms the edgelist of the network. The mapper processes one line at a time i.e., the map function gets called for every input line. In this case, since each line is an edge of the network, we can say that it processes one edge at a time. The map function tokenizes the input line into tokens. The tokens are the two nodes(say A and B) which had formed the edge. For every edge, two (key,value) pairs

are written to the output. They are,

$$A \quad 1$$

$$B \quad 1$$

Algorithm 2 Degree Centrality

Input: Edgelist of $G(V, E)$
Output: Degree Centrality

```
class MAPPER
    method MAP(lineNumber, edge)
        (nodeA,nodeB) = edge.split()
        EMIT(nodeA, 1)
        EMIT(nodeB, 1)

class REDUCER
    method REDUCE(node, list(values))
        degree = 0
        for all value ∈ values do
            degree += value
        end for
        EMIT(node, degree)
```

Once every mapper in all the machines has completed processing their chunk, the output of all the mappers go to a sorting and shuffle phase. In this phase, all the map outputs are sorted and a partitioner sends them to the reducer. The partitioner uses a hash function and makes sure that all the (key,value) pairs which have the same key goes to the same reducer. In reducer, the values corresponding to a single key are put together in a list. The reduce function gets called for every (key, value) pair. It aggregates the values in the list corresponding to each key. Each key here is a node in the network and the aggregated sum is the degree of that node. So, the output of the reducer is just the pair (node, degree). The output file from all the machines are concatenated which gives

the degree centrality of all nodes in the network.

Game 1: Number of agents at-most 1 degree away

The first game theoretic centrality algorithm computes centrality by considering the fringe as the set of all nodes that are reached in at-most one hop. Algorithm 3 gives the steps to calculate this centrality. It has a running time of $O(V + E)$.

Algorithm 3 Game 1

Input: Unweighted graph $G(V, E)$
Output: Centrality values of all nodes

for all $v \in V$ **do**
    Centrality(v) = $\frac{1}{1+\text{degree}(v)}$
    for all $u \in$ Neighbors(v) **do**
        Centrality(v) += $\frac{1}{1+\text{degree}(u)}$
    end for
end for

From Algorithm 3, we observe that in order to compute the centrality of a node, we need to find degree of the node and its neighbors. The intuition behind this algorithm is that the centrality of a node will be higher not just when the node has a high degree but also when the node has more neighbors who have low degree. We parallelize this algorithm using two map-reduce phases.

In the first map-reduce phase, we calculate degree of all the nodes and their neighbors in a parallel fashion as described in Algorithm 4 . The map-phase of this stage do not do any computation but modifies the input edge in such a way that degree of nodes can be calculated from the edgelist. For every edge in the input file, two lines are written in the output. Let "A B" be the line which is getting processed by the mapper. The first line of the output will have A as the key and B as the value. This indicates that B is

a neighbor of A. Since the graph is undirected, A is a neighbor of B. So, the second line of the output will have B as the key and A as the value. The combiner is not needed in this phase since we are not interesting in merging the values corresponding to the same key. We use the default partitioner which makes sure that all the (key, value) pairs with the same key goes to the same reducer.

Algorithm 4 First Map-Reduce Phase of Game 1

```
class MAPPER
    method MAP(lineNumber, edge)
        (nodeA, nodeB) = edge.split()
        EMIT(nodeA, nodeB)
        EMIT(nodeB, nodeA)

class REDUCER
    method REDUCE(node, list(neighbors))
        degree = length(neighbors)
        marginalContribution = 1/(1+degree)
        EMIT(node, marginalContribution)
        for all neighbor ∈ neighbors do
            EMIT(neighbor, marginalContribution)
        end for
```

The reducer in this stage receives nodes as keys and their neighbors as values. We calculate the degree of nodes by counting the number of their neighbors. Note that, if a weighted graph is given as input, duplicate representations of the same neighbor will be found. We find the duplicates using the hash value of the neighbors and ignore the duplicates while counting the number of neighbors.

Since, we are interested in the calculation of the marginal contributions of each node, which is $\frac{1}{1+degree}$ in this case, we write the marginal contributions of the nodes in the output instead of their degrees. Every node needs it's marginal contribution in order to calculate it's own centrality. So, the reducer writes the node as the key and it's marginal contribution as value. Also, all the neighbors need the marginal contribution of the node

in order to calculate their centrality. So, the reducer writes every neighbor as key and marginal contribution of the node as value for all the neighbors.

Algorithm 5 Second Map-Reduce Phase of Game 1

```
class MAPPER
    method MAP(lineNumber, value)
        (node, marginalContribution) = value.split()
        EMIT(node, marginalContribution)

class REDUCER
    method REDUCE(node, marginalContributions)
        centrality = 0
        for all marginalContribution ∈ marginalContributions do
            centrality += marginalContribution
        end for
        EMIT(node, centrality)
```

In the second map-reduce phase, centrality values of all the nodes are calculated. The mapper in this phase does not do any computation. It reads the input from the file written by the reducer of first phase and tokenizes them into (key,value) pairs and writes them to the output. The reducer in this phase gets nodes as keys and the marginal contributions of all their neighbors as values. So, the reducer aggregates the values corresponding to the nodes which will give the centrality of the nodes. A combiner is used in this phase which does the same computation as the reducer.

Game 2: Number of agents with at least k neighbors in C

The second game theoretic centrality algorithm computes centrality by considering the fringe as the set of all nodes that are either in the coalition or that are adjacent to at least k nodes which are already in the coalition. Algorithm 6 gives the steps to calculate this centrality. It has a running time of $O(V + E)$.

Algorithm 6 Game 2

Input: Unweighted graph $G(V, E)$
Output: Centrality values of all nodes

for all $v \in V$ **do**
    k(v) = Random(1,degree(v)+1)
end for
for all $v \in V$ **do**
    Centrality(v) = min $\left(1, \frac{k(v)}{1+\text{degree}(v)}\right)$
    for all $u \in$ Neighbors(v) **do**
        Centrality(v) += max $\left(0, \frac{\text{degree}(u)-k(v)+1}{\text{degree}(u)(1+\text{degree}(u))}\right)$
    end for
end for

From Algorithm 6, we observe that in order to compute the centrality of a node, we need to find the degree of the node and its neighbors. The intuition behind this algorithm is that every node will be influenced only when k of it's neighbors are already influenced. This k is the threshold which varies from 1 to degree(node)+1. A threshold of degree(node)+1 for a node indicates that the node cannot be influenced even when all its neighbors are influenced. We parallelize this algorithm using two map-reduce phases.

The first map-reduce phase of this algorithm is given in Algorithm 7 . The only difference between the first phase of game 1 and game 2 algorithms is the way in which the marginal contributions are calculated. So, the mapper does the same job as Algorithm 4 . In the reduce phase, marginal contributions for a node is calculated by the formula $\frac{k}{1+degree}$ and the marginal contribution of its neighbors are calculated by the formula $\frac{degree-k+1}{degree(1+degree)}$.

Algorithm 7 First Map-Reduce Phase of Game 2

```
class MAPPER
    method MAP(lineNumber, edge)
        (nodeA, nodeB) = edge.split()
        EMIT(nodeA, nodeB)
        EMIT(nodeB, nodeA)

class REDUCER
    method REDUCE(node, list(neighbors))
        degree = length(neighbors)
        marginalContribution = k/(1+degree)
        EMIT(node, marginalContribution)
        marginalContribution = (degree-k+1)/(degree(1+degree))
        for all neighbor ∈ neighbors do
            EMIT(neighbor, marginalContribution)
        end for
```

The second map-reduce phase of this algorithm is essentially the same as that of the second phase of game 1 $(Algorithm\ 5)$ which aggregates the marginal contributions to obtain the centrality.

Game 3: Number of agents at-most d_{cutoff} away

The third game theoretic centrality algorithm computes centrality by considering the fringe as the set of all nodes that are within the distance of d_{cutoff} from the node. Algorithm 8 gives the steps to calculate this centrality. It has a running time of $O(VE + V^2 log(v))$.

Algorithm 8 Game 3

Input: Weighted graph $G(V, E)$
Output: Centrality values of all nodes

for all $v \in V$ **do**
  extNeighbors(v) = {}
  extDegree(v) = 0
  for all $u \in V$ such that u≠v **do**
    if Distance(u) $< d_{\text{cutoff}}$ **then**
      extNeighbors(v).push(u)
      extDegree(v)++
    end if
  end for
end for
for all $v \in V$ **do**
  Centrality(v) = $\frac{1}{1+extDegeee(v)}$
  for all $u \in$ extNeighbors(v) **do**
    Centrality(v) += $\frac{1}{1+extDegree(u)}$
  end for
end for

From Algorithm 8, we observe that in order to compute the centrality of a node, we need to find the extDegree and extNeighbors of every node. This algorithm is an extension of game 1 to the weighted networks. The intuition behind this algorithm is that a node can be influenced by an influencer only when the distance between the node and the influencer is not more than d_{cutoff}. This d_{cutoff} is generally fixed to a constant value and we have fixed it as 2 in our parallelization. For an unweighted graph, all the nodes which are one and two hops away will form the extNeighbors but for a weighted graph, it depends on the weights of the edges of the graph. We parallelize this algorithm using four map-reduce phases.

The first map-reduce phase of this algorithm is given in Algorithm 9 . The map phase of this algorithm does the same job as the map phase of the first map-reduce phase of game 1 and game 2 algorithms. The reducer in this phase gets nodes as keys and list

of their neighbors as values. These neighbors are one hop neighbors i.e., they are connected to the node by a single edge. The number of times a neighbor appears in the list is the weight of the edge between the node and neighbor. So, the number of occurrences of each of the neighbors is calculated and is stored in map named neighbor2weight. The key to this map is the neighbor and the value to this map will be the weight of the edge between the node and the neighbor. We create another map named weight2neighbors using this map where key is weight of the edge between node and neighbor and value is the list of neighbors who are connected to the node with that weight.

Two hop neighbors are the neighbors who are reachable in at most two hops. Let A and B be two nodes which are two hops away i.e., A and B are not connected directly but connected through another node. Let the node through which A and B are connected be C. Now, A and B are one hop neighbors of C and similarly all the pair of nodes which are two hops away will have a common neighbor from which both of them will be one hop away. So, each node in the list of neighbors(input to the reducer of this stage) are two hops away from every other node in the same list of neighbors. Also, some of the nodes which are two hops away might as well be connected by a single edge. In this case, the shortest distance between them is 1 and not 2. So, whenever a reducer encounters a neighbor having two different values as weight for a same edge, it always has to choose the least value. Also, the edges may have weights. So, the reducer also has to check whether the sum of weights of edges is less than the d_{cutoff}. The reducer is this phase essentially finds the neighbors which are one hop more than what it has received as input. In the current map-reduce phase, reducer has received the neighbors which are one hops away and has found the neighbors which are two hops away. This can be extended further for higher hops in the same way. [1]

[1]Map(a→b) used in the algorithms of this thesis represents the mapping from a to b. Here, a is the key and b is the value. The key is always unique and it can have multiple values.

Algorithm 9 First Map-Reduce Phase of Game 3

```
class MAPPER
    method MAP(lineNumber, Edge)
        (nodeA, nodeB) = Edge.split()
        EMIT(nodeA, nodeB)
        EMIT(nodeB, nodeA)

class REDUCER
    method REDUCE(node, list(neighbors))
        for all neighbor ∈ neighbors do
            currentWeight = Map(neighbor→weight).get(neighbor)
            Map(neighbor→weight).delete(neighbor,currentWeight)
            Map(neighbor→weight).add(neighbor,currentWeight+1)
        end for
        for all neighbor ∈ neighbors do
            EMIT(node, neighbor)
            weight = Map(neighbor→weight).get(neighbor)
            Map(weight→neighbors).add(weight, neighbor)
        end for
        weights = keys(Map(weight→neighbors))
        for all weight ∈ weights do
            neighborString = null
            for all neighbor ∈ neighbors do
                neighborWeight = Map(neighbor→weight).get(neighbor)
                newWeight = weight + neighborWeight
                if newWeight < d_cutoff then
                    neighborString.concat(neighbor+":"+newWeight+":")
                end if
            end for
            Map(weight→neighborString).add(weight, neighborString)
        end for
        for all neighbor ∈ neighbors do
            neighborWeight = Map(neighbor→weight).get(neighbor)
            neighborString = Map(weight→neighborString).get(neighborWeight)
            EMIT(neighbor, neighborString)
        end for
```

The reducer constructs a neighborString which contains neighbors that one more hop away and whose path length is less than or equal to the d_{cutoff}. So, for every neighbor

in the list of neighbors, the reducer writes the neighbor as key and a neighborString as value depending on the weight of the neighbor to the node(input to the reducer of this phase).

Algorithm 10 Second Map Phase of Game 3

```
class MAPPER
    method MAP(lineNumber, Edge)
        index = Edge.find(":")
        node = Edge.substring(0,index)
        listOfNeighbors = Edge.substring(index+1,end)
        EMIT(node, listOfNeighbors)
```

The second map-reduce phase of this algorithm makes the neighborhood grow by one more hop. The mapper in this phase is given in Algorithm 10 . It reads the output of the previous phase and breaks the line into (key,value) pairs and sends it to the reducer. The reducer in this phase is essentially the same as that of the previous map-reduce phase. This map-reduce phase is run iteratively until all the neighbors which are within d_{cutoff} away are visited.

Algorithm 11 Third Map-Reduce Phase of Game 3

```
class MAPPER
    method MAP(lineNumber, Edge)
        index = Edge.find(":")
        node = Edge.substring(0,index)
        listOfNeighbors = Edge.substring(index+1,end)
        EMIT(node, listOfNeighbors)

class REDUCER
    method REDUCE(node, list(extNeighbors))
        extNeighbors = set(list(extNeighbors))
        extDegree = length(extNeighbors)
        marginalContribution = 1/(1+extDegree)
        EMIT(node, marginalContribution)
        for all extNeighbor ∈ extNeighbors do
            EMIT(extNeighbor, marginalContribution)
        end for
```

The third map-reduce phase of this algorithm is given in Algorithm 11 . The mapper in this phase reads the input from the file and splits them into (key, value) pairs. The reducer in this stage receives nodes as keys and their extNeighbors as values. We calculate the extDegree of nodes by removing the duplicates and then counting the number of extNeighbors. Once the extDegree is calculated, marginal contribution is calculated by the formula $\frac{1}{1+extDegree}$. The reducer writes the marginal contributions to the output similar to game 1.

The fourth and the final map-reduce phase of this algorithm is essentially the same as that of the second phase of game 1 $(Algorithm\ 5)$ which aggregates the marginal contributions to obtain the centrality.

Game 4: Number of agents in the network

The fourth game theoretic centrality algorithm computes centrality by considering the fringe as the set of all nodes in the network. Algorithm 12 gives the steps to calculate this centrality. It has a running time of $O(VE + V^2 log(v))$.

From Algorithm 12, we observe that in order to compute the centrality of a node, we need to find the extNeighbors and the distance to the extNeighbors for every node. This algorithm is an extension of game 3. The intuition behind this algorithm is that the power of influence decreases as the distance increases. Generally, the extNeighbors are all the nodes in the network for this game. But the contribution of neighbors who are farther away is not highly significant and so in our parallelization, we have fixed the d_{cutoff} to be 2. We parallelize this algorithm using four map-reduce phases.

Algorithm 12 Game 4

Input: Weighted graph $G(V, E)$
Output: Centrality values of all nodes

for all $v \in V$ **do**
  (Distances, Nodes) $\leftarrow$ Dijkstra(v, G)
  sum = 0
  prevDistance = -1
  prevContribution = -1
  for all index $\in$ |V-1| to 1 **do**
    if Distances(index) = prevDistance **then**
      contribution = prevContribution
    else
      contribution = $\frac{f(D(index))}{1+index}$ - sum
    end if
    centrality[w(index)] += contribution
    sum += $\frac{f(D(index))}{index(1+index)}$
    prevDistance = Distances(index)
    prevContribution = contribution
  end for
  centrality(v) += f(0) - sum
end for

The first and second phase of this algorithm is essentially the same as game 3 ($Algorithm$ 9 and $Algorithm$ 10). In these phases, the neighbors of each of the nodes are found. Each map-reduce phase extends the neighborhood by one hop. This process is repeated iteratively until there is no more unvisited neighbors within the d_{cutoff}.

The third map-reduce phase of this algorithm is given in Algorithm 13 . The mapper in this phase reads the input from the file and splits them into (key, value) pairs such that every node in the graph is the key and the list of neighbors and weights as values. The weight here represents the distance from the node to the neighbor. Once the neighbors and weights are known, the marginal contributions are calculated according to the lines 3 to 17 of Algorithm 12 . The reducer writes the nodes and the marginal contributions to the output.

Algorithm 13 Third Map-Reduce Phase of Game 4

```
class MAPPER
    method MAP(lineNumber, Edge)
        index = Edge.find(":")
        node = Edge.substring(0,index)
        listOfNeighbors = Edge.substring(index+1,end)
        EMIT(node, listOfNeighbors)

class REDUCER
    method REDUCE(node, list(neighbors,distances))
        index = length(neighbors)
        sum = 0
        prevDistance = -1
        prevContribution = -1
        for all neighbor ∈ neighbors do
            if distances(index) == prevDistance then
                contribution = prevContribution
            else
                contribution = f(D(index))/(1+index) - sum
            end if
            EMIT(neighbor, contribution)
            sum += f(D(index))/(index(1+index))
            prevDistance = Distances(index)
            prevContribution = contribution
            index = index - 1
        end for
        contribution = f(0) - sum
        EMIT(node, contribution)
```

The fourth and the final map-reduce phase of this algorithm is essentially the same as that of the second phase of game 1 $(Algorithm\ 5)$ which aggregates the marginal contributions to obtain the centrality.

Game 5: Number of agents with $\sum$(weights inside C) $\geq W_{\text{cutoff}}$(agent)

The fifth game theoretic centrality algorithm computes centrality by considering the fringe as the set of all nodes whose agent specific threshold is less than the sum of influences on the node by the nodes who are already in the coalition. Algorithm 14 gives the steps to calculate this centrality. It has a running time of $O(V + E^2)$.

Algorithm 14 Game 5

Input: Weighted graph $G(V, E)$
Output: Centrality values of all nodes

---

for all $v \in V$ **do**
  $W_{\text{cutoff}}(v)$ = Random(1,degree(v)+1)
end for
for all $v \in V$ **do**
  centrality(v) = 0
  for all m in 0 to $deg(v)$ **do**
    $\mu = \mu(X_m^{vv})$
    $\sigma = \sigma(X_m^{vv})$
    $p = Pr\{\mathcal{N}(\mu, \sigma^2) < W_{\text{cutoff}}(v)\}$
    centrality(v) += $\frac{p}{1+deg(v)}$
  end for
  for all $u \in$ Neighbors(v) **do**
    p = 0
    for all m in 0 to $deg(v) - 1$ **do**
      $\mu = \mu(X_m^{uv})$
      $\sigma = \sigma(X_m^{uv})$
      $Z = Z_m^{uv}$
      p += z $\frac{deg(u)-m}{deg(u)(deg(u)+1)}$
    end for
    centrality(v) += p
  end for
end for

From Algorithm 14, we observe that in order to compute the centrality of a node, we need to find the degree of the node and it's neighbors. This algorithm is an extension

of Game 2 (Algorithm 6) for weighted networks. The intuition behind this algorithm is that every node will be influenced only when the sum of weights to all the active neighbors is greater than the cutoff of the node. Let α_v be the sum of weights of edges to all the neighbors of node v and β_v be the sum of squares of weights of edges to all the neighbors of v. Then, the results of the analysis done in [14] for this algorithm are as follows:

$$\mu(X_m^{vv}) = \frac{m}{deg(v)}\alpha_v$$
$$\sigma(X_m^{vv}) = \frac{m(deg(v)-m)}{deg(v)(deg(v)-1)}\left(\beta_v - \frac{\alpha_v^2}{deg(v)}\right)$$
$$\mu(X_m^{uv}) = \frac{m}{deg(v)-1}(\alpha_v - w(u,v))$$
$$\sigma(X_m^{uv}) = \frac{m(deg(v)-1-m)}{(deg(v)-1)(deg(v)-2)}\left(\beta_v - w(u,v)^2 - \frac{(\alpha_v - w(u,v))^2}{deg(v)-1}\right)$$
$$Z_m^{uv} = \frac{1}{2}\left[erf\left(\frac{W_{\text{cutoff}}(v) - \mu(X_m^{uv})}{\sqrt{2}\sigma(X_m^{uv})}\right) - erf\left(\frac{W_{\text{cutoff}}(v) - w(u,v) - \mu(X_m^{uv})}{\sqrt{2}\sigma(X_m^{uv})}\right)\right]$$

We parallelize this algorithm using two map-reduce phases.

The first map-reduce phase of this algorithm is given in Algorithm 15. The only difference between the first phase of game 1 and game 5 algorithms is the way in which the marginal contributions are calculated. So, the mapper does the same job as Algorithm 4. The input to the reducer is nodes and their neighbors. Since, weighted graphs will be the input for Game 5, we take the count of the occurrences of neighbor which will give the weight of the edge between the node and the neighbor. Marginal contribution of a node to itself is calculated as per the lines 5-11 of Algorithm 14 in lines 5-6 of the reduce phase of Algorithm 15. Marginal contribution of a node to its neighbor is calculated as per the lines 12-21 of Algorithm 14 in lines 7-10 of the reduce phase

of Algorithm 15. Once the marginal contributions are calculated, the reducer writes the nodes and the corresponding marginal contributions to the output.

Algorithm 15 First Map-Reduce Phase of Game 5

```
class MAPPER
    method MAP(lineNumber, edge)
        (nodeA, nodeB) = edge.split()
        EMIT(nodeA, nodeB)
        EMIT(nodeB, nodeA)

class REDUCER
    method REDUCE(node, list(neighbors))
        (neighbor,edgeWeight) ← neighbors
        degree = length(neighbor)
        weightedDegree = sum(edgeWeight)
        marginalContribution = contribution(threshold)
        EMIT(node, marginalContribution)
        for all neighbor ∈ neighbors do
            marginalContribution = contribution(threshold, edgeWeight)
            EMIT(neighbor, marginalContribution)
        end for
```

The second map-reduce phase of this algorithm is essentially the same as that of the second phase of game 1 (Algorithm 5) which aggregates the marginal contributions to obtain the centrality.

5.3 Experimental Results

The experiments in this section are designed to show

1. that the game-theoretic centrality measures perform on par with the greedy algorithm in the context of information diffusion.
2. the scalability of the parallelized algorithms with respective to the size of the input graph.

3. the scalability of the algorithms with respect to the number of machines in the hadoop cluster.

Cascade Experiment

Influence spread was explored in the literature in two different contexts. One is top k problem and the other is λ coverage problem. In the top k problem, a set of k nodes that will maximize the influence spread in the network has to be selected. In the λ coverage problem, the minimum value of k that is needed to obtain the desired influence spread has to be found. We have chosen the context of top k problem for our experiment. We did the cascade experiment to find the influence spread of the top k nodes given by five game theory based centrality algorithms and the greedy algorithm. We have used the linear threshold model to find the influence spread.

Influence Model : In this model every node can influence the neighbor nodes and every neighbor node can influence the node. Consider the edge (u, v) from node u to node v in a graph. Let d_v be the sum of weights of incoming edges of node v. In an unweighted network, d_v becomes degree of node v. Let $C_{u,v}$ be the weight of the edge from u to v. Then, node u has an influence of $\frac{C_{u,v}}{d_v}$ on node v. Note that the sum of influences on any node is 1. A node can influence its neighbors only if it is influenced already. There will be initial set of influencers who are assumed to be influenced by external forces. This model assumes that every node in graph has a threshold associate with it. A node is said to be influenced if sum of influences on that node by its neighbors is greater than threshold of the node. Neighbors of the newly influenced node might get influenced based on the above criteria. The influence spread stops when there is no more node in the graph which could be influenced. The threshold of the nodes assumed by this model cannot be determined in the real world scenarios. So, a number of different sets of thresholds are sampled from an appropriate probability distribution and the influence

spread is run for every set of threshold. The average of all these runs gives the number of nodes influenced. Typically, several different thresholds are sampled and run to cancel out the effect of threshold.

Greedy Algorithm : Consider a network with n nodes. Greedy algorithm runs the cascade spread n times with every node as a initial influencer. The node which is able to influence most number of nodes in the network is considered to be the top 1 influencer. Then it considers the rest of $n-1$ nodes and runs the cascade experiment n-1 times with top 1 influencer and every one of the n-1 nodes as initial influencers. The node which when combined with the top 1 influencer influences most number of nodes is added to the set of top influencers and thus top 2 influencers are formed. This process is repeated k times to find the top k influencers of the network. The problem of finding the exact top k nodes is hard. Greedy algorithm is the best approximation for this problem [11] .

We ran the cascade experiment in the collaboration network data. The network data can be obtained from `http://snap.stanford.edu/data/ca-GrQc.html`. This network is between collaborations of authors. Nodes in the network correspond to the authors. If an author u had collaborated with an author v in at least one paper, then an undirected edge is established between nodes u and v in the network. This network consists of 5242 nodes and 14496 edges. Figure 5.4 shows the plot of the number of nodes influenced against the number of initial influencers in this network. We observe that greedy algorithm performs well when the number of initial influencers is very less and the performance of game theoretic centrality algorithms increases as the number of initial influencers is increased. At k = 30, Game 4 and Game 5 perform marginally better than the greedy algorithm while Game 1 performs close to the greedy algorithm. Though Game 2 and Game 3 (dotted lines in Figure 5.4) algorithms give a slightly lesser performance than greedy algorithm, the difference is not enormous. We also note that the greedy algorithm doesn't scale well in terms of size of the network

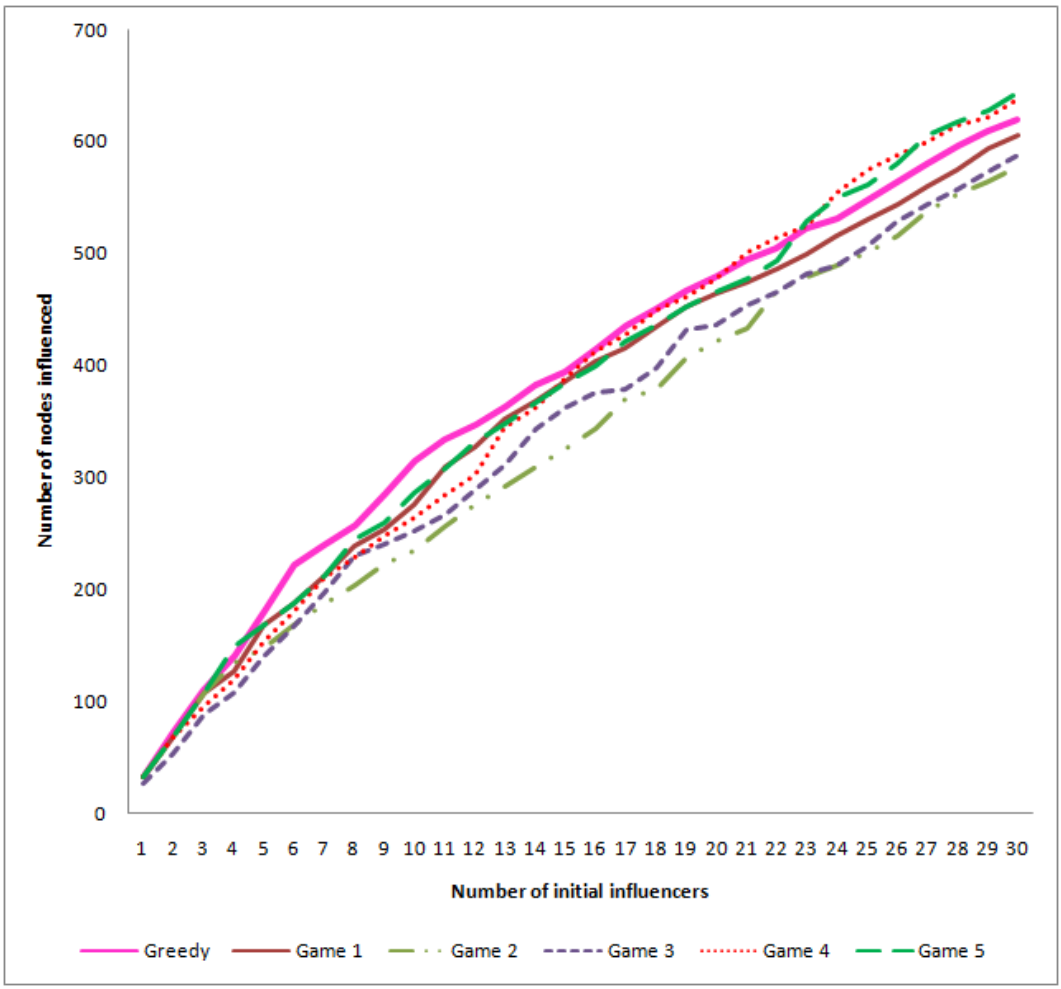

Figure 5.4: Result of cascade experiment in collaboration network

size as well as the number of initial influencers. Detailed analysis of the game theoretic centrality algorithms is done by Aadithya et al. [14] and Narayanam and Narahari [19].

Scalability Experiments

We generated synthetic networks to analyze the scalability of our algorithms. The synthetic networks were generated using the barabasi-albert model and Erdos-Renyi model.

Barabasi-Albert model: This model starts with small seed graph. This seed graph

needs to be connected i.e.,there should be a path from every node in the graph to every other node in the graph, to make sure that the final graph will be connected. The average degree of nodes in the network can calculated by the formula,

$$\text{Average Degree } = \frac{2 * \text{ Number of edges in the graph}}{\text{Number of nodes in the graph}}$$

Let x denote half of the average degree. The number of nodes in the seed graph should be a little higher than x. For every other node in the network, x nodes have to be chosen from the seed graph and all the x nodes should be connected to this new node. The seed graph grows with the addition of every node and once every node gets added to the seed graph, it becomes the final graph. Note that the probability with which the x nodes in the seed graph gets selected is proportional to the degree of the nodes of the seed graph i.e., higher the degree, more is the probability of getting selected. So, the resulting graph will have few nodes with high degree and many nodes with average degree.

Erdos-Renyi(E-R) model: This model generates network with the probability of existence of the edge. Depending on the density of the edges needed in the graph, the probability of existence of an edge can be calculated as follows.

$$\text{Probability of existence an edge, } P = \frac{2 * \text{ Number of edges in the graph}}{\text{Number of possible edges in the graph}}$$

Every edge in the graph is included with probability P and discarded otherwise. We observe that the graph will have no edges if P is zero and all the edges if P is 1.

Scalability with respect to input size: We ran our parallelized algorithms in a hadoop cluster having 10 machines with each machine having 8 cores. We have varied the size of the input graph and have calculated the running time of these parallelized algorithms. Table 5.1, Table 5.2, Table 5.3, Table 5.4, Table 5.5 and Table 5.6 shows the running time of parallelized version of the Game 1, Game 2, Game 3, Game 4 and Game 5

algorithms in seconds for different input sizes and densities of the graphs.

Table 5.1: Running time of Game 1 on E-R graphs of different densities in seconds

Network Size (# of edges)	Density of edges			
	0.01	0.001	0.0001	0.00001
10^3	28	28	28	29
10^4	31	29	29	29
10^5	31	31	31	33
10^6	45	46	46	45
10^7	84	85	79	74

Table 5.2: Running time of Game 2 on E-R graphs of different densities in seconds

Network Size (# of edges)	Density of edges			
	0.01	0.001	0.0001	0.00001
10^3	29	29	30	28
10^4	30	28	29	29
10^5	32	29	31	33
10^6	41	46	45	44
10^7	83	80	81	74

Table 5.3: Running time of Game 3 on E-R graphs of different densities in seconds

Network Size (# of edges)	Density of edges			
	0.01	0.001	0.0001	0.00001
10^3	43	42	41	43
10^4	43	45	44	44
10^5	49	46	49	46
10^6	109	89	70	70
10^7	1616	913	301	167

Table 5.4: Running time of Game 4 on E-R graphs of different densities in seconds

Network Size (# of edges)	Density of edges			
	0.01	0.001	0.0001	0.00001
10^3	42	43	44	42
10^4	46	43	45	45
10^5	51	50	45	46
10^6	110	89	70	71
10^7	1735	886	312	176

Table 5.5: Running time of Game 5 on E-R graphs of different densities in seconds

Network Size (# of edges)	Density of edges			
	0.01	0.001	0.0001	0.00001
10^3	30	28	29	28
10^4	30	30	30	29
10^5	33	32	30	32
10^6	45	45	46	43
10^7	86	114	79	83

Table 5.6: Running time of all the game theoretic algorithms on Barabasi Albert graphs of density 0.1 in seconds

Network Size	Game 1	Game 2	Game 3	Game 4	Game 5
10^3	33	30	45	46	30
10^4	31	29	47	48	30
10^5	31	30	93	90	39
10^6	60	59	146	141	67
10^7	73	79	5128	5354	128

Scalability with respect to number of CPUs: We ran our parallelized algorithms in a hadoop cluster by varying the number of machines in the cluster for a fixed input size. We chose the number of edges in the input graph to be 1 million for running game 3 and game 4 while we chose it to be 10 million for game 1, game 2 and game 5. Table 5.7 and Table 5.8 shows the running time of the algorithms when the number of machines in the cluster is varied. Each machine in our hadoop cluster had an 8 core CPU. So, 64 mappers or reducers can be run at a time.

Table 5.7: Running time of game 1, game 2 and game 3 algorithms in seconds on a 10 million edge network

Number of machines	Game 1	Game 2	Game 5
1	101	103	167
2	94	82	113
3	92	85	114
4	90	84	102
5	87	84	97
6	84	84	97
7	83	83	96
8	82	84	96

Table 5.8: Running time of game 1, game 2 and game 3 algorithms in seconds on a 1 million edge network

Number of machines	Game 3	Game 4
1	632	606
2	289	313
3	206	213
4	176	179
5	166	172
6	146	144
7	137	141
8	146	141

From the above experiments, we observe the following:

- We can process the networks of million edges in few seconds using the parallelization techniques even if the algorithm has a quadratic time complexity(e.g., Game 3 and Game 4).
- Game 1, Game 2 and Game 5 are highly scalable as these algorithm take only few seconds for running on network sizes which are as large as few millions of edges.
- The running time of the algorithms decrease as the density of edges in the graph decreases. This is due to the fact that every node in the sparse graph will have less number of neighbors when compared to the nodes in the dense graphs.
- The running time of the reduces as the number of machines in the cluster is increased.

CHAPTER 6

Conclusions and Future Work

This thesis has characterized the necessary features required for a good community scoring function and has shown that both the internal and external properties of the communities have to be taken into account in order to design a good scoring function. It is the first to propose a scoring function having explicit components to represent the internal and external score and then combining them to get the actual community scoring function. A variable α can be introduced in the external score and it can be modified as $\frac{a_s}{a_s+\alpha b_s}$. The relative importance of internal score and the external score can be controlled using this parameter and hence it can be made specific to applications. Earlier algorithms to find communities were resolution limit free but were non-scalable. Several methods addressed the problem of scalability but each one of them suffer from it's own limitations. CEIL algorithm addresses both the problem of scalability and resolution limit. Hence, it is a better alternative to find communities in large networks.

The greedy algorithm selects a good top k with a minimum guarantee but has an exponential complexity. Though the traditional centrality measures are of polynomial time complexity, the size of the network we come across these days have limited their applicability also. Parallelization of the polynomial time algorithms has given rise to a better alternative for degree centrality and page rank centrality to find centralities in large networks.

Scalability is an issue in every field nowadays. This thesis has addressed it in two major contexts in the field of social network analysis - community detection and centrality.

APPENDIX A

Results of perturbation experiments

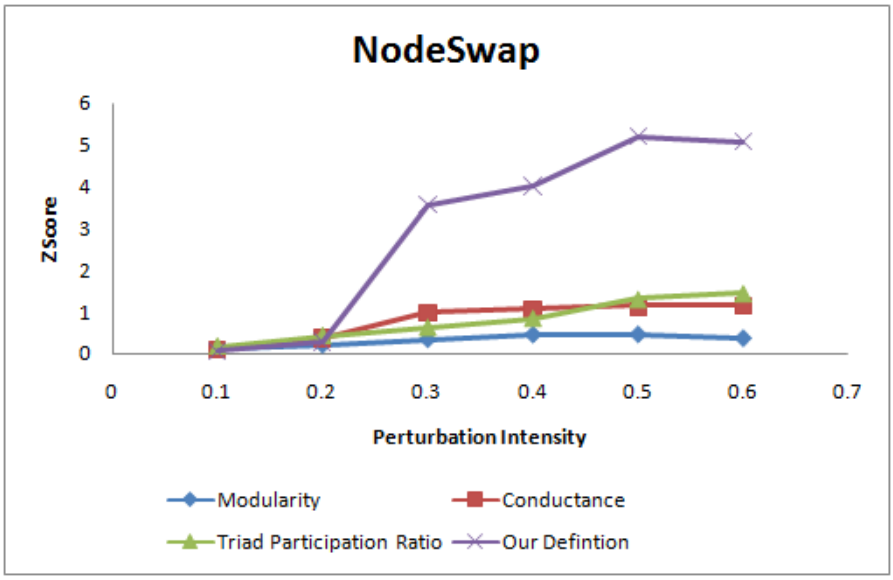

Figure A.1: Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under NodeSwap perturbation.

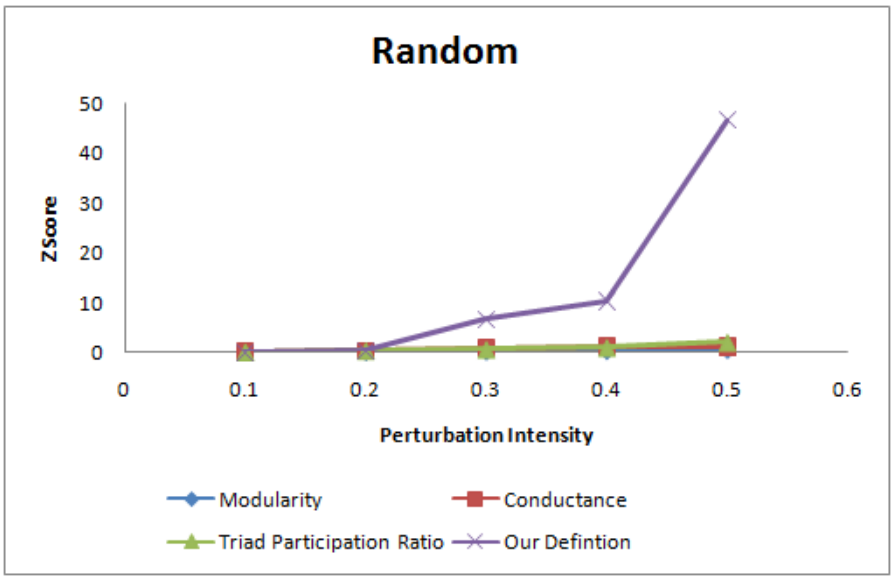

Figure A.2: Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Random perturbation.

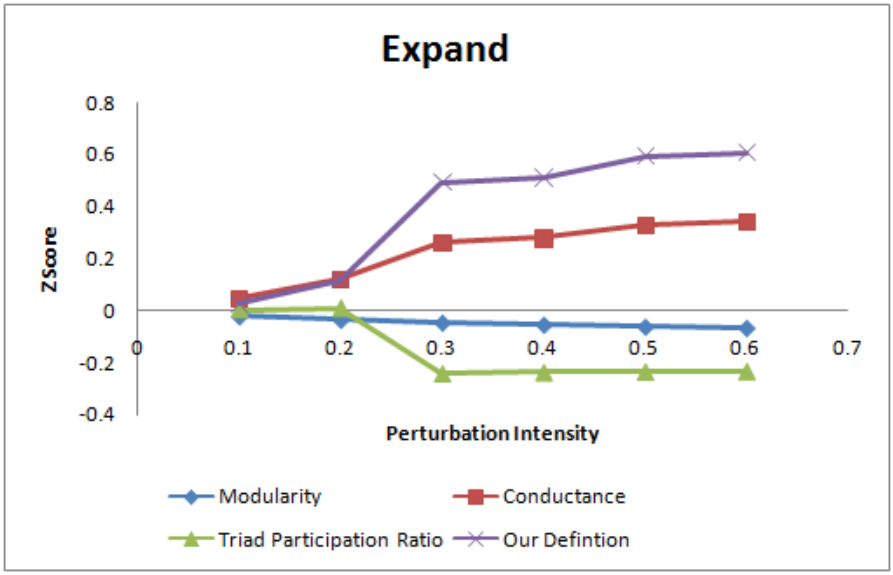

Figure A.3: Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Expand perturbation.

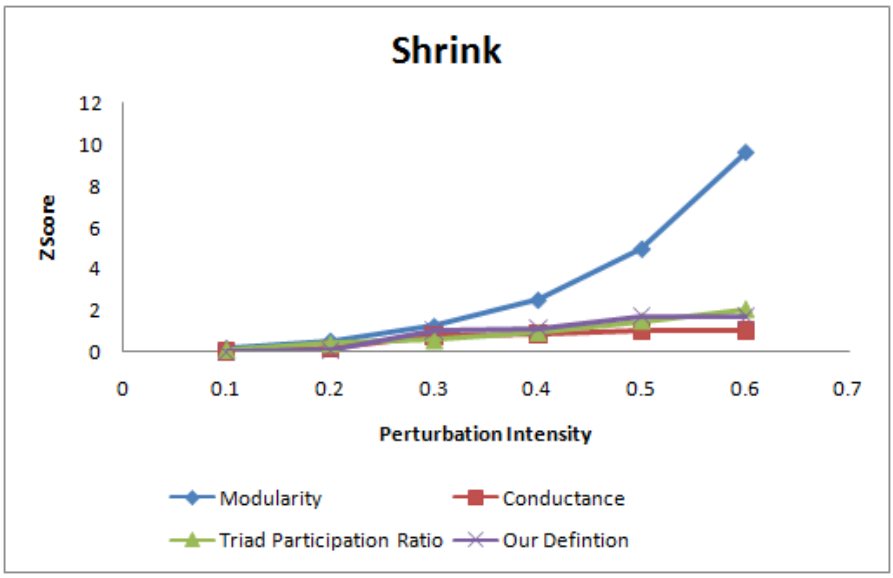

Figure A.4: Z-score given by various scoring functions as a function of perturbation intensity in Youtube network under Shrink perturbation.

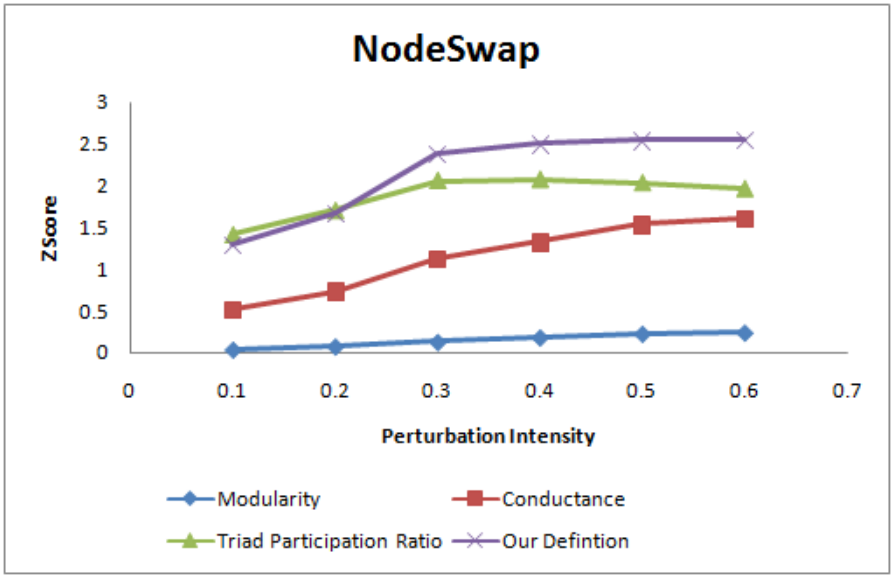

Figure A.5: Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under NodeSwap perturbation.

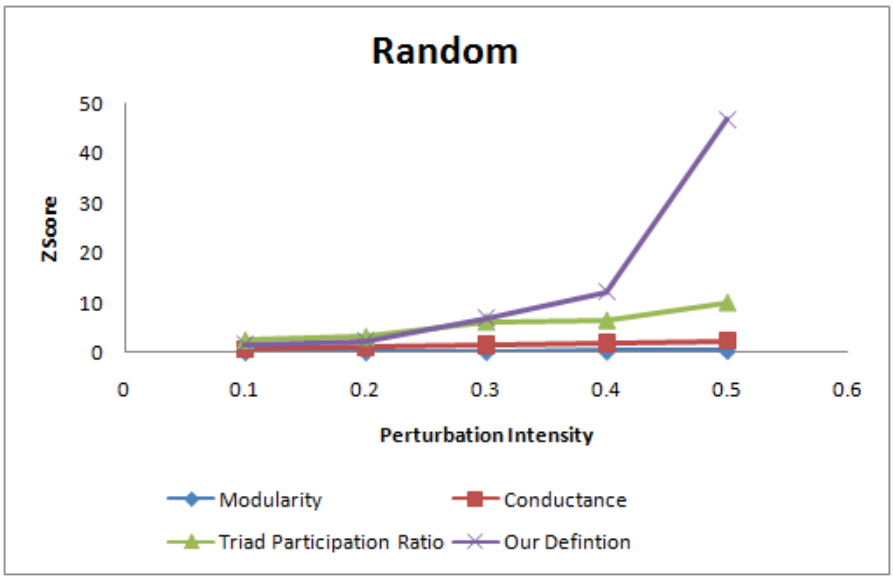

Figure A.6: Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Random perturbation.

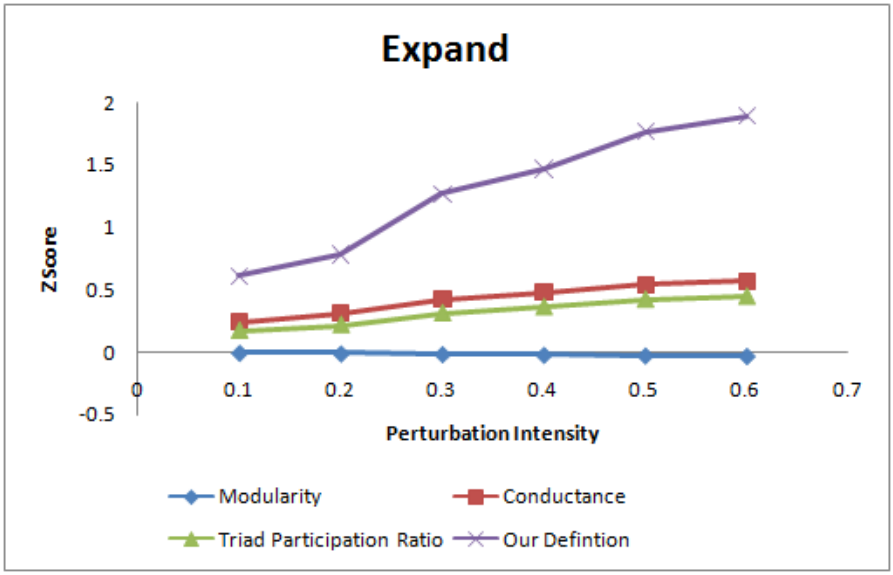

Figure A.7: Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Expand perturbation.

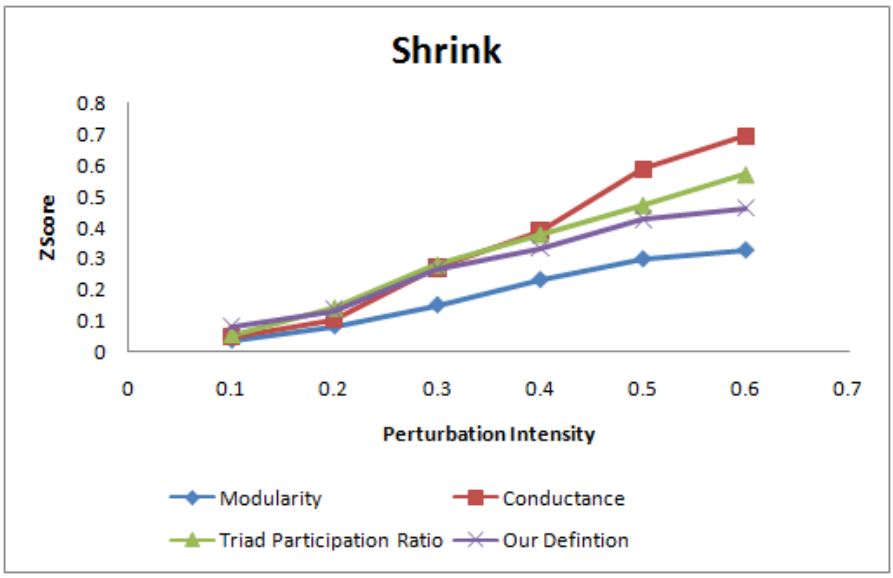

Figure A.8: Z-score given by various scoring functions as a function of perturbation intensity in DBLP network under Shrink perturbation.

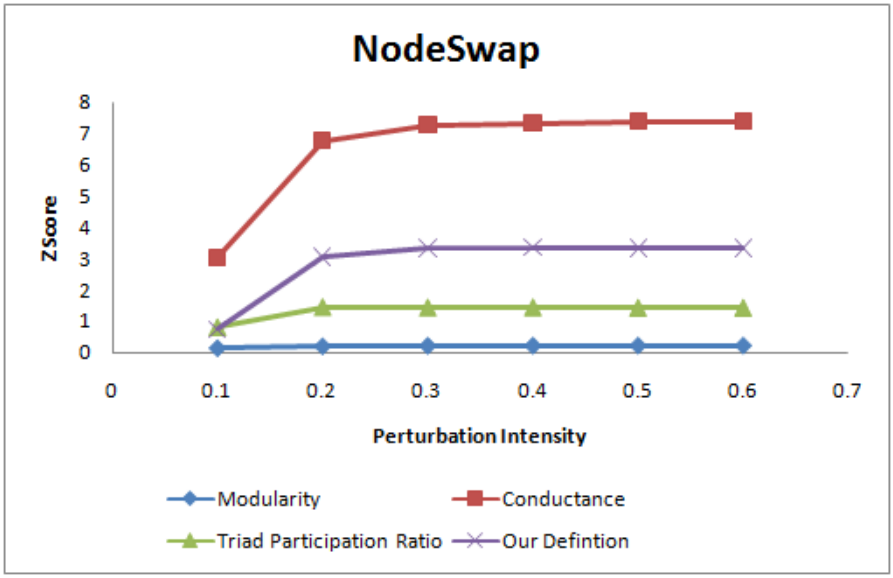

Figure A.9: Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under NodeSwap perturbation.

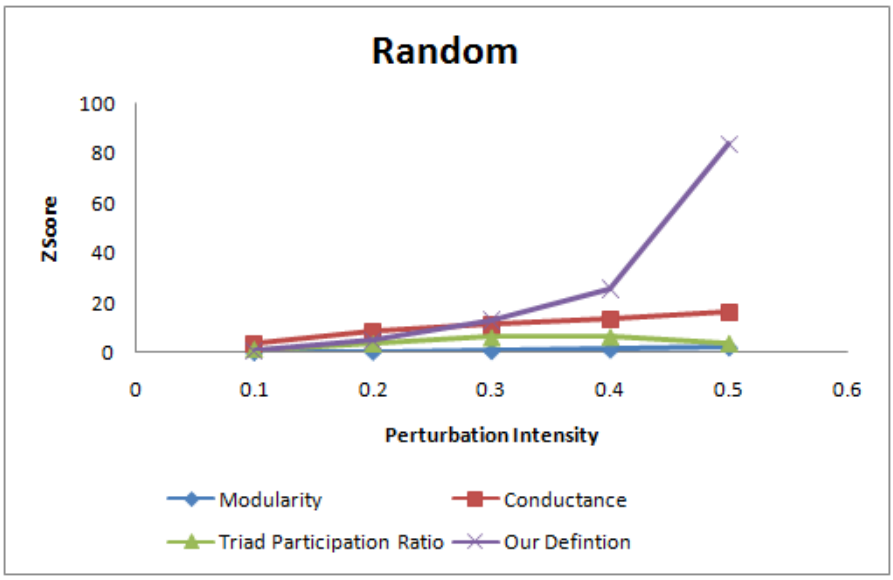

Figure A.10: Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Random perturbation.

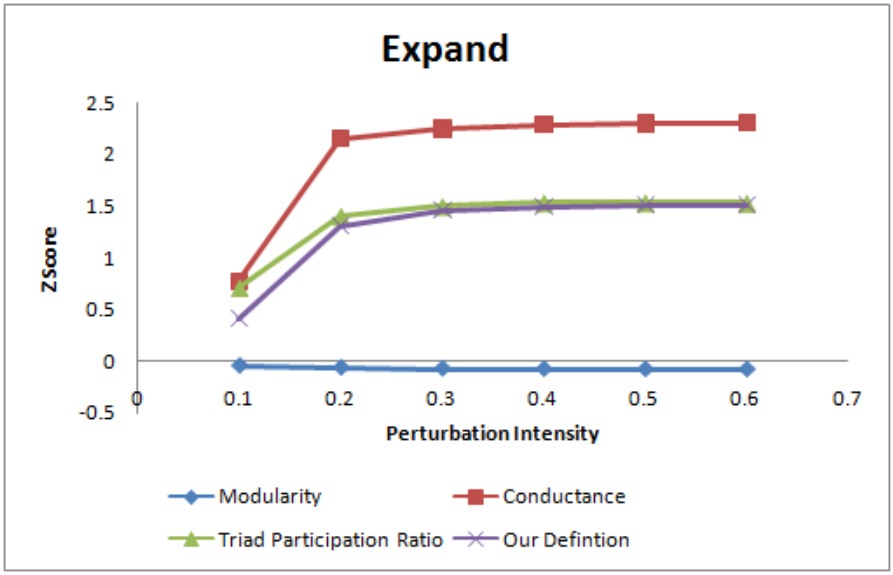

Figure A.11: Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Expand perturbation.

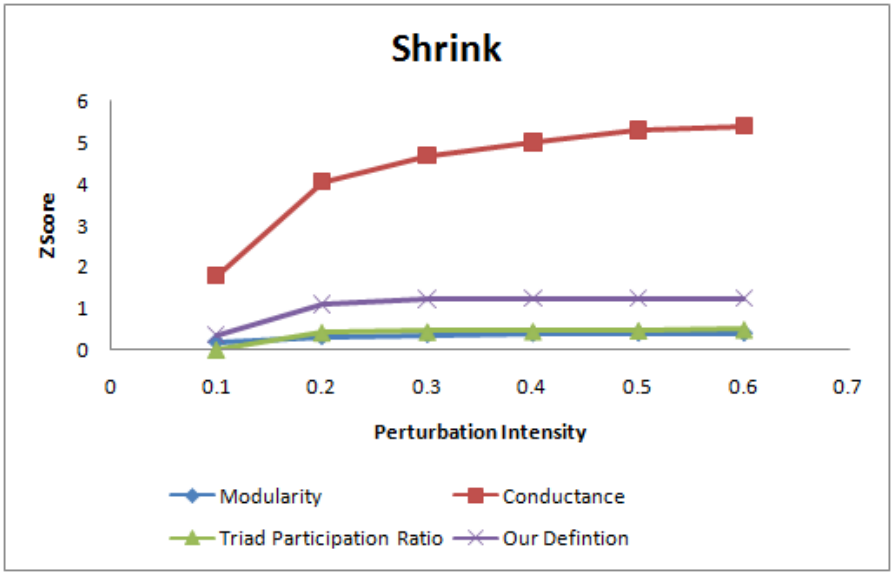

Figure A.12: Z-score given by various scoring functions as a function of perturbation intensity in Amazon network under Shrink perturbation.

References

[1] Alex Arenas, Alberto Fernandez, and Sergio Gomez. Analysis of the structure of complex networks at different resolution levels. *New Journal of Physics*, page 053039, 2008.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, page P10008, 2008.

[3] D. Borthakur. The hadoop distributed file system: Architecture and design. *Hadoop Project Website*, 2007.

[4] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *arXiv:physics/0608255*, 2006.

[5] Tanmoy Chakraborty, Sriram Srinivasan, Niloy Ganguly, Animesh Mukherjee, and Sanjukta Bhowmick. On the permanence of vertices in network communities. *Conference on Knowledge Discovery and Data Mining*, pages 1396–1405, 2014.

[6] L. Danon, A. Díaz-Guilera, J. Duch, and A. Arenas. Comparing community structure identification. *Journal of Staistical Mechanics*, page P09008, 2005.

[7] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Conference on Symposium on Operating Systems Design and Implementation*, pages 107–113, 2004.

[8] S. Fortunato and M. Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, pages 36–41, 2007.

[9] Santo Fortunato. Community detection in graphs. *Physics Reports*, pages 75–174, 2010.

[10] Apache Software Foundation. Apache hadoop. *http://hadoop.apache.org/*, 2011.

[11] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

[12] Andrea Lancichinetti and Santo Fortunato. Limits of modularity maximization in community detection. *Computing Research Repository*, abs/1107.1155, 2011.

[13] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Conference on World Wide Web*, pages 695–704, 2008.

[14] Tomasz Michalak, Karthik V. Aadithya, L. Szczepanski, Balaraman Ravindran, and Nicholas R. Jennings. Efficient computation of the shapley value for game-theoretic network centrality. *Journal of Artificial Intelligence Research*, pages 607–650, 2013.

[15] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Conference on Internet Measurement*, pages 29–42, 2007.

[16] J. L. Myers and A. D. Well. *Research Design and Statistical Analysis*. Lawrence Erlbaum Associates, 2003.

[17] Amit Anil Nanavati, Siva Gurumurthy, Gautam Das, Dipanjan Chakraborty, Koustuv Dasgupta, Sougata Mukherjea, and Anupam Joshi. On the structural properties of massive telecom call graphs: findings and implications. In *Conference on Information and Knowledge Management*, pages 435–444, 2006.

[18] Ramasuri Narayanam and Y. Narahari. Determining the top-k nodes in social networks using the shapley value. *The International Foundation for Autonomous Agents and Multi Agent Systems*, pages 1509–1512, 2008.

[19] Ramasuri Narayanam and Yadati Narahari. A shapley value-based approach to discover influential nodes in social networks. *IEEE Transactions on Automation Science and Engineering*, pages 130–147, 2011.

[20] M. E. J Newman and Michelle Girvan. Mixing patterns and community structure in network. *arXiv:cond-mat/0210146*, 2002.

[21] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, page 026113, 2004.

[22] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, page 066133, 2004.

[23] M. E. J. Newman. Spectral methods for network community detection and graph partitioning. *Computing Research Repository*, abs/1307.7729, 2013.

[24] Michael Ovelgönne and Andreas Geyer-Schulz. Cluster cores and modularity maximization. In *International Conference on Data Mining Workshops*, pages 1204–1213, 2010.

[25] Michael Ovelgonne and Andreas Geyer-Schulz. An ensemble learning strategy for graph clustering. *American Mathematical Society*, pages 187–206, 2013.

[26] Gergely Palla, Imre Derenyi, Illes Farkas, and Tamas Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, pages 814–818, 2005.

[27] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, pages 191–218, 2006.

[28] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences*, pages 2658–2663, 2004.

[29] Usha N. Raghavan, Reka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *arXiv:0709.2938*, 2007.

[30] W.M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, pages 846–850, 1971.

[31] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 888–905, 2000.

[32] Jaewon Yang and Jure Leskovec. Defining and evaluating network communities based on ground-truth. In *International Conference on Data Mining*, pages 745–754, 2012.

[33] Qiaofeng Yang and Stefano Lonardi. A parallel edge-betweenness clustering tool for protein-protein interaction networks. *International Journal of Data Mining and Bioinformatics*, pages 241–247, 2007.

Publications and Patents

Patent

1. M. Vishnu Sankar, Balaraman Ravindran and S. Shivashankar, “Method for finding communities in large networks," *US pat. 14/143283 (Application Number)*, 2014.

Publication

1. M. Vishnu Sankar, Balaraman Ravindran and S. Shivashankar, “CEIL: A scalable, resolution limit free approach for detecting communities in large networks," *International Joint Conference on Artificial Intelligence*, 2015 *(Communicated)*

2. M. Vishnu Sankar and Balaraman Ravindran, “Parallelization of Game Theoretic Centrality Algorithms," *Sadhana, Academy proceedings in Engineering Sciences, Special issue on machine learning for big data*, 2015 *(Communicated)*

Curriculum Vitae

Name : Vishnu Sankar M

Date of Birth : 15^{th} December 1988

Education : Bachelor of Engineering,
Electronics and Communication Engineering,
College of Engineering, Guindy,
Anna University,
Chennai.

Address : 711 B/5, Ettayapuram Road,
Opposite to Mahindra Two Wheeler,
Kovilpatti - 628501
Tuticorin District.

GTC Members

Chairman : Dr. C. Pandu Rangan
Department of Computer Science and Engineering,
Indian Institute of Technology, Madras.

Guide : Dr. Balaraman Ravindran
Department of Computer Science and Engineering,
Indian Institute of Technology, Madras.

Members : Dr. N.S. Narayanaswamy
Department of Computer Science and Engineering,
Indian Institute of Technology, Madras.

: Dr. Ashwin Mahalingam
Department of Civil Engineering,
Indian Institute of Technology, Madras.