11-2013

Detecting Stable Communities In Large Scale Networks

Sriram Srinivasan
University of Nebraska at Omaha

Detecting Stable Communities In Large Scale Networks

A Thesis

Presented to the

Department of Computer Science

and the

Faculty of the Graduate College

University of Nebraska

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

University of Nebraska at Omaha

By

Sriram Srinivasan

November, 2013

<u>Supervisory Committee:</u>

Dr. Sanjukta Bhowmick

Dr.Robin A. Gandhi

Dr.Qiuming Zhu

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Detecting Stable Communities In Large Scale Networks

Sriram Srinivasan

University of Nebraska, 2013

Advisor: Dr . Sanjukta Bhowmick

Abstract

A network is said to exhibit community structure if the nodes of the network can be easily grouped into groups of nodes, such that each group is densely connected internally but sparsely connected with other groups. Most real world networks exhibit community structure.

A popular technique for detecting communities is based on computing the modularity of the network. Modularity reflects how well the vertices in a group are connected as opposed to being randomly connected. We propose a parallel algorithm for detecting modularity in large networks.

However, all modularity based algorithms for detecting community structure are affected by the order in which the vertices in the network are processed. Therefore, detecting communities in real world graphs becomes increasingly difficult. We introduce the concept of *stable community,* that is, a group of vertices that are always partitioned to the same community independent of the vertex perturbations to the input. We develop a preprocessing step that identifies stable communities and empirically show that the number of stable communities in a network affects the range of modularity values obtained. In particular, stable communities can also help determine strong communities in the network.

Modularity is a widely accepted metric for measuring the quality of a partition identified by various community detection algorithms. However,a growing number of researchers have started to explore the limitations of modularity maximization such as resolution limit,degeneracy of solutions and asymptotic growth of the modularity value for detecting communities. In order to address these issues we propose a novel vertex-level metric called permanence. We show that our metric permanence as compared to other standard metrics such as modularity, conductance and cut-ratio performs as a better community scoring function for evaluating the detected community structures from both synthetic networks and real-world networks. We demonstarte that maximizing permanence results in communities that match the ground-truth structure of networks more accurately than modularity based and other approaches. Finally,we demonstrate how maximizing permanence overcomes limitations associated with modularity maximization.

Table of Contents

1. Introduction …… 1

1.1Contribution……...2

1.2Outline of thesis ……....2

2. Background …… 4

2.1GraphTerminology …….....4

2.1.1 Vertex Based Properties……**5**

2.1.2 Network Based Properties…….....**6**

2.2.1 Community Detection ……**7**

2.2.2 Normalized Mutual Infomration ……**8**

2.2.3 LFR networks…….....**9**

3. Parallelizing the Louvain Method for Modularity Maximization……...10

3.1 Introduction …….....10

3.2 Background……..11

3.3 Louvain Method……...12

3.4 Shared Memory Algorithm for Parallelizing the Louvain Method…….14

3.5 Empirical Results ……..15

3.5.1 Scalability Results…….....17

3.5.2 Evaluation of correctness …….....17

3.6 Discussion …….....19

4. Stable Communities……20

4.1 Introduction…….20

4.2 Sensitivity of Community Structure to Vertex Perturbation …….20

4.3Stable Community for Improving the Modularity…….....29

4.4 Discussion……...34

5. Detecting Stable Communities for Maximization of Modularity……36

5.1 Introduction……..36

5.2 Related Research……..37

5.3Detecting Stable Communities in Complex Networks…….37

5.4 Modularity Maximization Using Stable Communities…….....41

5.5 Shared Memory Algorithm…… 46

5.6 Discussion ……...47

6. Detecting Communities Using Relative Permanence as a Metric……...48

6.1 Introduction ……....48

6.2 Related Research , Network Datasets and Ground Truth Communities……..49

6.3 Permanence ……51

6.3.1 Evaluating Community Scoring Functions.....54

6.4 Senitivity of Permanence ……...56

6.4.1 Community Detection Based On Permanence ……58

6.4.1.1 Algorithm Overview……..58

6.4.2 Performance Evaluation ……...60

6.5 Permanence Resolving Issues related with Modularity Maximization ……..61

6.5.1 Terminology…….....62

6.5.2 Discussion on Issues in Modularity Maximization……...63

6.5.3 Discussion……65

7.Conclusion and Future Work ……66

8.References…….....68

List of Figures

Figure 2.1: Undirected Graph**……5**

Figure3.1:Scalability Results for Parallel Louvain Method**……17**

Figure 3.2: Variability in Modularity across Processors**……...18**

Figure 4.1: Sensitivity of each network across 5000 permutations**……23**

Figure 4.2: Comparison between relative size ……**24**

Figure 4.3: Schematic diagram…….....**26**

Figure 4.4: Distribution of relative permanence values**…….....29**

Figure 4.5: Variation of NMI for different values of mixing parameters**….....33**

Figure 4.6 : Modularity after partially collapsing the stable communities**……**.....**34**

Figure 5.1. Partition of network into communities**……**….....**38**

Figure 5.2: Modularity Values for the Dolphin Network**……**….....**44**

Figure 5.3: Modularity Values for the Power Network**……**.....**45**

Figure 5.4: Strong Scalability**……**…….....**47**

Figure 6.1: Toyexample to measure permanence of vertex v…….....**52**

Figure 6.2: Computing the values of community scoring functions……...**55**

Figure 6.3: Toy examples demonstrating four cases**…**…….....**62**

List of Tables

Table 4.1: Networks……21-22

Table4.2: Modularity before and after preprocessing for real-world networks…….31

Table4.3: Modularity before and after preprocessing for LFR networks…….……..32

Table 5.1: Network Description…….....41

Table5.2 :Comparision of Modularity values using CNM method….....44

Table5.3: Comparision of Modularity values using Louvain method.....45

Table 5.4: Comparison of Execution Time……..45

Table5.5: Node and Edge counts for networks…….46

Table 6.1: Real world network properties…….....50

Table 6.2: Performance of community scoring function…….....50

Table 6.5: Average improvement of our algorithm…….……61

Chapter 1

Introduction

Networks consist of a set of vertices and a set of edges and have been proven to be useful for solving real world problems arising in systems of interacting objects. In a network model, vertices represent objects and edges represent interactions between them. In the study of networks such as social networks[24] and biological networks it has been found that networks have common characteristic[24] like community structure and heavy tailed degree distribution[24]. A network is said to have community structure if the nodes of the network can be easily grouped in to set of nodes such that each set of nodes is densely connected internally and sparsely connected externally[26].

A fundamental problem in network analysis is detecting communities correctly. Most community detection algorithms are based on optimizing a combinatorial metric, for example modularity [26] and conductance [27]. The goodness of community detection algorithm is often measured according to how well they achieve optimization. Optimization is generally NP- hard thus merely changing the ordering of the vertices influences the community structure detected by any community detection algorithm. In my thesis we study the effect of vertex perturbation on the community structure detected using Louvain et.al[3] and Clauset et.al[4].

However there exist a group of vertices which are not affected by any vertex perturbation, we call those set of vertices as stable community. We study various characteristics of stable community and design an algorithm to identify such community. In the next part of my thesis we have implemented a parallel version of the popular

modularity maximization approach called the Louvain method, which iteratively optimizes local communities until overall modularity can no longer be improved. In this process we discovered the modularity and other metrics like conductance suffer from a resolution limit which makes it difficult to detect communities which is smaller in size. We propose a new metric termed as relative permanence which overcomes the effect of the resolution limit. In the final part of my thesis we develop a new algorithm to detect communities using relative permanence as a metric.

1.1 Contribution

Given below is a list of our significant contributions.

- We have carried out comprehensive research on different community detection algorithms that use modularity maximization and studied the effects of vertex perturbations on them.
- We have designed an efficient constant community detection algorithm for static networks that detects group of vertices which are not affected by vertex perturbations.
- We designed and developed a new metric called relative permanence to detect community in static networks.

1.2 Outline of Thesis

This thesis is organized as follows. In chapter 2 we discuss background of graph theory and community detection using modularity maximization. In chapter 3 we present the parallel version of the popular modularity maximization approach known as the Louvain method. In chapter 4 we discuss the effect of vertex perturbation on

the results of community detection algorithms and judge the goodness of a community detection algorithm. In chapter 5 we present our new constant community detection algorithm, which overcomes vertex perturbation.

In chapter 6 we discuss demerits of modularity maximization and propose a new metric relative permanence to detect community in networks. In chapter 7 we present our concluding remarks and present potential ideas for future research.

Chapter 2

Background

Many problems of practical interest can be represented as graphs. In computer science, graphs are used to represent different networks such as biological networks and social networks[24]. Each of these networks consists of a set of vertices and a set of edges. For instance people in the social networks represent vertices in a graph and connections between people are represented by the edges in social networks. Here, we introduce some network or graph terminology. We classify the list of graph properties as (i) vertex based properties, and (ii) network based properties.

2.1 Graph Terminology[25]

A graph is collection of vertices and edges. Formally, G=(V,E) consists of set of vertices V and a set of edges W, where E is subset of $(V \times V)$. In general graphs are classified as directed and undirected. A graph is directed if edges point in one direction from one vertex to another vertex, otherwise the graph is undirected.

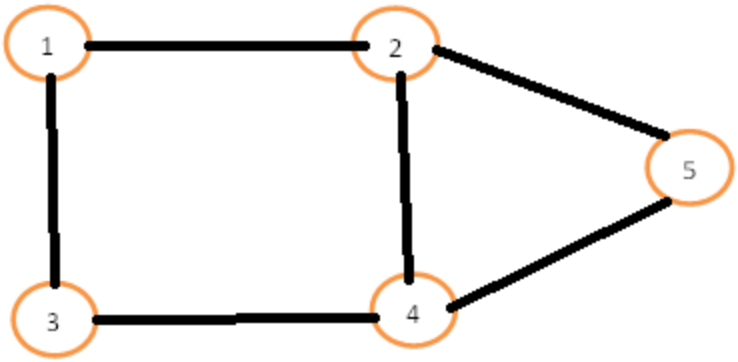

Figure 2.1: Undirected Graph

Graph Properties

2.1.1 Vertex Based Properties

- **Degree**

  The degree of a vertex in a graph is the number of edges the vertex shares with the other vertices. The degree of vertex v is denoted by deg(v). In a directed graph, vertices have two different degrees, in-degree: number of incoming edges and out-degree: the number of outgoing edges. In figure 2.1, degree of vertices are deg(1)=2, deg(2) =3, deg(3)= 2, deg(4)=3 and deg(5)=2.

- **Clustering Coefficient**

  Clustering coefficient is a measure of the degree to which the nodes in a graph tend to cluster together. Clustering coefficient is calculated as the ratio of the edges between the neighbors of a vertex to the total number of possible connections between them. In general, the higher the clustering coefficient the

more likely that vertex is part of a dense module. Mathematically clustering coefficient of a vertex V is defined as,

$$C_i = \frac{2 \times n_i}{k_i \times (k_i - 1)}$$

Where n_i denotes the number of connections connecting the k_i neighbors of vertex i to each other.

2.1.2 Network Based Properties

- **Degree Distribution**

Degree distribution is the distribution of the different degrees (and their frequency) of the vertices over the network. Most scale free networks like social networks observe a power law distribution [5] that is there exist many vertices with low degree and the number of vertices exponentially go down as the degree increases.

- **Modularity**

Modularity is a metric to determine how good a network is partitioned into communities. Newman and Girvan proposed this metric to judge the goodness of a community detection method. Modularity is based on the conception that random networks do not form strong communities. Given a partition of a network in to M groups, let C_{ij} represent the fraction of total connections starting at a node in group I and ending at group j.

Let $a_i = \sum_j C_{ij}$ corresponds to the fraction of connections connected to subgroup i. Probability of edges begin at i is a_i, probability of edges that end at node j is a_j. Internal connections or within-community links of group i is a_i^2. Total number of actual edges within each group i is C_{ii}. Comparison of actual and expected values, summed over all partitions gives us modularity. $Q=\sum(C_{ii}-a_i^2)$. In general high modularity gives us the better estimation of community structure in the network. Maximizing modularity is a popular method for finding communities in networks. However finding maximum modularity is an NP-hard problem [26]. There exist many heuristics for maximizing modularity. However our research focuses on two popular agglomerative modularity maximization algorithms.

2.2.1 Community Detection

A network is said to have clusters if vertices of the network can be grouped into a set of vertices such that each set of vertices are densely connected internally. Community detection is a fundamental problem in network analysis. Newman and Girvan [3] proposed a greedy algorithm based on maximizing the modularity metric for detecting community. Clauset, Newman and Moore [4] (popularly known as CNM) proposed fast implementation of a previous technique proposed by Newman et al[3]. The CNM method is a greedy algorithm. This algorithm initially considers each vertex in network as individual community. At each iteration pair of communities with high increase in modularity is merged. This process is repeated until there exist no combination of vertices that show increase in modularity.

Blondel et.al [3] proposed a faster and efficient method to detect communities. In this approach all vertices are initially assigned as an individual community like CNM method. However instead of a search over all edges, Louvain method searches over the edges of each vertex. Each vertex is combined with the neighbor that shows highest increase in modularity. In subsequent steps of the iteration neighbor itself can be detached from its original community and join new one. Allowing vertices to be removed from earlier communities, the Louvain method provides mechanism for rectifying bad choices. Process of reassigning communities is repeated over several iterations until modularity is increased. Once the first phase allocation of vertices is completed in second phase it aggregates vertices belonging to same community and network is formed whose nodes the communities. Two steps are repeated iteratively until modularity converges.

While comparing Louvain method and CNM method Louvain method is generally faster than two becomes it executes a combination for each vertex if possible. However CNM method finds maximum over all edges per iteration. Another advantage of Louvain method is to withdraw or backtrack from community if found necessary.

2.2.2 Normalized mutual information (NMI)

NMI is used to compare how good partitions produced by each approaches when compared against the ground truth. Let C be the confusion matrix, and N_{ij} represent the element at row I and column j . N_{ij} denote the number of nodes in the intersection of original community I and the generated community j. if C_A denote number of communities in ground truth , C_B number of communities generated by an approach, N_i sum of row I, N_j the sum of column j, and N sum of all elements in C, then NMI score

between the ground truth partition A and the generated partition B can be computed as shown in following equation.

$$NMI(A,B) = \frac{-2\sum_{i=1}^{C_A}\sum_{j=1}^{C_B} N_{ij}\log\frac{N_{ij}N}{N_i N_j}}{\sum_{i=1}^{C_A} N_i \log\frac{N_i}{N} + \sum_{j=1}^{C_B} N_j \log\frac{N_j}{N}}$$

NMI value ranges between 0 and 1. 0 refers there is no match between with ground truth and 1 refers to perfect match.

2.2.3 LFR networks

For our experiments we have used LFR benchmark model[18] to generate artificial networks with a community structure[3]. LFR model allows us to control following properties: number of nodes n, desired average degree k, maximal degree k_{max}, exponent γ for degree distribution , exponent β for the community size distribution, and mixing coefficient μ. The latter represents average proportion of links between a node and nodes located outside its community, called intercommunity links. Portion of intra community links is 1- μ. For our experiments we mostly vary nodes (n) and μ is varied from 0.1 to 0.6 remaining parameterswe use default values mentioned in implementation of Lancichinetti and Fortunato[18].

Chapter 3

Parallelizing the Louvain Method for Modularity Maximization

3.1 Introduction

A popular method for finding communities in a network is by maximizing modularity. Modularity measures how better the vertices in a community are connected as opposed to a random connection as discussed in chapter2. As network size increases, it is difficult to store them in memory so it is essential to develop parallel implementations for the modularity maximization algorithms.

Parallel algorithms for graphs are a well-researched topic. There exist few parallel algorithms for modularity maximization[12,13,14]. Most agglomerative methods for obtaining high modularity require frequent synchronization, which reduces the scope of parallelization. In addition we have observed results of modularity maximization are affected by vertex perturbation. Therefore it is difficult to evaluate the accuracy of a parallel algorithm.

In this chapter, we present a shared memory parallel algorithm for the Louvain method. We are the first to introduce a parallel implementation of the original Louvain method. In Section 3.2 we discuss some of the existing parallel algorithms for modularity maximization. In Section 3.3 we describe the Louvain method. In Section 3.4 we discuss

a simple shared memory algorithm for parallelizing the Louvain method. In Section 3.5 we discuss scalability and correctness of our results.

3.2 Background

Detecting communities using modularity maximization can be affected by the resolution limit, that is, the algorithms are unable to detect communities smaller than a certain size [5]. The Louvain method[2] addresses this problem by creating a hierarchy of communities with the smaller ones discovered in initial iterations followed by larger ones in subsequent iterations. This somewhat reduces the effect of the resolution limit problem, compared to the CNM algorithm.

As networks increase in size, it is essential to use parallel algorithms to handle large data. In our research on parallelizing modularity maximization algorithms we discovered there are only two approaches. The first implementation is based on label propagation by Raghavan et.al [11]. In this algorithm, initially all vertices are assigned a unique label and with subsequent iterations the vertices adopt labels of their neighbors to denote the community to which they belong. Label propagation is based on local updates. A highly scalable implementation of this algorithm has been produced for GPGPUS by Soman et.al [12].

The second implementation is based on the algorithm proposed by Clauset et.al [4]. In this method each vertex is initially assigned to a separate community. In each subsequent iteration the pair of vertices with the highest edge weight are combined. Reidy et.al [13] implemented this algorithm (on CRAY XMT and Open MP).

3.3 Louvain Method

Assume that we start with a weighted network of N nodes. First each node of the network is assigned to a different community. So, in this initial phase there are as many communities as there are nodes. Then for each node i we consider its neighbors j and we evaluate change in modularity that would take place by separating i from its community and placing it in the community of j. The node i is then placed the community for which change in modularity is maximum, but only if change is positive. If no positive gain is possible, i stays in its original community. This process is repeated for all nodes until no further improvement can be achieved. This simple algorithm improves the agglomerative process of modularity maximization due to two major contributions.

First contribution is to increase the speed instead of considering all vertex pairs; the Louvain method considers only maximum increase in modularity amongst every vertex and its neighbors.

Second contribution is to improve flexibility Louvain methods attempts to improve on modularity maximization by removing vertices from their assigned communities and evaluating if modularity can be improved by re- assigning the vertex to any of the other neighboring communities. This process is repeated over several iterations. These two features of Louvain method should be preserved by any parallel algorithm. Algorithm 1 provides the pseudo code for the Louvain method.

Algorithm 3.1: Louvain Method for Modularity Maximization

Input: - A Graph G= (V, E). Vector A to store fraction of edges of each community.

Output: - A vector VID for mapping vertices to communities Q to store value of modularity.

1. **Procedure** INITIALIZATION
2. Int=0
3. Degree =A // Store Values of A in degree
4. Q= - $\sum_{v=1}^{[V]} A[v]^2$
5. Old_Q=Q-1 // Initialize Modularity Value
6. **for all** v ∈ V **do** // Assign individual communities to each vertex
7. set VID[v].node=v
8. set VID[V].comm=v
9. Set Total_comms to [V]
10. **Procedure** Louvain Method
11. **while**old_Q<Q do
12. Old_Q=Q // Beginning Phase 2
13. **while**It_int<Total_its do // Beginning Phase 1
14. **for all** C<Total_comms do // Going through Each Community
15. Set Cur_comm to c // Initialize current community of c
16. //Remove C from Curr_Comm
17. Set dQ to increase in modularity by adding C to Cur_Comm
18. Q=Q-dQ
19. //Find best community for C
20. Find set of neighboring communities N_c of C
21. Max_dQ=dQ =dQ
22. **for all** n ∈N_c**do**
23. Compute dQ_n, change in modularity by adding c to n
24. **if**dQ_n>Max_dQ then
25. Set New_comm to n
26. Move c to New_Comm
27. A[cur_Comm]= A[cur_comm]-Degree[Cur_Comm]
28. A[New_comm]= A[New_Comm]+ Degree[Cur_Comm]
29. **for all** v ∈ V **do**
30. **if** VID[V].comm=curr_Comm**then**
31. VID[v].comm=New_Comm
32. Update Q= Q+ Max_dQ // End pf Phase 1
33. Combine communities to supervertices
34. Total_Comms= max(VID.comm)
35. Reduce size of A to only contain valid communities

3.4 Shared Memory Algorithm For Parallelizing The Louvain Method

In this section we describe our parallel implementation of the Louvain method for modularity maximization. We choose the regions with loops such as for and while as they are the most natural part of code to exhibit parallelism. We have parallelized most of the initialization process such as assignment of values to degree and assignment of vertices to communities. Now we consider areas of iteration. We first consider the while loop at line 13 and then two other regions within the while loop which can be parallelized. The first is at Line 20 where we find the set of neighboring communities N_c. In this operation at first we find the neighbors of the vertices within community c, and then the communities of the neighbors to N_c. We can implement this process in parallel for each vertex. In the next section of code we can parallelize the module for finding the best community amongst the members of N_c. Change in modularity , dQ_n due to adding c for each neighboring community n can be computed in parallel. If we store the dQ_n of each community in a data structure like array or vector , then finding the maximum increase in modularity becomes a reduction operation. Finally after detecting the most suitable community to join we can update the assignment of communities to vertices in a critical section.

Based on this analysis we discovered that in phase 1 (Lines 13- 33) the update of edges associated with each community, A vector (line 17 and Line 28-29), the community assignment, VID vector(Line 30 – 32), and modularity Q ,(Line 19, Line 33) needs to be computed sequentially due to this the parallel potential of the code is reduced.

We can further improve our approach by reducing a few operations such as avoiding computation of Q in phase 1. We can compute Q in phase 2 using community assignment stored in VID and we can update the value of Q, where it will be a perfect parallel operation. The value of dQ with respect to current community is already being computed earlier, we can avoid that computation in (Lines 23- 27). Operations in (Line 28 -32) needs to be performed only if a vertex is moved from its earlier community, i.e if New_Comm is different from Cur_Comm. These updates are implemented as atomic operations on A. This ordering ensures that communities are combined only when modularity is increased.

We discovered in the second phase there is less scope for parallelization, and this depends on the technique of operation. For example, to detect vertices belonging to the same community, we sort the vector based on increasing order of communities such that vertices within the same community are arranged consecutively. Sorting operation can be done in parallel using parallel merge sort algorithm. Algorithm 2 provides pseudo code for parallel implementation of Louvain algorithm.

3.5 Empirical Results

In this section we present our experimental results that demonstrate that the algorithm is highly scalable. We observed that if a network has a well-defined community structure, then the algorithm is faster and deviation amongst the timings and the values are less than networks with more unstructured communities. We implemented our algorithm on an Opteron quad-core system with only 8 GB RAM. Our Experimental setup as follows; we create set of LFR bench-marks [7] of 10,000 vertices with mixing parameters μ being

0.1,0.3,0.5 and 0.7. By lowering mixing parameter it is guaranteed to have more distinctive community distribution. We kept average degree 15, maximum degree 50 . The power –law exponent for degree distribution was 2 and exponent for community distribution was 1. Community size ranges from 7 to 50.

Algorithm 3.2 : Parallel Implementation Louvain Method.

Input: - A Graph G= (V, E). Vector A to store fraction of edges of each community.

Output: - A vector VID for mapping vertices to communities Q to store value of modularity.

1. **Procedure** INITIALIZATION
2. Int=0
3. Total_its=4 **The Number of outer iterations**
4. Degree =A **Values assigned in parallel**
5. Q= - $\sum_{v=1}^{[V]} A[v]^2$ **Obtained by parallel reduction**
6. Old_Q=Q-1
7. **for all** v ∈ V **do in parallel**
8. set VID[v].node=v
9. set VID[v].comm=v
10. Set Total_commsto [V]
11. **Procedure** Louvain Method
12. **While** old_Q<Q do
13. Old_Q=Q // Beginning Phase 2
14. **While**It_int<Total_its do // Beginning Phase 1
15. **for all** C<Total_comms do // Going through Each Community
16. Set Cur_comm to c // Initialize current community of c

//Remove c from Curr_Comm

17. Set dQ to increase in modularity by adding c to Cur_Comm
18. Find set of neighboring communities N_c of c **in parallel**
19. Max_dQ= dQ
20. Set New_Comm to Cur_Comm
21. **for all** n ∈N_c**do in parallel**
22. Compute dQ_n, change in modularity by adding c to n
23. **if**dQ_n>Max_dQ**then use parallel reduction**
24. Max_dQ= dQ_n
25. Set New_Comm to n
26. **if**Cur_Comm ¡= New_Comm then use atomic operations to update A
27. A[Cur_Comm]= A[Cur_Comm]- Degree[Cur_Comm]
28. A[New_Comm]= A[New_Comm]+ Degree[Cur_Comm]
29. **for all** v ∈ V do
30. **if** VID[V].comm=curr_Comm**then**
31. VID[v].comm=New Comm // End of Phase 1
32. Combine communities to supervertices**parallelmergesort**
33. Compute Q **in parallel**
34. Total_Comms= max(VID.comm)
35. Reduce size of A to only contain valid communities

3.5.1 Scalability Results

A parallel algorithm is scalable if execution time decreases as the number of processing units is increased. We performed an experiment by changing the number of threads from 2,4,8,16 and 32. In Figure 3.1 we show the execution time progressively decreases as the number of processing units are increased.

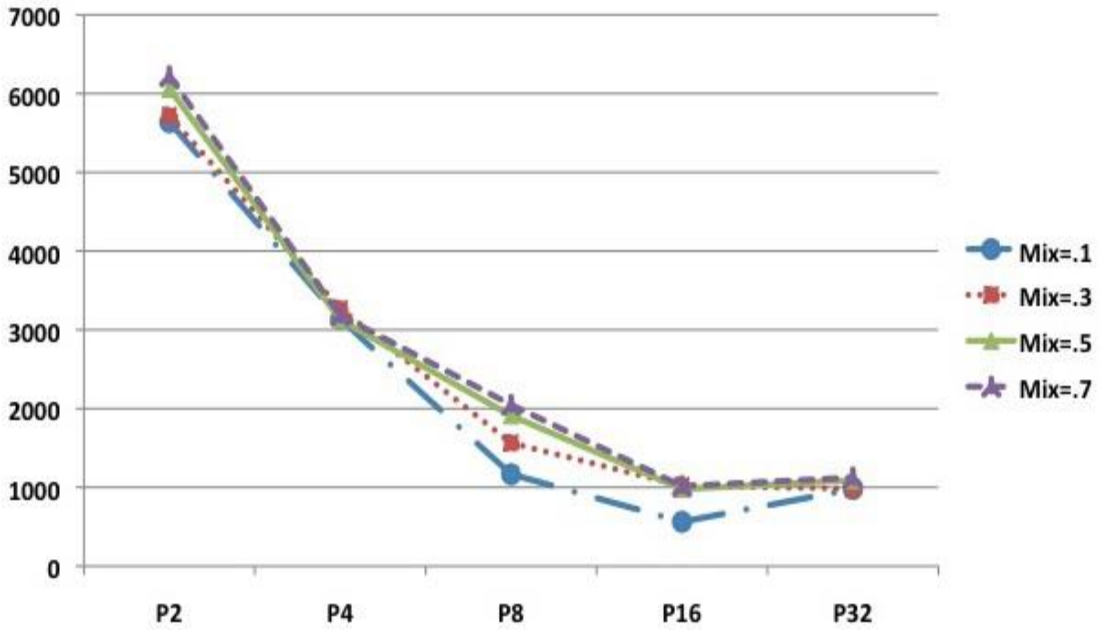

Figure3.1: Scalability Results for Parallel Louvain Method: Results for networks with 10 K vertices. Each point represents the total execution time of one network for a given mixing parameter and a processor.

3.5.2 Evaluation of Correctness

The empirical method for evaluating the correctness of parallel programs is by comparing the communities obtained by its sequential counterpart. However as mentioned earlier, results of Louvain method, like all other combinatorial optimization

techniques is dependent on the order in which vertices are processed. In other words it is impossible to compare results. The effect is further aggravated in the parallel case, as the sequence in which processors execute the code can change for each execution cycle. We compared the communities using normalized mutual information (NMI). NMI values range between 1 to 0, the higher the number the better the similarity between two sets of communities. In our experiments we observe that for lower mixing parameters NMI value across processor was around 0.90. For mixing parameter 0.7 the difference was as much as 0.76. Ordering of vertices (which is affected by parallelization), plays important role in the community distribution. Louvain method is ultimately designed to increase modularity. More accurate evaluation of our algorithm is to compare standard deviation of modularity value across each processor. In figure 3.2 we demonstrate the values of modularity and standard deviation across networks among processors. In general standard deviation values are quite low though the modularity values are more consistent when μ=0.1. In general lowering mixing parameter produce higher modularity.

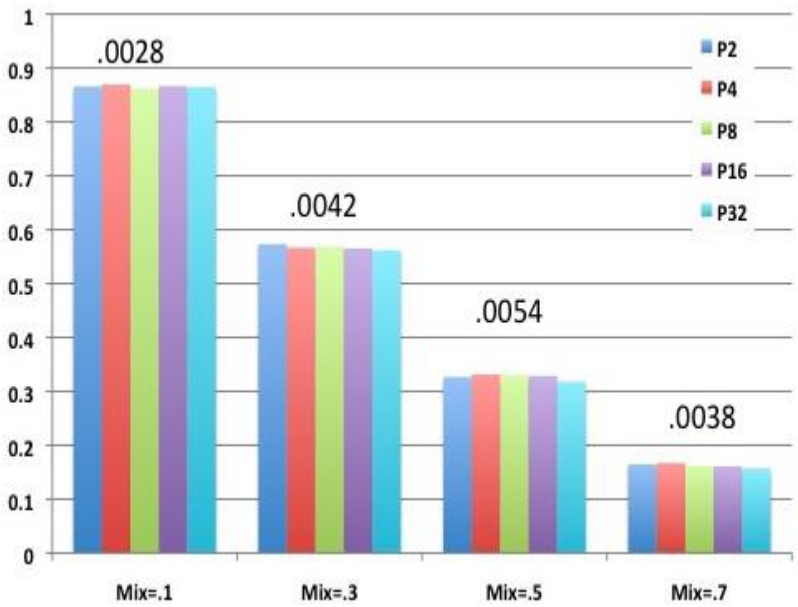

Figure 3.2: Variability in Modularity across Processors: Results for networks with 10 K vertices.

3.6 Discussion

In this chapter we presented a shared –memory algorithm for the Louvain method for modularity maximization. Our results indicate our algorithm is scalable and produces modularity values equivalent with those expected from sequential value. Performance of our algorithm and variability of the results depends on properties of networks and its size.

Chapter 4

Stable Communities

4.1 Introduction

In previous chapters, we have mentioned community detection algorithms are based on optimizing certain parameters such as modularity. Changing the order of vertices can vary their mapping to a community. There has been less study on how vertex ordering influences the results of community detection algorithms. In this chapter, we discuss the properties of groups of vertices whose mappings to communities are not affected by vertex ordering. This chapter is arranged as follows. In section 4.2 we discuss the sensitivity of community structure to vertex perturbation. In section 4.3 we discuss how detecting and using stable communities as a preprocessing step improves the modularity value.

4.2 Sensitivity of Community Structure to Vertex Perturbation

In this section we demonstrate that the modularity maximization method can significantly change the results. Based on our results we define metrics to estimate the tendency of a network to form communities. Finally we show that using stable communities as a preprocessing step can help improve the modularity of the community detection algorithm as a whole. We select two popular agglomerative modularity maximization techniques; CNM and the Louvain method which are discussed in chapter

2. In general the Louvain method produces a higher value of modularity than CNM, because it allows vertices to migrate across communities.

In order to detect these communities, for each network in the test suite, we applied CNM and the Louvain method over different permutations of the vertices and we preserved common groups across the different orderings. Common groups of vertices were marked as a stable community for each respective network. Ideally the total number of different orderings to be tested should be equal to the factorial of the number of vertices in the network. If you consider the smallest network in our set(Chesapeake with 39 vertices) this value is prodigious. We therefore restrict our permutations to maintain degree-preserving order. The vertices are ordered such that the degree of v_i is greater than the degree of v_j, then v_i is processed prior to v_j. The degree ordering permutation also has another advantage if few vertices in network have high degree and more have low degrees. Therefore arranging vertices with high degree guarantees that most of the fluctuations will occur towards the later stage of agglomeration.

We conducted experiments on real-world data as networks generated using LFR model as discussed in chapter2. We took real-world networks from the 10^{th} DIMACS challenge website. We considered the following undirected and unweighted networks:

Network	Size
Jazz	V=198, E=2742
Polbooks	V=105,E=441
Chesapeake	V=39,E=340
Dolphin	V=62,E=159

Football	V=115,E=1226
Celegans	V=453,E=2025
Power	V=4941,E=6594
Email	V=1133,E=5451

Table 4.1: Networks

Networks generated using the LFR model are associated with a mixing parameter μ that indicates the ratio of external connections of a node to its total degree. We created LFR networks based on the following parameters: number of nodes =500, average degree = 20, maximum degree =50, minimum community size =10, maximum community size =50, degree exponent power law =2, community size exponent = 2 and community size exponent = 3. We altered the value of μ from 0.05 to 0.90. In general low values of μ correspond to well separated communities that can be detected easily andthese networks contain a larger percentageof stable communities. As the value of μ increases, community structure gets ambiguous or amorphous and community detection algorithms give different sets of results.

We performed an experiment to study how the community structure of networks changes under vertex perturbations. We measure change in community structure based on the number of stable communities. We use sensitivity (ø **change this symbol, it means 'empty set'**) as the ratio of the number of stable communities to the total number of vertices. If ø is 1 each vertex itself will be a stable community (the trite case).The higher the sensitivity metric, the fewer the vertices in individual stable communities. This metric is helpful for detecting networks that have good community structure under modularity maximization.

We plot the sensitivity of each network in Figure 4.1. X-axis indicates the number of different permutations of the vertices and Y-axis plots the value of sensitivity. We observed for most of the networks the number of stable communities becomes does not increase within the first 100 permutations and sensitivity values are low. If sensitivity is low there exist strong groups in the network that have to be combined to obtain high modularity. For networks like Power grid and Email the number of stable communities keeps increasing until sensitivity reaches 1 or close. Community detection for those networks are extremely sensitive to vertex perturbations. This also indicates community structure in those networks is very amorphous.

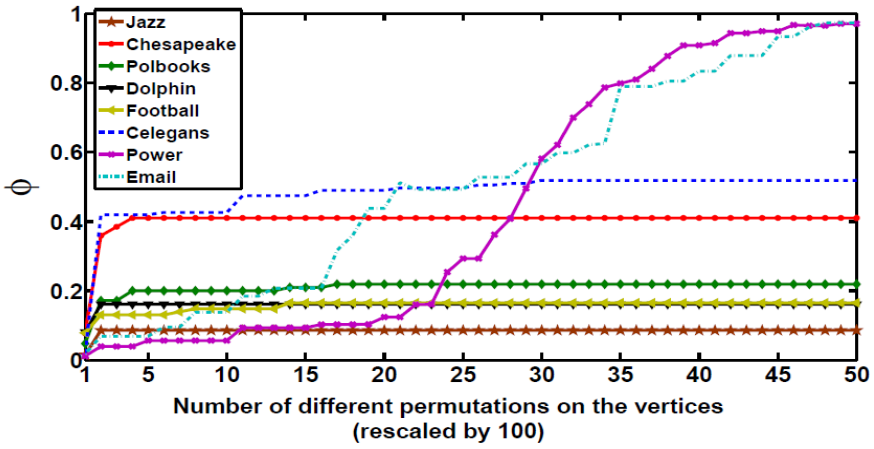

Figure 4.1: Sensitivity of each network across 5000 permutations

We investigate the properties of stable communities. Relative size (ξ) for a stable community is the ratio of the total number of nodes in the stable community to the total number of vertices in the network. Strength (Θ) is defined as ratio of the edges internal to the stable community to the edges external to the stable community.

In Figure 4.2 we plot the relative sizes of stable communities with respect to their strength. If the strength of a stable community in log scale is above 1 then the number of internal connections is larger than external connections. In general, the higher the value the more tightly connected the community. If the relative size of stable communities is low then the remaining vertices have freedom to migrate across other communities.

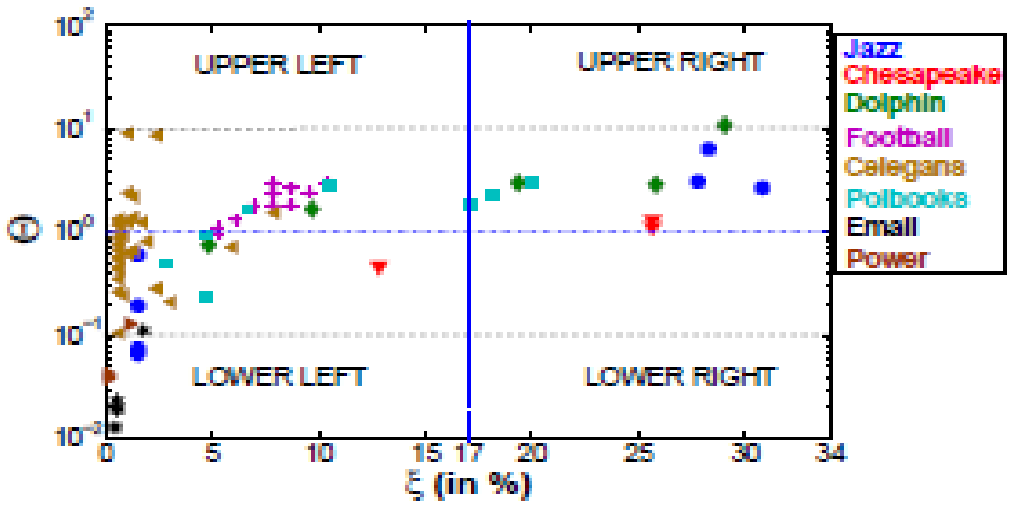

Figure 4.2: Comparison between relative size and strength of stable communities. X- axis indicates relative size in percentage and Y –axis indicates strength in log scale.

Relative size and strength together indicate the community structure of networks. When we divide X axis at 17 and Y-axis at 1 we get four quadrants. In the upper right quadrant communities have high size and high strength. In general if networks contain stable communities in this quadrant then they are less likely affected by perturbations. The third quadrant which is lower left contains communities of low relative size and low strength. Networks having communities from this quadrant will be significantly affected by vertex perturbations. In the upper left quadrant communities are strongly connected but have small relative size. This indicates there is some portion of the network with strong

community structure. The fourth quadrant represents communities that have high relative size and low strength.

In Figure 4.2 we noticed there are several communities whose strength is below one. It means there are more external connections than internal connections. In general, good community should have internal connections greater than external connections. Vertices within the community do not experience significant pull from any external communities. We mathematically define pull as follows:

Let v be a vertex in stable community, let D(v) denote degree of v and EN(v) and IN(v) denote number of internal and external neighbors of v, i.e., D(v)= IN(v)+ EN(v). EN(v) is divided in to k external groups. ENG(v) denotes a set of k elements. For example in Figure 4.3 D(3)=6, IN(3)=2 & EN(3) =4.ENG(3)= {2,1,1} (2 external neighbors in community 2, one external neighbors in community 3 and community 4). Similarly we can calculate ENG(v) for all vertices in the graph and form a list DEGN(G) by performing the union operation on ENG(V). The list is then ranked in ascending order. For a particular vertex if the inverse rank of each external group is equal to one it would point that all external neighbors are externally distributed. Therefore the pull experienced will be minimum. If the value is much lower than one it implies the vertex experiences strong pull from its external neighbors. Relative permanence can be expressed mathematically as:

$$\Omega(v) = \Theta(\mathrm{v}) \times \sum_{i=1}^{k} \frac{\frac{1}{Rank(ENG_i(v))}}{D(v)}$$

Where

Ω(v)= Relative permanence of vertex v.

Θ(v)= Strength of vertex v.

Using an example we have demonstrated how to calculate relative permanence of a vertex.

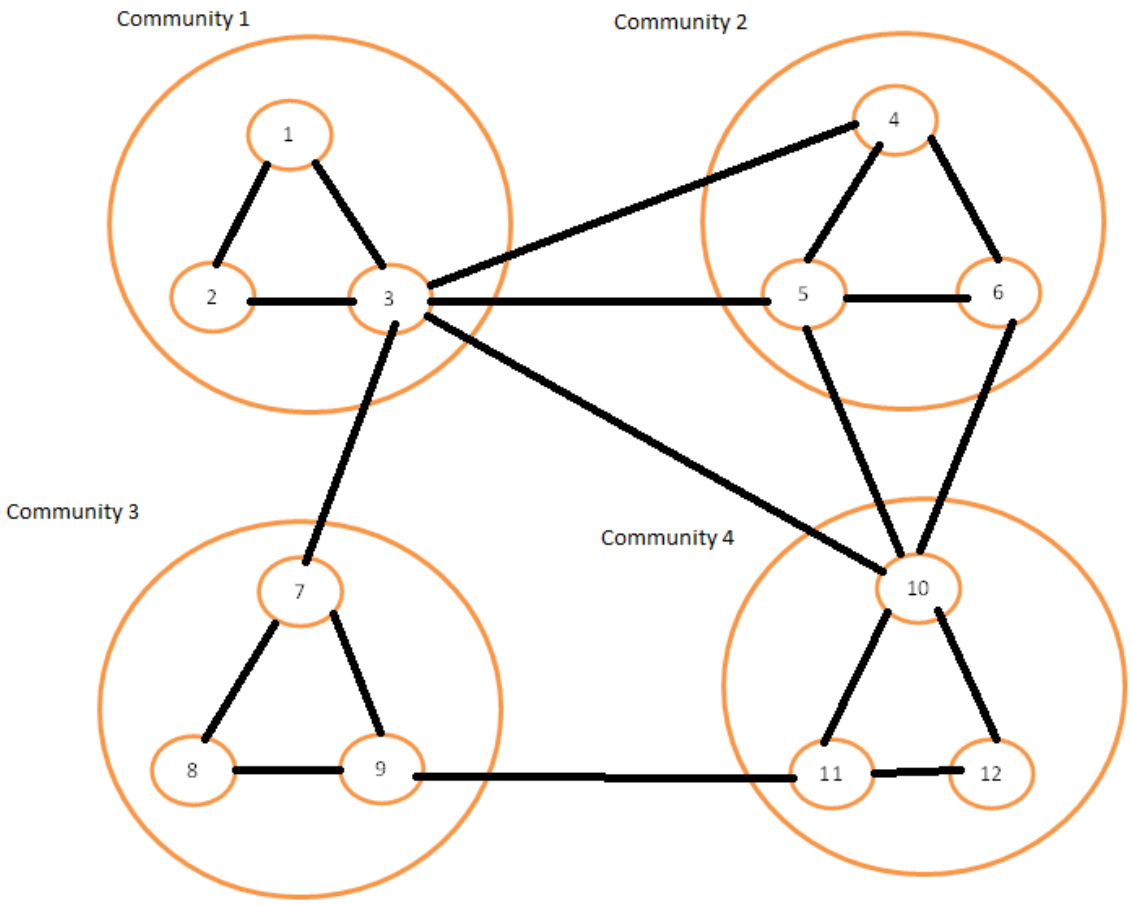

Figure 4.3: Schematic diagram illustrating computation of relative permanence of the vertices.

Using Figure 4.3 I calculate relative permanence for vertex 3 in stable community .

Vertex 3:- IN(v) = Internal Connections (with in Community)

EN(v) = External Connections (Connections Outside the Community)

I(3) = 2 [2 Connections with in community 1]. Equation (1)

E(3)= 4 [4 external connections]. Equation (2)

D(3)= Degree= I(3) + E(3) = 2 +4=6 Equation (3)

Now I compute ENG(V) that is the number of connections to other communities for vertex v.

ENG(V) is defined as Number of connections to external group.

ENG(3)= {2,1,1 } [Vertex 3 has 2 connection to community 2 , 1 connection to community 3 , 1 connection to community 4). Equation (4)

Relative permanence of a vertex is defined as

$$RP(v) = \frac{\frac{1}{ENG_i(v)}}{D(v)} \times \frac{I(v)}{E(v)}$$ ----------Formula(1)

From equation (4) I get ENG(3) . I use value of ENG(3) and then calculate

$$\sum_{i=1....k} \frac{1}{ENG(v)}$$

When k=1

$\sum_{i=1....k} \frac{1}{ENG(3)_{k=1}}$ =1/2----------Equation(5)

When k=2

$\sum_{i=1....k} \frac{1}{ENG(3)_{k=2}}$= 1/1=1 ---------- Equation(6)

When k=3

$\sum_{i=1....k} \frac{1}{ENG(3)_{k=3}}$= 1/1=1 ---------- Equation(7)

Now substituting values obtained from equation(5), equation (6) and equation (7) on Formula(1) we get

$$RP(3) = \frac{\frac{1}{2} + 1 + 1}{D(3)} \times \frac{I(3)}{E(3)}$$

From equation(1), equation(2) and equation(3) I get values for I(3) , E(3) and D(3) respectively .

$Relative\ permanence(3) = \frac{\frac{1}{2}+1+1}{6} \times \frac{2}{4} = \frac{2.5}{6} \times 0.5$=0.20833

Therefore Relative Permanence (3) = 0.208 .

Similarly relative permanence for all vertices is calculated using formula (1).

In Figure 4.4 we plot the cumulative distribution of the relative permanence over the vertices in all networks. The X-axis indicates the value of relative permanence and the Y-axis indicates the cumulative fraction of vertices having the corresponding value. The

cumulative distribution of vertices is roughly same across all networks except Email and Power. The cumulative distribution of Email and Power indicate these networks have lower relative permanence value and therefore experience more pull from external communities. A high fraction of vertices in Jazz, Polbooks, Dolphin and Celegans have relative permanence close to one. Therefore vertices in these networks experience less relative pull from external communities.

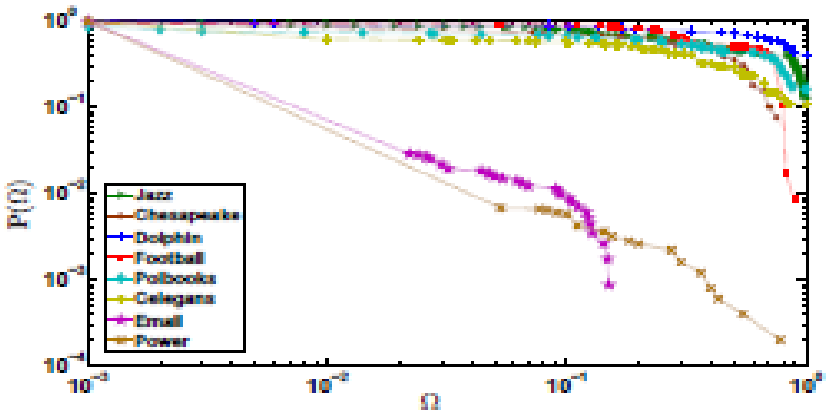

Figure 4.4: Distribution of relative permanence values**.** X-axis indicate the values of relative permanence and Y-axis indicate cumulative fraction of vertices which exhibits relative permanence.

4.3 Stable Community For Improving The Modularity

In our experiments we discovered stable communities are formed only by a small percentage of vertices. Finding stable communities is not sufficient as it may just provide inadequate information about the relationship amongst the rest of the vertices. We permute the vertices 5000 times in degree descending order as discussed in the previous section. For each permutation we run the Louvain algorithm and obtain community

structure and a modularity value. From this community structure we detect stable communities using algorithm 4.1.

Algorithm 4.1 : Modularity Maximization Using Stable Communities

Input: - A Graph G= (V, E); Community Detection Algorithm A.

Output: - Set of stable Community

1. **Procedure** Detect Stable Communities
2. Sort vertices in V degree descending order.
3. Apply degree preserving permutation P to vertices such that degree (v_i) >degree (v_{i+1}) in P.
4. |P| is number of degree preserving permutations applied.
5. Initialize array vertex [|V |][|P|] to -1
6. Vertex [|V|][|P|]will store the community membership of vertices in each permutation.
7. Set i=0
8. **for all $P_i \in P$ do**
9. Apply algorithm A to find communities of the permuted network G_{pi}
10. If vertex v is in community c then
11. Vertex[v][i]=c
12. Applying A to P_i
13. i=i+1
14. set j=0
15. for all $v \in V$ do
16. information stored in vertex
17. if vertex v is not in stable community then
18. create stable community CC_j
19. Insert v to CC_j
20. For all $u \in V \backslash CC_j$ do
21. If vertex[v][i]=vertex[u][i]
22. Insert u to CC_j
23. J=j+1

Initially vertices are ordered according to their degrees (Line 2). The permutations of the vertex preserve this order, that is vertex v_i is placed before v_j in the list if degree(v_i) > degree(v_j). In the next phase we detect communities for each permutation i. Stable communities are those vertices which are assigned together (Line 13-20).

Table 4.2 shows the mean modularity and variance obtained by averaging the modularity values of all iterations.

Networks	Before processing (Mean)	Before processing (Variance)	After processing (Mean)	After processing (Variance)
Jazz	0.448	3.13e-6	0.452	0
Chesapeake	0.301	1.17e-5	0.303	3.36e-33
Polbooks	0.539	1.74e-5	0.557	1.24e-32
Dolphin	0.543	1.76e-5	0.550	0
Football	0.610	2.01e-5	0.623	0
Celegans	0.438	2.89e-5	0.442	1.33e-26
Email	0.542	6.89e-5	0.568	0.95e-12
Power	0.936	1.09e-5	0.937	2.25e-10

Table4.2: Modularity before and after preprocessing for real-world networks.

As shown in Table4.2 combining stable communities as a preprocessing step both increases mean modularity. From our experiments on real –world networks we believe that preprocessing using stable communities is more effective if a network is not amorphous or has a strong community structure. To make our hypothesis stronger we created LFR graphs with mixing parameter from 0.05 to 0.90. In general low mixing parameter indicates good community structure. We repeat the same set of experiments as discussed on real world networks and obtain mean modularity and its variance. Table 4.3 shows the mean modularity and variance.

μ	Before processing (Mean)	Before processing (Variance)	After processing (Mean)	After processing (Variance)
0.05	0.834	1.98e-24	0.877	0
0.10	0.802	2.28e-28	0.817	0
0.20	0.690	5.74e-7	0.686	0
0.50	0.385	2.05e-6	0.389	1.58e-28
0.70	0.298	9.70e-10	0.219	1.04e-28
0.90	0.225	4.25e-10	0.205	5.64e-28

Table4.3: Modularity before and after preprocessing for LFR networks for different mixing parameter (μ).

As LFR networks have ground truth i.e., correct distribution of communities. We used NMI to compare the communities obtained, with and without using the preprocessing step with the ground truth community structure of LFR graphs with different mixing parameters. In Figure 4.5 when community structure is strong, stable communities push the result towards ground truth. In contrast when the network is amorphous or community structure is not well defined, the use of stable communities does not push the result towards ground truth.

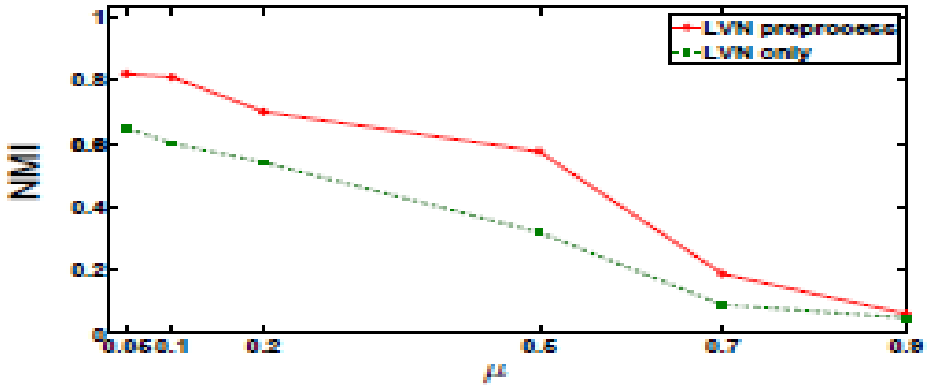

Figure 4.5: Variation of NMI for different values of mixing parameters. Broken line represents to the experiment without preprocessing step and solid line represents experiment with preprocessing step.

A stable community is meaningful if it is large in size and has high relative permanence. We ordered stable communities according to decreasing order of size and decreasing order of relative permanence. We combine stable communities into supper-vertices one by one following the order obtained from (a) and (b) separately. After the combination we compute modularity obtained using the Louvain method without any preprocessing.

Figure 4.6 distinguishes the modularity obtained by collapsing stable communities according to order obtain from (a) (dotted blue line) and (b) (dotted green lines). For all the networks there is a change when modularity values cross over the mean modularity (solid red line). After this change the modularity value is generally high or equal to mean modularity.

The critical point indicates the smallest fraction of stable communities required to outperform the Louvain algorithm without preprocessing i.e., original algorithm. In Figure 4.6 the broken green lines show a great increase in modularity value than the

broken blue lines after critical point. Therefore from our experiments we conclude relative permanence is better indication of stable community.

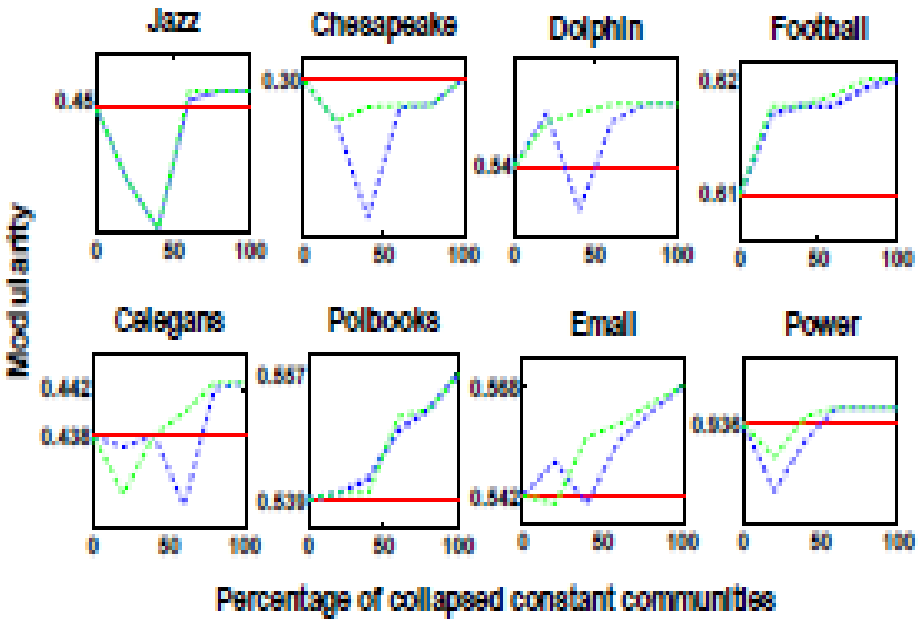

Figure 4.6 : Modularity after partially collapsing the stable communities. Blue (broken lines) are in decreasing order of size and green lines decreasing order of relative permanence.

4.4 Discussion

In this chapter we discussed the effect of vertex perturbation, how vertex perturbation affects community structure and stable communities. We performed experiments to show there exist stable communities in networks and using stable communities as a

preprocessing step to the original Louvain algorithm gives improvement in modularity value if network has good community structure.

Chapter 5

Detecting Stable Communities for Maximization of Modularity

5.1 Introduction

Modularity maximization is an NP- hard problem [3]. There exist many classes of heuristics to maximize modularity including agglomerative, diverse and spectral methods [3]. In general like other NP- hard combinatorial optimization problems, the value of modularity and the partition of vertices into communities are dependent on the order in which the vertices are processed.

We assume that if the network is not modular enough to be classified into communities then these instabilities may occur. Some portions of the network have a tendency to form natural communities, while the remaining vertices are mapped to communities based on combinatorial parameters of the underlying algorithms and permutations to the input. We define a stable community to be a group of vertices which are always mapped to the same community independent of the perturbations to the input. The number of stable communities can give a rough estimate of modularity. In this chapter, we discuss an algorithm to detect stable communities. We also demonstrate that combining vertices in stable communities as a preprocessing step to agglomerative community detection can improve the value of modularity.

The rest of the chapter is arranged as follows. In section 5.2 we discuss some related research in this area. In section 5.3, we present our algorithm to detect communities in networks. In section 5.4 we demonstrate using experimental results, on a test suite of

networks, how detected stable communities as preprocessing step can increase the modularity value. In section 5.5 we present the parallel template of our algorithm and applications to biological networks. In section 5.6 we conclude with discussions.

5.2 Related Research

The effect of perturbations of the input to the community detection is still a major issue. Karrer et.al[5] conducted a study by comparing change in community structures after perturbing the connectivity of the network. In chapter 4 we have perfomed experiments and discussed effects of vertex ordering and its effect on community structure.

5.3 Detecting Stable Communities in Complex Networks

Given a network, our objective is to estimate whether the network possesses distinct communities. We have observed that permutations of the vertex order can change the partition into communities and if the network has amorphous community structure these partitions can significantly vary. We conducted an experiment for finding stable communities, that is, groups of vertices that are always grouped together over different permutations.

A ideal method for detecting these stable communities might be to search for densely connected sets of vertices, preferably large cliques. However members of cliques may not always fall in the same community. For example let us consider an example shown in Figure 5.1 In the given example vertices {2,3,4,5} form a clique. If we consider the following partition of six vertices ({1},{2,3,4,5},{6}). This partition gives negative

modularity of -0.06. Even though the vertices in the clique are tightly coupled or connected we get negative modularity. This is because each subgroup (2,3) and (4,5) has a strong connection to an external community. For example (2,3) has two edges to external vertex(1) and also two edges to internal vertex(4). Thus (2,3) has equal probability to combine with vertex(1), vertex (4) or with vertex(5). In general each subgroup within a stable community should have more internal connections than external connections. It is expensive to detect groups of vertices that satisfy this condition. We therefore relax the definition and detect communities where the number of internal connections is considerably greater than the external connections. Stable communities having external edges are fine as long as the pull from other communities is less. We assume stable communities are of at least size 2. Stable communities are composed of a core vertex, its distance 1 neighbors and neighbors of neighbors, i.e. vertices at distance 2 from the core vertex.

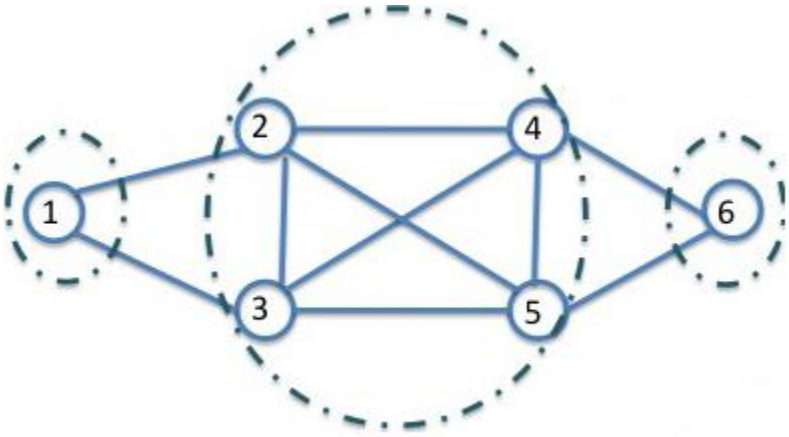

Figure 5.1. Partition of network into communities.

We detect stable communities by computing the fill-in [6] of the vertices as discussed in chapter 2. We consider only those vertices with low fill-in (generally 0 -2). We form a temporary community C composed of the vertex v and its neighbors. If the number of internal connections of each vertex in C is more than twice the number of external connections then C is designated as a stable community. Otherwise, we consider set N of the distance 2 neighbors of v, that are not elements of C. Edges in N can be classified as follows; (1) one endpoint connected to a vertex in community C (Case 1); (2) both endpoints connected to vertices in set N (Case 2) and (3) one endpoint connected to a vertex that is neither in C nor N (Case 3). A vertex in C is considered to be eligible for a stable cluster if that vertex has fewer edges of case 1 than case2 ;(Condition1) and fewer edges of case1 and case2 together than case3 ;(Condition2). Condition (1) guarantees that distance 2 neighbors do not have enough connections to vertices in a stable community. Condition 2 ensures that the set of external vertices has a larger pull from external communities other than C such that those sets don't exert much pull on vertices within C.

In general it is possible vertices can be assigned to multiple stable communities. If we discover that a vertex has been assigned to multiple communities we remove it. Algorithm5.1 provides pseudocode for our proposed stable community algorithm.

Algorithm 5.1 Detecting Stable Community in Networks

Input: - A Graph G= (V, E).

Output: - Stable Communities C_1, C_2,……C_n .

1. **procedure** Detecting Stable Communities
2. Set max-fill for Fill-In threshold
3. **for all** v ∈ V do
4. Compute Fill-In of v
5. **if** Fill-In of v<max_fill**then**
6. Create cluster C_v of v and its neighbors
7. In_Edge= Internal Edges of C_v
8. Ex_Edge= External Edges of C_v
9. **if** Ex_Edge<In_Edge /2 **then**
10. Associate cluster id v for each vertex in C_v
11. Mark C_v as stable community
12. **else**
13. Create set N of n // n is a distance 2 neighbor of core vertex v
14. Edgecase1= Edges with both end points in N
15. For all u ∈ C_v do
16. Edgecase2 = Edges with one endpoint in N and other in u
17. Edgecase3= Edges with one endpoint in N and not other not in u
18. **if** Edgecase2 < Edgecase3 AND (Edgecase1 + Edgecase2)<Edgecase 3 **then**
19. **if** Vertex u does not have cluster id **then**
20. Associate cluster id v with u
21. Mark u as a vertex in stable community.

The primary objective of our algorithm is to detect whether a network has community structure. Our algorithm will not detect any stable community if there exists no community structure in the network.

5.4 Modularity Maximization Using Stable Communities

Detecting stable communities can be used as a preprocessing step to improve the results of modularity maximization. The vertices with the same stable community id are assigned to the same community and then modularity maximization algorithm is applied to the transformed network. In this section we present the results of using this preprocessing technique combined with CNM and Louvain methods discussed in chpater 2. Our test network consists of unweighted and undirected networks obtained from DIMACS website[27]. Networks and their description are discussed in Table 5.1.

Network	Network Size	Network Description
Karate	(V=34, E=78)	Network of members in karate club.
Jazz	(V=198, E=2742)	Network of Jazz musicians
PolBooks	(V=105, E=441)	Network about USA politics
Celegans	(V=453, E=2025)	Metabolic network
Dolphin	(V=62, E=159)	Social network
Email	(V=1133, E=5451)	Network of e-mail interchanges
Power	(V=4941, E=6594)	Topology of power grid
PGP	(V=10680, E=24316)	Network of users of the Pretty-Good –privacy algorithm

Table 5.1: Network Description

Empirical Results. We applied permutations to each of the networks in the test suite. For each permutation we applied CNM and the Louvain method as well as the methods after detecting and combining stable communities. Some statistics for modularity obtained by the four methods are given in Tables 5.2 and 5.3.

Name	Modularity using CNM	Modularity using CNM+ stable community	Stable Community %
Karate	0.3938 (Avg) 0.4156(Max)	0.4022(Avg) 0.4197(Max)	29%
Jazz	0.43877(Avg) 0.4388(Max)	0.4234(Avg) 0.4442(Max)	26%
PolBooks	0.5019(Avg) 0.5019(Max)	0.5140(Avg) 0.5260(Max)	27%
Celegans	0.4046(Avg) 0.4149(Max)	0.4231 (Avg) 0.4327(Max)	30%
Dolphin	0.4802(Avg) 0.5094(Max)	0.4904 (Avg) 0.5242(Max)	22%
Email	0.4715 (Avg) 0.5201(Max)	0.4908(Avg) 0.5462(Max)	27%
Power	0.8997(Avg) 0.9221(Max)	0.9148(Avg) 0.9200(Max)	9%
PGP	0.8628(Avg) 0.8696(Max)	0.8616(Avg) 0.8716(Max)	40%

TABLE 5.2 :Comparision of Modularity values obtained by using CNM method and stable community preprocessing. Last column gives percentage of vertices in stable community.

Name	Modularity using Louvain	Modularity using Louvain+ stable community
Karate	0.4156(Avg) 0.4198(Max)	0.4170(Avg) 0.4198(Max)
Jazz	0.4427(Avg) 0.445(Max)	0.4435(Avg) 0.445(Max)
PolBooks	0.5258(Avg) 0.5268(Max)	0.5266(Avg) 0.5268(Max)
Celegans	0.4355(Avg) 0.4421(Max)	0.4320(Avg) 0.4447(Max)
Dolphin	0.5202(Avg)	0.5200(Avg)

	0.5233(Max)	0.5241(Max)
Email	0.5671(Avg) 0.5555(Max)	0.5664(Avg) 0.5745(Max)
Power	0.9360(Avg) 0.9365(Max)	0.9359(Avg) 0.9370(Max)
PGP	0.8776(Avg) 0.8807(Max)	0.8775(Avg) 0.8796(Max)

TABLE 5.3: Comparision of Modularity values obtained by using Louvain method and stable community preprocessing.

In general we observe that detecting stable communities as a pre processing step increases the final modularity value. However we observed there are a few exceptions such as the average for Jazz and maximum for power in CNM and average for Email and Celegans and max for PGP in Louvain. In general, improvement is higher for CNM than for the Louvain methods. In the CNM method once vertices are assigned to a community in a later step it doesn't have any back tracking feature to assign itself to a better community if discovered. However in the Louvain method if a vertex is assigned to a community and it is discovered at later stage of the algorithm that vertex may better fit in a different community, so the vertex is mapped to the most suitable community. This feature is called backtracking. From our results and observations we discover our preprocessing step woruld be more effective when the underlying algorithm doesn't contain a backtracking feature like CNM.

In Figure 5.2 and Figure 5.3 we plot the change in modularity over all the permutations of the Dolphin and the Power networks. In the dolphin network we can see using stable communities as a preprocessing step gives a significant boost to the CNM method. We also observe the Louvain method in general always produces high modularity. There exist certain cases where the CNM method along with preprocessing step is equivalent to the Louvain method. Dolphin network possesses good community structure. The values

in the Power network are well separated. Seperation of values by two algorithms indicate the power network does not have strong community structure.

In Table 5.4 we present the average time (in seconds) to compute individual methods , individual methods with preprocessing and time for the preprocessing step. Codes were compiled with GNU-g++ and experiments were performed on dual-core processor with 2.7 GHZ speed and 32 GB RAM. In some cases we observed the preprocessing step reduces the overall agglomeration time, however detecting stable communities is generally expensive.

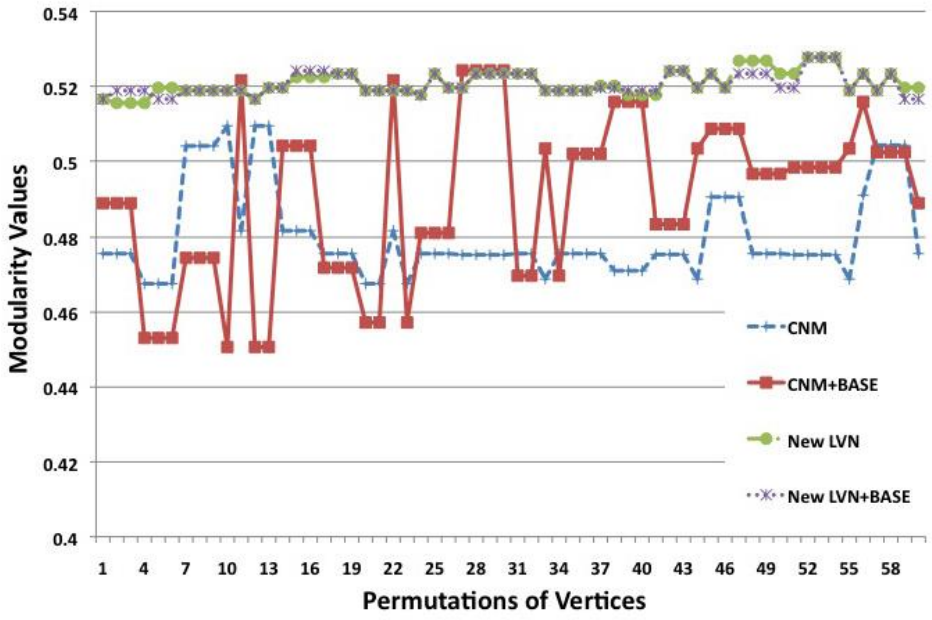

Figure 5.2: Modularity Values for the Dolphin Network

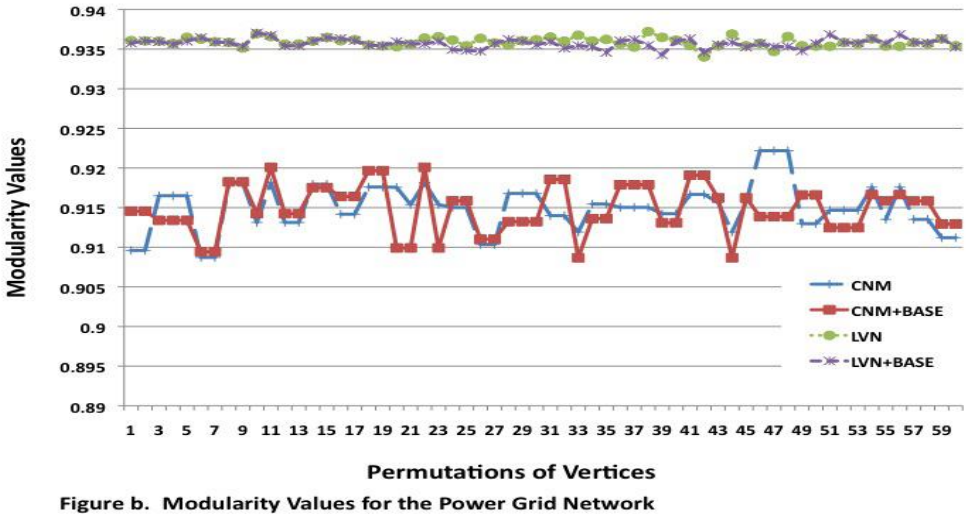

Figure b. Modularity Values for the Power Grid Network

Figure 5.3: Modularity Values for the Power Network

Name	CNM	CNM+Preprocessig	LVN	LVN+Preprocessing	Preprocessing
Jazz	1.50	1.51	0.57	0.68	0.45
Polbooks	0.085	0.067	0.06	0.05	0.04
Celegans	3.67	1.80	1.35	1.50	0.86
Dolphins	0.01	0.018	0.003	0.005	8e-04
Email	32.31	18.6	11.84	10.31	3.15
Power	52.59	50.19	24.12	24.68	31.4
PGP	760.78	757.25	579.88	577.87	25.79

Table 5.4: Comparison of Execution Time (In Seconds) of both methods and time to detect stable community.

5.5 Shared Memory Algorithm for Parallelizing the Stable Community Detection method

In this section we present our parallel implementation of the stable community algorithm to detect stable communities. We consider regions with loops as they are the most natural part to exhibit parallelism. We have parallelised line 4 that is computing the fill-in for each vertex v. We divide the vertices across threads and each thread computes the fill-in for each vertex mapped to its thread id. Once all threads are executed we combine the fill-in values for each vertex and based on the threshold of fill-in, the cliques are formed. The remaning portion of the code is sequential as discussed in section 5.3. We tested scalability on larger networks obtained from creatine and untreated mice and breast cancer networks. In Table 5.5 we list the node and edge counts for the networks . We conducted experiments using an opteron multicore processor with 64 cores per node and 256GB Ram per node. We used shared memory OpenMp and tested the scalability of the algorithm by execution over 1 to 64 threads. Figure 5.4 demonstrates our algorithms shows good scalability.

Network	Node	Edge
Untreated	45020	655698
Creatine	45023	714628
Familal	48803	687783
Non	48803	1109553

Table5.5: Node and Edge counts for networks.

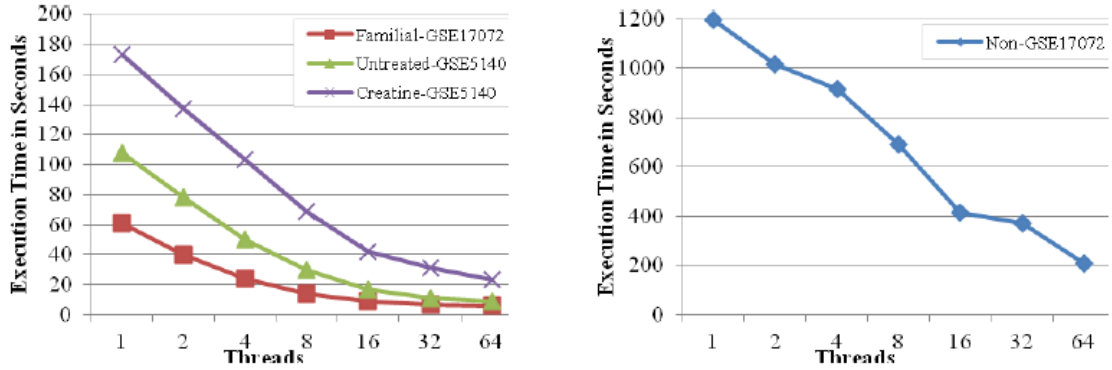

Figure 5.4: Strong Scalability for the parallel implementation of stable community algorithm.

5.6 Discussion

In this chapter we have attempted to design and develop an algorithm to detect stable communities in a network. We detect stable communities as a preprocessing step and use those stable communities in well known algorithms like CNM and Louvain to detect communities.The percentage of stable communities in the initial step can give a rough indication of how modular a network is. In general we conclude if the percentage of vertices within stable communities is high, detecting communities in such networks will be of practical value else detecting communities will be only of academic interest.

Chapter 6

Detecting Communities using Relative Permanence as a Metric

6.1 Introduction

Modularity isa widely accepted metric for detecting and estimating thequality of community structure as discussed in chapter2. However many researchers have begun to discover the demerits or limitation of the maximizing modularity approach for community detection. Various limitations include the resolution limit, the degeneracy of solutions and asymptotic growth of modularity value. There still exist fundamental questions which arenot answered – does a network possess community structure? Or would the partition be accurate. In this chapter we answer those questions by proposing a novel metric called permanence which is built on pull experienced by a vertex from neighbors that is mapped to adifferent community. We show that our new metric when compared to modularity and conductance is a better optimization parameter for detecting communities on synthetic networks and real-world networks. We also demonstrate permanence is more sensitive to different perturbations applied to community structures. The rest of the chapter is arranged as follows. In section 6.2 we present network datasets and ground truth communities. In section 6.3 we discuss permanence, community detection algorithms and evaluate the community scoring function. In section 6.4 we present our new community detection algorithm named **Max_Permanence** based on maximizing permanence, we study the performance of our proposed algorithm. In section

6.5 we discuss how permanence resolves issues related with modularity maximization and finally conclude with dicussion and results.

6.2 Related Research, Network Datasets and Ground Truth Communities

Fortunato and Barthelemy[15] presented a resolution limit problem of modularity, which states that optimizing modularity will fail to detect communities smaller than a threshold size or weight[16]. Good et al.[17] presented another issue of modularity called degeneracy of solutions which states that for a single graph we can get adiffferent community structure for exponential number of high modularity . They also studied limiting thebehaviour of modularity foran infintely modular network and show that it strongly depends on both thesize of the network and thenumber of module**s** it contains. Lancichinetti and Fortunato[18] presented that the multi resolution version of modularity is not only inclined to merge small communities but also to split large well defined communities.

We provide a description of the networks used for our experiments. We used theLFR benchmark model[19] to generate artifical networks with a well defined community structure.The LFR benchmark model has been discussed in chapter 4 and 5. In this chapter for our experiments we have usedthe following LFR benchmark parameters. We set thenumber of nodes (n) as 1000 and μ is varied between 0.1 to 0.6. We used three large real-world networks whose well defined community structure are available. Network properties are discussed in Table 6.1.

Networks	N	E	<K>	K_{max}	C	n_c^{min}	n_c^{max}
Football	115	613	10.57	12	12	5	13
Railway	301	1224	6.36	48	21	1	46
Coauthorship	103677	352183	5.53	1230	24	34	14404

Table 6.1: Real world network properties where N and e are number of nodes and number of edges , C is the number of communities, <K> and K_{max} average and maximum degree, n_c^{min} n_c^{max} size of smallest and largest communities.

Football network as discussed in chapter 4 contains the network of American football games between Division IA colleges during regular season Fall 2000. Indian Railway proposed by Ghosh et.al[20] consists of nodes representing stations and two stations connected by an edge if there exist at least one train –route such that both stations are scheduled stop or hault on that route. In case of the weighted version the weight of an edge will be thenumber of train –routes on which both station are scheduled halts. We mark each station with region (state in India). States act as communities because the number of trains within each state is higher than the number of trains between two stations.

A coauthorship network is developed by Chakraborty et al.[21] from the citation dataset. The dataset contains information of all the papers of computer science published between 1960 to 2009 archived in DBLP. From this dataset we build an undirected coauthorship network where each node represents an author and an edge is drawn if two authors collaborate at least once via publishing a paper. Each paper is categorised by its related field. We map this field as the research area of the authors writing that paper. Author may be mapped to more than one research area of interest. We resolve this issue

by mapping author to the major field of interest in which they have written most of their papers.We consider research area or major field as the ground truth communities since author have tendency to cite papers belonging to same area.

6.3 Permanence

In this section we develop the formula for permanence based on the following two assumptions : (1)a vertex should have more internal connections than the number of connections to any of the neighboring communties. (ii) The substructure of a community's internal neighbors of vertex should be highly connected among each other. In general both assumptions guarantee the number of internal connections is larger than the number of connections to any one single external community. The permanence of a vertex v is given below : $Perm(v) = [\frac{I(v)}{D(V)} \times \frac{1}{E_{max}(v)}] - (1 - c_{in}(v))]$

Where I(v) is the number of internal neighbors of v, D(v) is the degree of v, $E_{max}(v)$ number of connections of v to that external community (maximum) neighbors of v, and $c_{in}(v)$ is the clustering coefficient of v . We use few toy example Figure 6.1 to measure permanence of vertex v.

According to Figure 6.1 for vertex v I(v) =4, $E_{max}(v)$= 2 and c_{in} (v)= 5/6. Using permanence formula we substitute the value of I(v) , $E_{max}(v)$ and c_{in} (v) we get perm(v)= 0.12. If vertices do not have any external connections permanence of vertex v is set to its internal clustering coefficient. Perm(v) is set to 0 if vertices in communities is less than 3 entries. If the vertex is a part of clique then Perm(v) obtain its maximum value 1. For every vertex v, -1<Perm(v) ≤1 . Overall permanence of a graph G(V,E) is given by

Perm(G)= 1/v $\sum_{v \in V}$(Perm(v)).

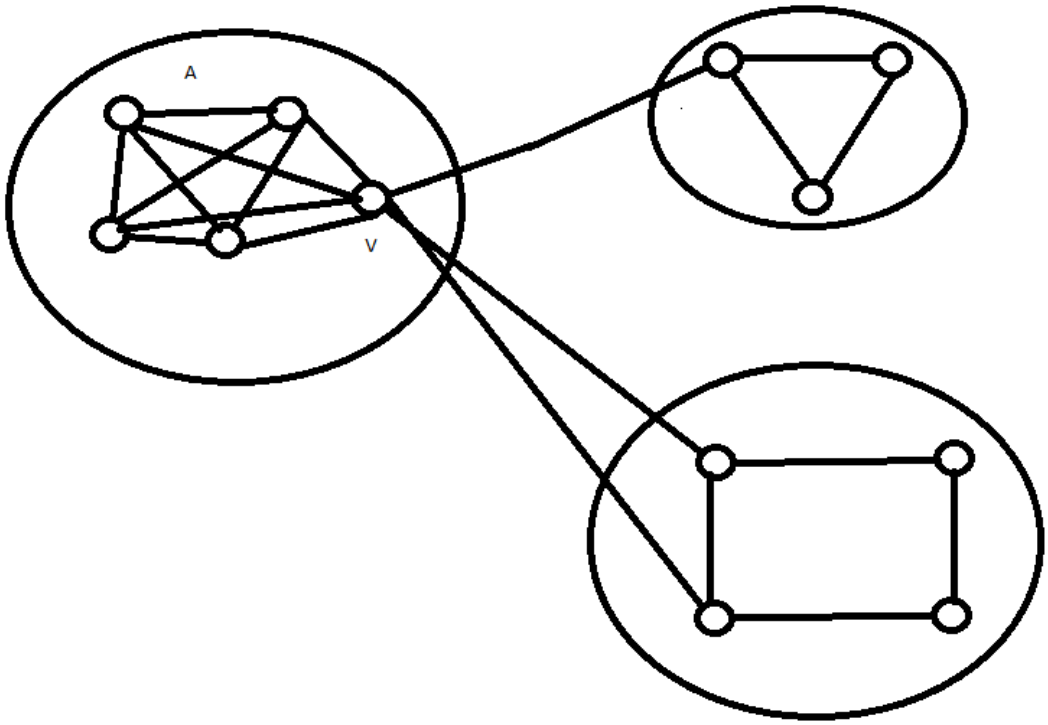

Figure 6.1. Toy example to measure permanence of vertex v.

We perform an experiment to determine whether permanence is a good community scoring function by comparing it with other scoring functions like modularity, conductance and cut ratio.We run several community detection algorithms on the graph and obtain a community set pertaining to each algorithm. We compute different community scoring functions and rank each algorithms based on the value of metric. We also compare the community set detected using a different validation measure such as NMI and purity as discussed in chapter 2.

There exist various community detection algorithms, we have categorized the set of algorithms based on the principle they use to detect communities.

(i) **Modularity based approaches:** Modularity based approaches are discussed in chapter 2 we use CNM and Louvain algorithm for our experiments.

(ii) **Node similarity approaches**: In this category community is determined as group of nodes which are similar to each other and dissimilar from rest of the network. For our experiments in this category we select Walk Trap[27] algorithm.

(iii) **Compression-based approaches:** In this approach community structure is the set of nodes represented in the adjacency matrix which has maximizing compactness while information loss is minimum. Popular algorithms are InfoMod[27] and InfoMap[27].

(iv) **Significance- based approaches:** Community structure can be expected under certain circumstaces, however group of densely connected nodes can appear by chance.

(v) **Diffusion-based approaches:** In this approach assumption is that information is more efficiently exchanged between nodes of the same community. In community Overlap Propagation Algorithm (COPRA)[27] information takes the form of a label, and the propagation mechanism relies on a vote between neighbors. Group of nodes with same label form communities.

6.3.1 Evaluating Community Scoring Functions and Ground-Truth Comparison Metrics

We run each algorithm discussed in section 6.3 on all datasets mentioned in section 6.2. We computed modularity, permanence,conductance and cut-rato and ranked algorithms based on each of the community scoring functions separately. In Figure 6.2 we present score and rank (in parenthesis) for the football network. We use three standard validation metrics:- Normalized Mutual Information (NMI)[27], Adjusted Rand Index (ARI)[27] and Purity (PU)[27] to measure the accuracy of detected communities with respect to ground-truth. These measures are not relevant in the context of network analysis. Modified versions of NMI, ARI and Purity are Weighted-NMI (W-NMI), Weighted-ARI(W-ARI) and Weighted-Purity (W-PU) respectively. We performed experiments using all six measures to validate the results. We performed the same experiments on LFR and real-world datasets. We compare the ranks obtained from community score functions with ranks obtained from validation measures.We assume that the rank of the best community scoring function should match the rank produced by the validation measures.

Algorithms	Mod	Perm	1-Con	1-Cut	NMI	ARI	PU	Avg (N)	W-NMI	W-ARI	W-PU	Avg (W)
Louvain	0.60(1)	0.36(1)	0.77(5)	0.44(7)	0.93(1)	0.99(1)	0.89(3)	1.67	0.99(2)	0.93(2)	0.99(1)	1.67
FastGreedy	0.58(2)	0.25(3)	0.81(3)	0.59(4)	0.93(1)	0.99(1)	0.91(1)	1.00	1.00(1)	0.94(1)	0.99(1)	1.00
CNM	0.55(3)	0.20(4)	0.85(1)	0.86(1)	0.67(5)	0.75(4)	0.42(7)	5.33	0.55(5)	0.63(5)	0.71(3)	4.33
WalkTrap	0.60(1)	0.36(1)	0.82(2)	0.69(2)	0.90(3)	0.98(2)	0.84(5)	3.33	0.98(3)	0.91(3)	0.99(1)	2.33
Infomod	0.60(1)	0.35(2)	0.82(2)	0.69(2)	0.89(4)	0.97(3)	0.82(6)	4.33	0.97(4)	0.89(4)	0.98(2)	3.33
Infomap	0.60(1)	0.35(2)	0.79(4)	0.51(6)	0.89(4)	0.97(3)	0.82(6)	4.33	0.97(4)	0.89(4)	0.98(2)	3.33
COPRA	0.60(1)	0.36(1)	0.82(2)	0.66(3)	0.92(2)	0.99(1)	0.90(2)	1.67	0.99(2)	0.93(2)	0.99(1)	1.67
OSLOM	0.60(1)	0.36(1)	0.50(6)	0.57(5)	0.90(3)	0.98(2)	0.85(4)	3.00	0.98(3)	0.91(3)	0.99(1)	2.33

Figure 6.2: We compute the values of four community scoring functions on output obtained from eight different algorithms and validation measures using ground –truth communities.

In Table 6.2 we present correlations of these community scoring functions across all the validation measures for each of the networks.

Networks	Modularity	Permanence	Conductance	Cut
LFR(μ=0.1)	0.88	0.88	0.88	0.02
LFR(μ=0.3)	0.61	0.74	0.72	0.28
LFR(μ=0.6)	0.87	0.96	-0.18	-0.44
Football	0.25	0.43	-0.29	-0.41
Railway	0.43	0.46	0.08	-0.48
Coauthorship	0.92	0.92	0.76	0.86

Table 6.2: Performance of community scoring function averaged overall validation measures for each network.

6.4.1 Community Detection Based On Permanence

We develop a community detection algorithm by maximizing permanence. Our algorithm Max_Permanence is motivated by the Louvain method[8] for modularity maximization. The pseudo code is presented in Algorithm 6.1.

6.4.1.1 Algorithm Overview

Each vertex in the network is initialized to a singelton community and their permanence is set to 0. For each vertex we test whether combining the vertex to a

neighboring community will increase its permanence. If permanence is found to be increased we join vertex and its appropriate vertex neighboring community. The process is repeated for each vertex and the entire location of all vertices is repeated over several iterations until the permanence value remains constant or converges. Our proposed algorithm always tries to maximize permanence. Our apporach is to move vertices to a community that preserves community structure. If such a move is not possible then the vertex **remains in a singleton** community or moves to another communit**y** where it is tightly coupled to its neighbors.

6.4.2 Performance Evaluation

In table 6.3 we present the average improvement of our algorithm over others for each validation metric. In general we discovered on average communities obtained by maximizing permanence matches known ground truth communities quite well for allmost all networks except LFR (μ=0.6).

As we discussed the permanence metric works good if the network has modular structure. If the network isn't modular enough the permanence value tends to degrade indicating that detecting communities in such networks is just of academic interest.

For the railway network our algorithm detects three singelton communities. Even the ground truth community structure for the railway network contains one of these singelton communities. Among all the algorithms discussed only our algorithm captures those singelton communities. We summarize that if a network is really modular or has good community structure like (LFR μ=0.1) ,maximizing permanence efficiently captures realistic modules.

Input : A graph G.

Ouput :- Permanence of G and community set**.**

1.Procedure Max permanence (G(V,E))

2.Each vertex assigned to a singelton community.

3.Set value of maximum iterations as maxIt

4. Sum=0

5.Old_sum= -1

6. Itern=0

7.While sum !=old_sum and Itern < maxIt do

8. Itern = itern +1

9. Old_sum=sum

10. Sum=0

11. **for all** v ∈ V do

12.(compute current permanence of v)

13.Cur_perm=perm(v)

14. if cur_perm==1 then continue

15.N is set of neighboring communities of v

16.**for all** n ∈ N do

17. Move v to community n

18.(Compute permanence of v in community n)

19.n_perm=Perm(v)

20.**if** cur_perm < n_perm then

21. cur_perm=n_perm

22. **else**

23. retain v in its original community

24.sum=sum+cur_perm

Algorithm 1 Max_Permanence

We also observed if intercommunity edge density starts to increase our algorithm's performs better to capture communities within a certain limit like(LFR μ=0.3) after which it starts deteriorating as the network doesn't have good community structure or is less modular.

Validation Metrics	LFR(μ=0.1)	LFR(μ=0.3)	LFR(μ=0.6)	Football	Railway	Coauthorship
NMI	0.04	0.15	-0.31	0.04	0.15	0.04
ARI	0.06	0.21	-0.39	0.07	0.03	0.03
PU	0.04	0.17	-0.38	0.01	0.13	0.03
W-NMI	0.02	0.14	-0.41	0.09	0.26	0.05
W-ARI	0.05	0.19	-0.35	0.05	0.02	0.04
W-PU	0.03	0.17	-0.45	0.00	0.05	0.02

Table 6.3: Average improvement of our algorithm over different algorithms for each network in terms of different validation measures.

6.5 Permanence Resolving Issues Related with Modularity Maximization

We have seen and discussed in previous chapters that modularity suffers from (a) resolution limit, (b) degeneracy of solutions (c)dependency on the size of the graph. In this section we present how each of these problems are resolved by maximizing permanence.

We use a simple example of two communities A and B connected by one vertex v (as shown in Figure 6.5). In this example the community mapping is primarily determined by v and its neighbors. We also assume apart from the edges through v, there is no connection between communities between A and B. Figure 6 shows four possible ways of assignment of v into communities. These are as follows: **Case 1:** v joins community A; **Case 2:** v joins community B; **Case 3:** community A,B and vertex v merge together; **Case 4:** communities A,B and v remain as three separate communities.

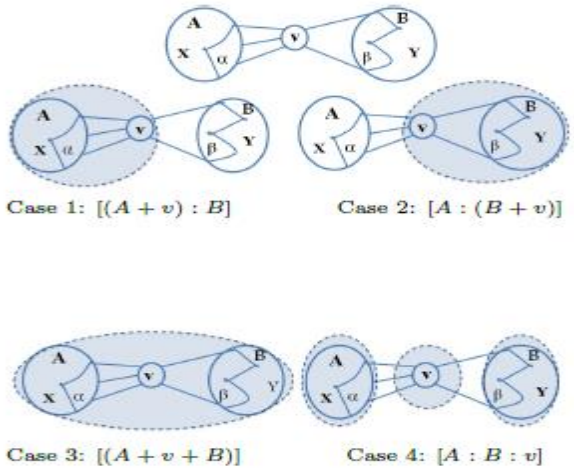

Figure 6.3: Toy examples demonstrating four cases.

6.5.1 Terminology

We assume vertex v as shown in Figure 6 is connected to α(β) nodes in communityA(B), and these (α(β)) nodes from the set N_α (N_β). The total number of vertices in community A is x+ α , and the total number of vertices in community B is y+ β. Before v is added the average internal degree for community A and community B is I_A and I_B respectively. The average internal clustering coefficient of neighbouring nodes in communities A andB be C_A and C_B. If v is added to communities A(B) then average internal clusteringcoefficient of v becomes $C_A^v(C_B^v)$. The average clustering coefficient of nodes in $N_\alpha(N_\beta)$becomes $C^\alpha(C^\beta)$.

We also assume communities A and B are tightly connected internally such that both communities have greater C_A and C_B. $C^\alpha(C^\beta)$ values dependent on connections of v to communities and connections of vertices in N_α (N_β).We assume neighbors of v are not

connected with each other, then the average clustering coefficient will decrease. If v does not add any new edges to group of neighbors then

$C^{\alpha} = C_A \times \frac{I_A - 1}{I_A + 1}$ and $C^{\beta} = C_B \times \frac{I_B - 1}{I_B + 1}$

6.5.2 Discussion on Issues in Modularity Maximization

In this section we show how permanence overcomes issues of modularity maximization.

Degeneracy of solution :- In figure 6.5 if we consider α=β then the community scoring function such as modularity will have multiple distinct high scoring solutions and will lack global maximum. We encounter a tie-breaking situation [23]. Modularity maximization will assign vertex v arbitrarily to A or B. In our algorithm or our metric permanence will assign v as a individual community as long as it maintains conditions as discussed below.

Condition 1. If α=β ,$C^{\beta} = C_B \times \frac{I_B - 1}{I_B + 1}$ communities A,B and v will remain separate rather than v joining A if $\alpha \times \left(\frac{2C_A - 1}{I_A + 1}\right) + (1 - C_A^v) \geq \frac{1}{2\alpha}$ if α=β=1 then C_A^v=0 then communities will remain separate. As α increases the left hand side of the equation will remain larger than the right which guarantees they remain separatecommunities. We experimentedour metric with a 5×5 complete grid we observed permanence generates one solution by assigning eachvertex into a separate singeltoncommunity; whereas modularity provides multiple solutions by combining two or more vertices. Vertex v will remain in the same community if vertex v is loosely connected to its neighboring community and has an equal number of connections to each community. However permanence doesn't provide the complete solution to **Degeneracy of solution.** In a few cases we get high permanence if vertex v is combined with community A or community B.

Resolution limit:- Communities of certain small size **fail to be detected** as they are merged to larger communities. We have witnessed the classic examples where the modularity metric fails to detect communities of small size in a cycle of m cliques since the maximum modularity is obtained by merging two neighboring cliques. If we use permanence as a metric we can determine four cases discussed above. We explore the condition to determine whether v will join community A rather then being separate (similarly we can do analysis if v joins community B).

Condition 2. Joining v to community A gives higher permanence rather than merging thecommunities A, B and v if ; $C^{\beta}=C_B\frac{I_B-1}{I_B+1}$, and $\frac{\gamma}{(\gamma+1)\beta}+\frac{C_A^v(2\gamma+1)-C_B^v}{(\gamma+1)^2}+\frac{\beta(2C_B-1)}{I_B+1}>1$where γ= α/β and also if ; $C^{\beta}=C_B(\frac{I_B-1}{I_B+1})$, and $\frac{\gamma}{(\gamma+1)\beta}+\frac{C_A^v(2\gamma+1)-C_B^v}{(\gamma+1)^2}+\frac{\beta(2C_B-1)}{I_B+1}>1$.

If we consider the clique example as a special case where v is connected by one edge to community B and is connected all nodes in community A. We observe β=1 and adding v decreases internal clustering coefficient of B. In general in a network if a node has lees than two neighbors we set permanence as zero. As A is a clique so $C_A^v=1$ and $C_B^v=0$. After subsituting all the values in condition 2 we observe we get a higher permanence when we combine vertex v with community A and neighboring communities shouldn't be merged. Our observation is independent of the size of cliques. This phenomena highlights that if v is tightly connected to a community and very loosely connected to another community ; highest permanence is obtained by combining vertex v with community to which v is more connected.

6.5.3 Discussion

In this section we proposed a new metric called permanence which overcomes issues or shortcomings of modularity. We have demonstrated with analytical proofs with experiments on synthetic and real-world networks that permanence is effective community evaluation metric compared modularity.

Chapter 7

Conclusion And Future Work

In my thesis we have proposed a parallel template for the Louvain method for modularity maximization. Our results show our proposed template is scalable, and produces modularity equivalent to those expected from the sequential case. In the future we plan to apply or further improve our template on dynamic networks. We have presented the effects of vertex perturbation on community structure and discussedthe existence of stable communities. We have also shown if a network has a good community structure then using stable communities as a preprocessing step to the Louvain or CNM algorithm wecan get an improvement in the modularity value. Our algorithm to detect stable communities has room for improvement. We consider only distance-1 neighbors as stable communities. We have to include vertices at longer distances to create a stronger stable community. Our proposed algorithm has a tendency to pick up some false positives if vertices have two nearby consensus communities that are tightly connected. In future we have to improve the conditions on stable communities to reduce false positives. In the final leg of my thesis we have discussed the limitation of modularity maximization and proposed a new metric called permanence which is able to reduce many of the shortcomings of modularity. We have also shown our proposed metric is effective when compared to other metrics on real –world and synthetic networks. Our proposed metric calls for more deeper levels of investigation. We have to test our metric on more diverse areas to prove the robustness. In my thesis we have restricted our

discussion on non overlapping communities. In the future we plan to further extend the permanence metric to evaluate overlapping community structure.

References

[1] S. Bansal, S. Bhowmick and P. Paymal. Fast Community Detection ForDynamic Complex Networks, Communications in Computer and Information Science Volume 116. Proceedings of the Second Workshop on Complex Networks (2010).

[2] V.D. Blondel, J.-L.Guillaume, R. Lambiotte and E. Lefebvre. Fast unfolding of community hierarchies in large networks. J. Stat. Mech. 2008 (10).

[3] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity clustering. IEEE Transactions on Knowledge and Data Engineering, 20(2):172188, (2008).

[4] Clauset, A., Newman, M.E.J. and Moore, C. Finding community structure in very large networks. Phys. Rev. E. 70(6), 66111 (2004).

[5] B. H. Good, Y.-A.deMontjoye and A. Clauset The performance of modularity maximization in practical contexts. Phys.ical Review. E, 81, 046106 (2010).

[6] D. Ediger, J. Riedy, H. Meyerhenke, and D.A. Bader. Tracking Structure of Streaming Social Networks, 5th Workshop on Multithreaded Architectures and Applications (MTAAP), (2011).

[7] A. Lancichinetti A and S. Fortunato Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. Phys Rev E 80: 016118 (2009).

[8] M.E.J. Newman, M. Girvan. Finding and evaluating community structure in networks.Phys. Rev. E 69(2), 026113 (2004).

[9] M. A. Porter, J.-P. Onnela, and P. J. Mucha. Communities in networks. Notices

of the American Mathematical Society. 56, (2009).

[10] W. Rand, Objective criteria for the evaluation of clustering methods. J. Am. Stat. Assoc. 66 (336), 846850 (1971).

[11] U.N. Raghavan, R. Albert, R. and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76, 036106. (2007).

[12] J. Soman and A. Narang. Fast Community Detection Algorithm with GPUs and Multicore Architectures. Proceedings of the 2011 IEEE International Parallel& Distributed Processing Symposium. (2011).

[13] E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader. Parallel community detection for massive graphs. In 10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering. (2012).

[14] V.A. Traag, P. Van Dooren, Y. Nesterov. Narrow scope for resolution-limit-free Community detection. Phys. Rev. E 84, 016114 (2011).

[15] S. Fortunato and M. Barthelemy. Resolution limit incommunity detection. PNAS, Jan. 2007.

[16] J. W. Berry, B. Hendrickson, R. A. LaViolette, and C. A. Phillips. Tolerating the community detection resolution limit with edge weighting. Physical ReviewE, 83(5):056119, May 2011.

[17] B. Good, Y. D. Montjoye, and A. Clauset. Performance of modularity maximization in practical contexts. Phys. Rev. E, 81(4):046106, 2010.

[18] A. Lancichinetti and S. Fortunato. Consensus clustering in complex networks. Scienti_c Reports, 2,2012.

[19] A. Lancichinetti and S. Fortunato. Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities.
Phys. Rev. E, 80(1):016118, July 2009.

[20] S. Ghosh, A. Banerjee, N. Sharma, S. Agarwal, and N. Ganguly. Statistical analysis of the indian railway network: a complex network approach. Acta Physica Polonica B Proceedings Supplement, 4:123{137, March2011.

[21] T. Chakraborty, S. Sikdar, V. Tammana, N. Ganguly,and A. Mukherjee. Computer science _elds as ground-truth communities: Their impact, rise and fall.
In ASONAM, pages 426 { 433, 2013.

[22] J. Yang and J. Leskovec. De_ning and evaluatingnetwork communities based on ground-truth. In Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics, MDS '12, pages 3:1{3:8, New York, NY, USA, 2012. ACM.

[23] B. Good, Y. D. Montjoye, and A. Clauset.Performance of modularity maximization in practicalcontexts. Phys. Rev. E, 81(4):046106, 2010.

[24] M. Newamn, *"Scientific collaboration networks: Ii. shortest paths, weighted networks, and centrality"*, Phys. Rev. E, vol. 016132, 2001.

[25] J. L. Gross and J. Yellen, *"Handbook of Graph Theory and Applications"*. CRC Press, 2004.

[26] T. Chakraborty, S. Srinivasan, N. Ganguly, S. Bhowmick, and A. Mukherjee. ConstantCommunities in Complex Networks. ScientificReports, 3, May 2013.

[27] T. Chakraborty, S. Srinivasan, N. Ganguly, S. Bhowmick, and A. Mukherjee. Stay where you belong : on the permanence of vertices in network communities WWW 2014(Submitted).