COMMUNITY DETECTION IN EVOLVING NETWORKS

TEJAS PURANIK

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fullfillment of The Requirements

For the Degree of Master of Computer Science

Concordia University

Montréal, Québec, Canada

March 2017

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Tejas Puranik**

Entitled: **Community Detection in Evolving Networks**

and submitted in partial fulfillment of the requirement for the degree of

Master of Computer Science

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

______________________________ Chair

Dr. M. Kersten-Oertel

______________________________ Examiner

Dr. J. Opatrny

______________________________ Examiner

Dr. T. Fevens

______________________________ Supervisor

Dr. L. Narayanan

Approved by ______________________________

Chair of Department or Graduate Program Director

__________ 20 ____ ______________________________

Dr.Amir Asif, PhD, PEng

Dean,Faculty of Engineering and Computer Science

Abstract

Community Detection in Evolving Networks

Tejas Puranik

Most social networks are characterized by the presence of *community structure*, *viz.* the existence of clusters of nodes with a much higher proportion of links within the clusters than between the clusters. Community detection has many applications in many kinds of networks, including social networks and biological networks. Many different approaches have been proposed to solve the problem. An approach that has been shown to scale well to large networks is the *Louvain* method, based on maximizing *modularity*, which is a quality function of a partition of the nodes.

In this thesis, we address the problem of community detection in evolving social networks. As social networks evolve, the community structure of the network can change. How can the community structure be updated in an efficient way? How often should community structure be updated? In this thesis, we give two methods based on the Louvain algorithm, to determine when to update the community structure. The first method, called the Edge-Distribution-Analysis algorithm, analyzes the newly added edges in order to make this decision. The second method, called the Modularity-Change-Rate algorithm, finds the rate of modularity change in a given network, and uses it to predict whether or not an update is required.

Due to the sparsity of real datasets of evolving networks, we propose three models to generate evolving networks: a Random model, a model based on the well-known phenomenon of homophily in social networks, and another based on the phenomenon of triadic and cyclic closure. Starting with real-world data sets, we used these models to generate evolving networks. We evaluated the Edge-Distribution-Analysis algorithm and Modularity-Change-Rate algorithm on these data sets. Our results show that both our methods predict quite well when the community structure should be updated. They result in significant computational savings compared to approaches that would update the community structure after a fixed number of edge additions, while ensuring that the quality of the community structure is comparable.

Acknowledgements

It is my pleasure to express my sincere gratitude to my supervisor Dr. Lata Naraynan for her continuous support throughout my studies and research work. I thank her for her patience and efforts to improve my thesis. I could not have imagined having a better supervisor.

Besides my supervisor, I am grateful to Dr. Gregory Butler for his support and confidence in me. I would also like to thank Dr. Jaroslav Opatrny for partially supporting my studies financially. In addition, I would like to thank the rest of my thesis committee.

Finally, I wish to express my heartfelt gratitude to my parents for their continuous motivation and unconditional love. A very special thanks to my dear brother Akash Puranik who has always assisted me. I would also like to thank my friends Gurpreet Bhinder, Brice Muco, and Akhil Jobby for their encouragement.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

SLM	**S**mart **L**ocal **M**oving
DSLM	**D**ynamic **S**mart **L**ocal **M**oving
LHM	**L**ocal **M**oving **H**euristic
EDA	**E**dge **D**istribution **A**nalysis
MCR	**M**odularity **C**hange **R**ate

Chapter 1

Introduction

Graphs have become remarkably useful to represent a wide range of systems of interest to scientists, engineers, and social scientists. Some examples are biological networks, telecommunication networks, Internet, world wide web, social networks, citation networks and collaboration networks. Social network analysis started in 1930 [19] and has become an important area of research in computer science. A social network is defined as a set of nodes connected by edges. Nodes could represent people, molecules, computers or routers and edges represent the connections between nodes. For example, in an email network, nodes represent a set of people in an organization, and an edge between nodes a and y indicates that there was an email exchange between x and y in a given time period. Advances in technology have made possible the collection of data on social networks that contain millions of nodes and billions of edges. The ever growing nature of such networks demands new methods to understand the hidden information from their structure.

One promising approach consists of decomposing the networks into sub-units or *communities*, which are sets of highly inter-connected nodes [9], [20]. The problem of community detection has been studied extensively for static networks. However, most social networks are not static in nature. Many social networks evolve rapidly in terms of size over time. Nodes may join or leave social networks. Even the nodes that stay, may lose connections or create new connections. In popular online social networks like Twitter, Facebook, Livejournal, within 24 hours, millions of users update their connections. Such networks may be termed *dynamic* social networks. For large networks with millions of nodes and billions of edges, some thousands of edge additions or deletions might seem insignificant, but over time they change the structure of the network. In particular, communities in the network may evolve and change as a result of edge additions or deletions. This change in community structure raises the need of re-identification of communities in the network [3]. In this thesis, we study the problem of community (re)-identification in dynamic networks. We call this the problem of dynamic community detection.

1.1 Community Detection

How do we define communities in a social network? A generally accepted, though somewhat vague, definition is that a communities are groups of nodes that have denser connections within the group than outside the group. We discuss different defintions of a community in the literature in Chapter 2. A classic example of social network that has been studied in this context is Zachary's karate club network [52]. This network contains 34 nodes and 78 edges. The nodes of this network represent the members of karate club and the edges connect the people who had interactions outside the club. This network was observed for a period of three years. During this period, the president of the club and the instructor had a conflict which led to the breakup of the club into two separate groups, supporting the instructor and the president. A key question of interest is: Would it have been possible to predict this separation by simply examining the structure of the network? From Figure 1, we can easily distinguish two groups of nodes that have a lot of connections within the groups and few between the two groups. The aim of community detection algorithms is to identify such communities.

For large networks, efficient algorithms for community detection become very important as it is impossible to visualize networks of millions of nodes. Community detection has a lot of applications in real networks. An important application is for recommendation systems. It makes sense to make similar recommendations to a group of people who are in the same community and possess similar interests [45]. Another example is in research, the communities obtained from citation and co-authorship networks like DBLP and Condmat can be used to develop new methods and theories and can help to understand research patterns. In [17], the authors have tried to detect criminal activities in criminal networks using clustering algorithms. Traditional methods focus on the movement of an individual but here the authors claim that one should use communities to detect these events. Community detection is also used for refactoring software packages in complex software networks [37] [1] There are a lot of applications in biological networks, but one of the most significant is a community-based lung cancer detection approach which focuses on high risk patients [7]. In a protein-protein interaction (PPI) networks, communities correspond to functional groups, i.e to proteins having the same

[1]Refactoring code means restructuring the code without changing its external behavior. Such refactoring improves code readability and reduces complexity.

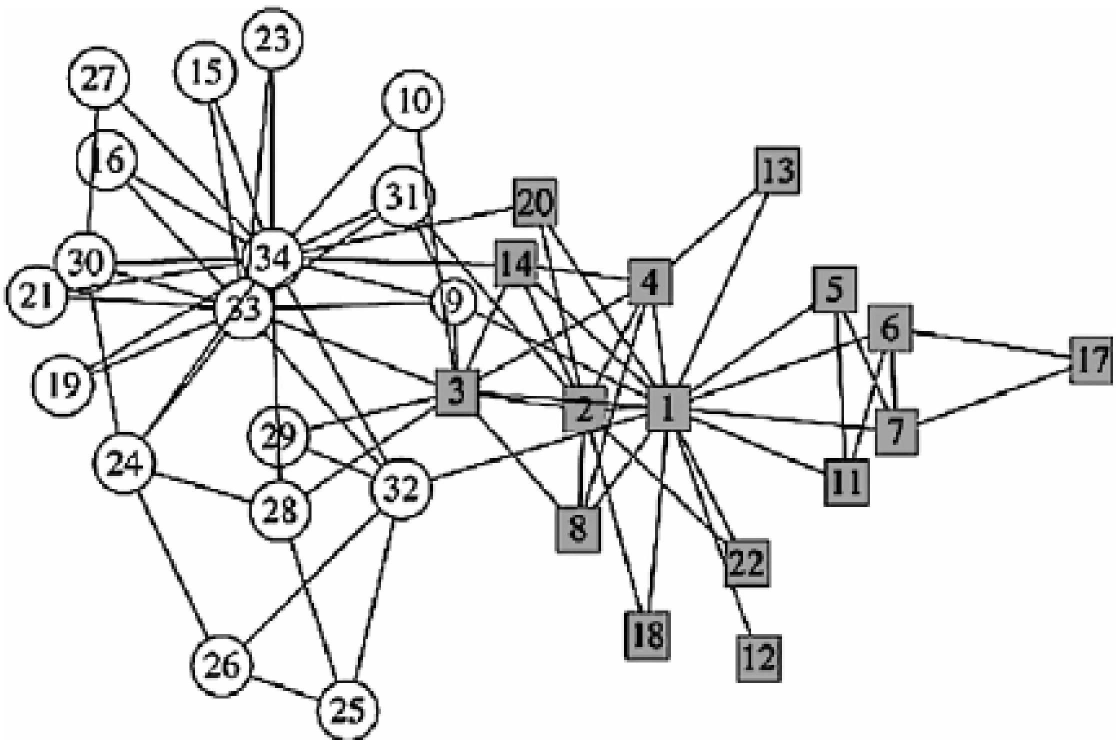


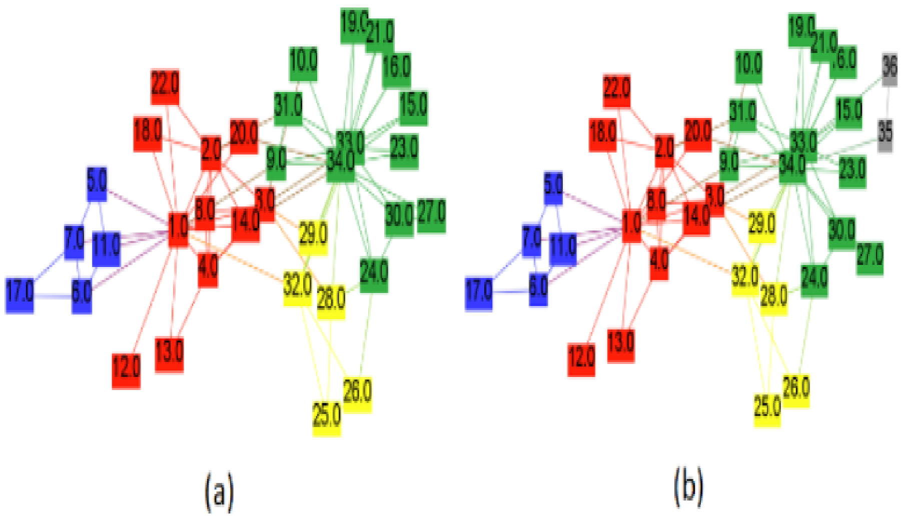
FIGURE 1: Zachary's karate club network [4]

or similar functions, which are expected to be involved in the same processes. Some communities in these networks are related to cancer and metastasis, hence, detecting these communities in PPI networks could be very important [18]. Community detection can also be helpful in solving influence maximization or viral marketing problems.

1.2 Dynamic Networks

Social networks may be classified as static or dynamic. Static networks do not change or evolve over time; in dynamic networks nodes may come and go, and edges might be added or deleted. In simple terms, the network topology changes over time. Social network researchers may seem to perceive the world as static but dynamic networks are present everywhere. Fifty four years ago, Wilbert Moore said [31] that "social sciences tended to neglect the way the limits and flows of time intersect the persistent and changeful qualities of human enterprises." This observation was based on the dominance of static models to study social networks. He further stated that "all analytical sciences tend to perfect their descriptions of elements and observations of combinations before they develop the capacity to observe orderly transformations in the course of time." Till this date, static models are dominant in social network analysis.

A dynamic network is modeled as a sequence of graphs $G_0, G_1, ..., G_n$, where $G_j = (V_j, E_j)$ denotes the graph at snapshot j, which consists of V_j nodes and E_j edges [43]. The example of Zachary's karate club network is useful to be reconsidered here. We run Louvain's [9] community detection algorithm on network G_0 which leads to four communities which are represented by four different colors shown in Figure 2.(a). Figure 2.(b) shows the next snapshot G_1 of the graph in which we have added two new nodes (node 35 and node 36) and 3 new edges from these nodes. Due to the change in network dynamics there is a need to re-run the community detection algorithm. The results of running community detection on graph G_1 are shown in Figure 2.(c). From this, we can observe that some of the communities are merged or split and new communities are created.



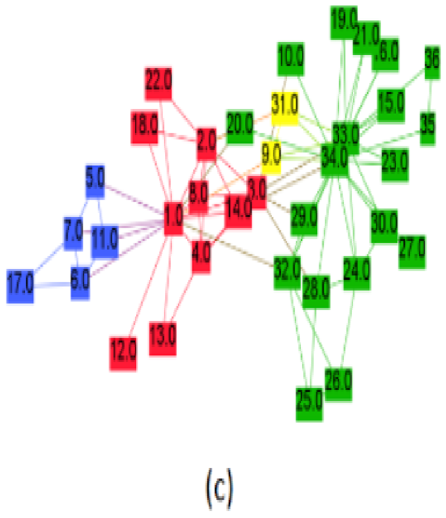


FIGURE 2: (a) Zachary's karate club original network (b) Zachary's karate club network after addition of new nodes and edges (c) Result of the Louvain algorithm on Zachary's network [4]

1.3 Problem Statement

In this thesis, we study the problem of community detection in dynamic networks. We assume we are given a dynamic evolving network, represented by snapshots $G_0, G_1, ..., G_n$, where $G_j = (V_j, E_j)$ denotes the graph at snapshot j, which contains $|V_j|$ nodes and $|E_j|$ edges. We assume that the only change to the network between the snapshots is the addition of new edges. We aim to have up-to-date and accurate community information for all snapshots. An obvious algorithm is simply to run a static community detection algorithm for every G_j. However this would be prohibitively expensive in general, and completely infeasible for large networks. In addition, community structure may not change significantly between snapshots.

How many edges need to be added to the network until the community structure changes? Are some kinds of edges likelier to change community structure than others? Are some nodes likelier to switch communities than others? How do we model the evolution of communities? We aim to get a better understanding of these questions in this thesis. Our final goal is to find algorithms to decide in every snapshot, knowing only which edges have been added to the network, whether or not the community structure is likely to have changed, necessitating the execution of a static community detection algorithm, to update the community structure.

1.4 Thesis Contribution

The contributions of this thesis are listed below:

- We classify edges into different categories and compute the minimum (threshold) number of different types of edges that would need to be added to a network before its community structure would change.
- We give several models for the addition of new edges to a social network.
- We give two new algorithms: the Edge-Distribution-Analysis algorithm , and the Modularity-Change-Rate algorithm to solve the problem of identifying the snapshots G_i in which to run a static community detection algorithm.
- We implement and run our algorithms on seven different benchmark social networks, and analyze the results.
- Our experiments show that both our algorithms do a good job at identifying snapshots in which to update the community structure. Compared to an approach

of updating community structure after a fixed number of edge additions, our algorithms obtain large savings in computation effort, while ensuring comparable quality of community structure.

1.5 Thesis Outline

Chapter 2 presents a literature review on community detection topics. First, we briefly explain the problem of community detection. Then we present different existing solutions for the problem. Chapter 3 presents our two new algorithms to solve the problem of dynamic community detection. Chapter 4 conducts a detailed study on generating evolving networks. Chapter 5 presents a comprehensive study of the performance of the proposed algorithms in the previous chapters. The final chapter concludes our thesis and gives some leads for future work.

Chapter 2

Related Work

In this chapter, we give a literature review on community detection. We start by attempting to define a community in Section 2.1. In Section 2.2, we study a few community detection techniques discussed within the literature. In Section 2.3, the literature on the Louvain algorithm and its variants have been reviewed. Section 2.4 extends the concept of community detection for evolving networks and details a few important algorithms from the literature. Finally, the limitations of existing algorithms have been outlined in Section 2.5.

2.1 What is a community?

In this section, we give the definition of community detection in depth and study some of the fundamental concepts of graph clustering. Afterwards, a brief explanation of the computation time for community detection is considered. There are many definitions of community detection. No definition is universally acclaimed. Often, the definition is based on the type of system under consideration or application dependent. After a careful review of these definitions, it can be concluded that communities are the subgraphs in which nodes are densely connected to each other when compared to the rest of the network. There are two types of community definitions; local and global. Local definitions focus on the subgraph under study but neglect the rest of the graph. On the other hand, in the case of global definitions, communities are defined with respect to the graph as a whole.

2.1.1 Local Definition

In social networks, a community often means a group whose participants are all friends with each other. In graph theory, such a group is termed as a clique in which every two distinct nodes are adjacent. This is a rather strict definition of community. According to this definition, if a single pair of nodes is not connected to other nodes in the network, it can be termed a community. Also a subgraph in which all but one pair of nodes is connected would not qualify as a community. Another important limitation of this definition comes about if we want to understand the hierarchical roles of nodes within the community. Using the definition of community as a clique, it is simply not possible.

One can always relax the notion of clique and define an n-clique. An n-clique is a maximal subgraph for which the distance between any pair of nodes does not exceed n. There are problems associated with this definition as well which are stated in [18]. There are some definitions available that are based on the similarity of the nodes. Each node ends up in a community whose nodes are most similar to it. This similarity can be a local or global property of the network. For example it can be geographical distance between the two nodes, the degree of the nodes etc. Therefore, two nodes who are not connected to each other by a short path might appear in the same community. [16] and [18] provide some number of suitable local definitions, however, in this work, our focus is on global definitions.

2.1.2 Global Definition

In contrast to local definitions, in a global definition of community, we specify not only the relationship between nodes within the community, but also their relationship to nodes outside the community. A community is defined as a subgraph in which nodes are densely connected to each other and sparsely to the rest of the network. Within such a context, a global property of the graph is used in an algorithm which delivers communities. A key idea in the literature is that if a network has community structure, it is different from a random graph. Several definitions [18] draw on this notion. The random graph defined by [15], will not have community structure. As any two nodes of the graph have the same probability of being connected to each other, as a result, there will not be any special group or community. In the literature, the *null model* is defined as a random graph which matches the given graph in some of its structural properties. This null model is used as a comparison tool in order to find out whether the original graph exhibits community structure or not. The most famous proposed null model by [34] rewires the edges randomly by keeping the degree of the node the same as it was in the original graph. This new random graph is used as the null model.

This null model is the basic concept behind the global parameter *modularity*. There are a lot of definitions of modularity based on different null models. According to [34], a subgraph is a community only if the total number of internal edges in the subgraph are greater than the expected total number of edges that are inside the same subgraph of the null model. Modularity also acts as a quality function which maps any partition of the graph to a numerical value. Then the communities can be said to be the partition that maximizes this value. Since we adopt this approach in our thesis, we describe it in detail in the next section.

2.1.3 Quality function: Modularity

Modularity is a function that evaluates the quality of a given partition of nodes in a graph as good communities. It is based on evaluating how much the graph, and the given partition differ from null model.

Consider a graph $G = (V, E)$ and let $|V| = n$ and $|E| = m$. Suppose we are given a community structure e for the graph where $e = C_1, C_2, ..C_k$ is a partition of V into communities. Define $d(v)$ to be the degree of node v. Let A be the adjacency matrix for G where $A_{vw} = 1$ means there is an edge from node v to node w and $A_{vw} = 0$ means there is no edge from node v to node w.

To obtain a null model, the following procedure is described in [34]. For every edge in the graph, we cut it in a half so we have two *stubs*. The total number of stubs is $2m$. We now reattach the stubs at random to obtain a null graph $G^{'}$. If $(v, w) \in e$, the probability that v and w are connected in $G^{'}$ is

$$\frac{d(v)d(w)}{2m}$$

Therefore, the difference between the actual number of edges and the expected number of edges between v and w is

$$A_{vw} - \frac{d(v)d(w)}{2m}$$

The modularity of the partition e is now defined to be the summation of the difference over all edges within communities [32]. In particular:

$$Q = \frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{d(v)d(w)}{2m}\right]\delta(c_v, c_w) \tag{2.1}$$

where c_v denotes the community to which node v belongs and $\delta(i, j) = 1$ if $i = j$ and 0 otherwise and $v, w \in V$. In [13], authors have simplified Equation 2.1. In order to do that, they define two variables which are:

$$e_{i,j} = \frac{1}{2m}\sum_{vw} A_{vw}\delta(c_v, i)\delta(c_w, j) \tag{2.2}$$

$e_{i,j}$ denotes the fraction of edges connecting nodes in community c_i and c_j. Further let,

$$a_i = \frac{1}{2m}\sum_{v} d(v)\delta(c_v, i) \tag{2.3}$$

a_i denotes the fraction of edges who have one endpoint in community i. It was shown in [13] that

$$Q = \sum_i (e_{ii} - a_i^2) \tag{2.4}$$

Notice that modularity can be seen in two different ways:

- Given a graph G and two different community structures e_1 and e_2 for G, the one that has a higher associated modularity value reflects tighter connections within communities. Thus modularity is a way of assessing different communities structures for a graph.
- Given two graphs G_1 and G_2 with similar structural characteristics (number of nodes, edges, and node degrees), and best possible community structure for them, the graph that has a higher modularity value can be said to have tighter communities. Thus modularity is a way of assessing how strong the community structure is in a graph.

It can be shown that the value of modularity resides in between $[-\frac{1}{2}, 1)$. In [32], the definition of this modularity is extended for weighted networks, but within this thesis, we focus on unweighted networks.

2.1.4 Maximizing a modularity is NP-hard

Theorem [3] of [12] states that maximizing modularity is strongly NP-complete. The proof is based on reduction from the 3-Partition decision problem which can be stated as follows: Given $3k$ positive integer numbers $a_1, ..a_{3k}$ such that the sum $\sum_{i=1}^{3k} a_i = kb$ and $\frac{b}{4} < a_i < \frac{b}{2}$ for an integer b and for all $i = 1, .., 3k$, is there a partition of these numbers into k sets, such that the numbers in each set sum up to b. Therefore, they suggest the use of heuristics for maximizing modularity.

2.2 Techniques of Community Detection

There is a variety of algorithms for community detection that are based on different definitions and the size of the networks. For every definition, there is at least one algorithm in literature. In [18], the authors perform an in-depth review of the relevant literature, detailing more than 20 algorithms for community detection. According to [38] and [18], community detection methods are divided into different classes. In this section, we describe several existing strategies from these classes namely traditional, divisive, modularity based and some of the other methods.

2.2.1 Traditional algorithms

Traditional algorithms are further categorized into following four classes:

- Graph partitioning: The problem of community detection has been studied since the 19th century. Earlier version of this problem resembled the problem of graph partitioning. The problem of graph partitioning consists of dividing the graph into fixed groups of a predefined size such that edges between groups are minimized. This problem is quite well-known due to its applications in parallel computing and circuit partitioning. Some of the popular approaches were *Kernighan-Lin* [23], spectral bisection [42] etc. Such algorithms are not good for community detection as they require the knowledge of the number of communities and the size of communities in advance to run the algorithm. In reality, one should find these values after running a community detection algorithm.
- Hierarchical clustering: Quite often, social network contain hierarchical structure. In such cases, one can use hierarchical clustering techniques such as:
    1. Agglomerative algorithms: Nodes are merged iteratively based on the some similarity measure.
    2. Divisive algorithms: Supernodes are split into nodes by removing edges which connects the dissimilar nodes. This creates structure similar to a dendrogram. From Figure 3, we can observe the hierarchical structure of nodes.
- Partitional clustering: Similar to graph partitioning, within partitional clustering, one should know in advance the number of clusters for given graph. The main goal is to divide the nodes into k clusters by maximizing or minimizing some measure such as the shortest distance, similarity etc. Famous approaches include k-median, k-clustering sum, minimum k-clustering sum. The cost function in case

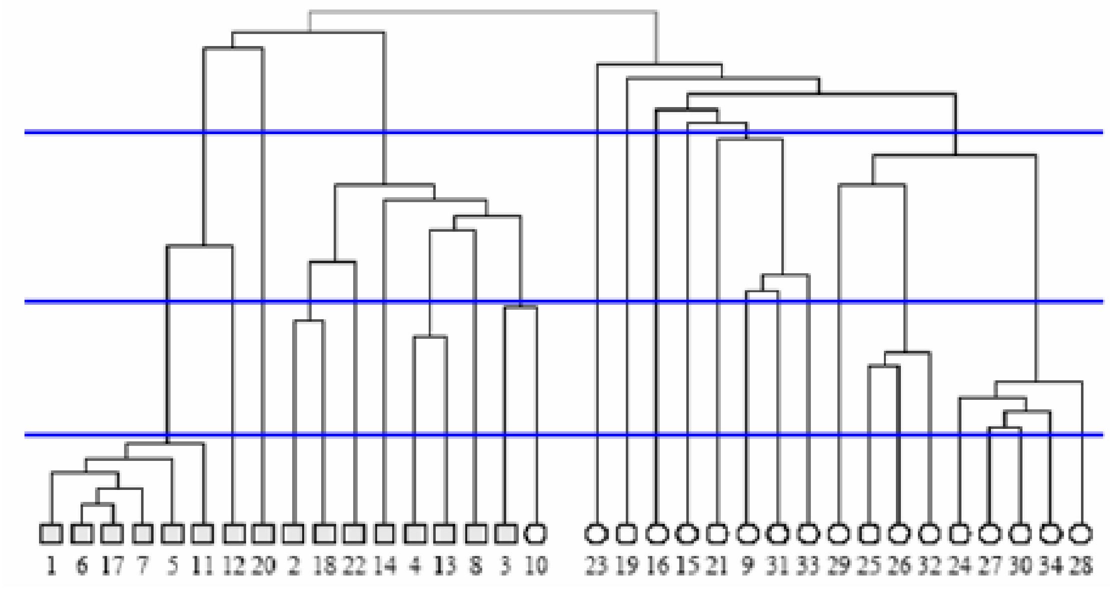


FIGURE 3: Hierarchical dendrogram for Zachary's karate network club [35]

of minimum k-clustering is the diameter of the cluster. For the largest cluster, diameter should be as small as possible. The idea is to keep the clusters compact. In case of an k-clustering sum, diameter is replaced by average distance for all pairs of points of a cluster. This approach also faces the same problems as graph partitioning.

- Spectral clustering: This type of clustering includes all the methods which use eigenvectors of matrices based on similarity to group the nodes. for example, algorithms given in [8] and [36].

2.2.2 Divisive algoritms

The Girvan and Newman algorithm [33] is a benchmark algorithm. It started a new era of community detection. Betweenness of an edge is defined as the total number of shortest paths for all the pairs of nodes that use the edge [18]. The steps of the algorithm are as follows:

1. Compute the edge betweenness centrality for every edge.
2. Remove the one with largest centrality; choose randomly in case of a tie.
3. Recalculate centralities for current version of graph.
4. Repeat from step 2.

The authors have used this algorithm for a network of jazz musicians and divided them into communities that are based on their collaboration efforts with each other. The worst case complexity of an algorithm is $O(n^3)$ which made it impossible to extend the algorithm for networks with size greater than 10000. Authors have used the edge betweenness as centrality measure in later versions to obtain better results. There are many other algorithms which use different centrality measures such as node centrality based on loops, similar to the clustering coefficient by [39].

2.2.3 Modularity based algorithms

The modularity function is by far the most used and most significant quality function. Girvan and Newman used it as a stopping criteria for their divisive algorithm. Since then a variety of algorithms have been proposed which focus on modularity optimization. Spectral clustering, divisive techniques or simulated annealing [22] have been used for modularity optimization. In 2008, a new heuristic for modularity optimization called the Louvain method was introduced by Blondel [9]. Today this is the fastest algorithm, for large networks. The modularity obtained by this algorithm is not the best as compared to other algorithms provided in literature. However, considering the time-modularity trade off, it gives exceptional results. This algorithm is further extended by Noack and Rotta [41] to improve the proposed heuristic. Their algorithm is known as *The Louvain algorithm with multilevel refinement*. In [30], they exploit the measure of edge centrality for modularity optimization. Their approach is called *Generalized Louvain method*. In 2013, Waltman & Ludo [47] introduced a newer algorithm called SLM based on [9]. We have used the aforementioned algorithms in our thesis. A detailed discussion of these algorithms is provided within Section 2.3. The limitations of modularity based algorithms are discussed at the end of the chapter.

2.2.4 Other methods

In [18], the authors have published a book explaining all the community detection algorithms present till 2010. Apart from the ones that we have mentioned, there are other algorithms based on spin models, conformational space annealing [26], random walks and synchronization. Also, there is separate section for methods based on statistical inference. Some of these algorithms address different aspects of community detection which are not our focus. For example, our thesis does not focus on overlapping communities but there are few algorithms which allow for this.

2.3 Communities in large networks: The Louvain Algorithm and its variants

As per our discussion in Subsection 2.2.3, the Louvain algorithm is a benchmark algorithm for community detection in large networks. In this section, we give a detailed explanation of this algorithm and its variants. A fundamental part of this algorithm is the so-called *local moving heuristic*. The concept behind the local moving heuristic is to move the nodes repeatedly from one community to another community as long as there is gain in modularity. The gain in modularity by moving an isolated node v into community C_i is calculated as follows:

$$\Delta Q = \left[\frac{E_i + 2 * d_i(v)}{2m} - \left(\frac{D(i) + d(v)}{2m}\right)^2\right] - \left[\frac{E_i}{2m} - \left(\frac{D(i)}{2m}\right)^2 - \left(\frac{d(v)}{2m}\right)^2\right] \tag{2.5}$$

where $d(v)$ denotes the degree of the node, m denote the total number of edges, $D(i)$ the sum of the degrees of the nodes in community C_i, $d_i(v)$ represents the number of edges from node v to other nodes in community C_i and finally, E_i denotes the total number of edges for which both the endpoints are in community C_i. The first part of Equation 2.5 represents the modularity obtained by moving node v into community C_i. The second part of shows the modularity obtained when the node v is an isolated node. Equation 2.5 uses the simplified Equation 2.4 for modularity.

2.3.1 Local moving heuristic

In this section, we describe the Local Moving Heuristic. The local moving heuristic has been used in [6], [9], [41], [29], [47] etc. Initially, every node is in its own community. Next, we take an arbitrary node and see if including it in any of its neighboring communities increases the modularity. We check all neighboring communities and find the best community that v can move to. We repeatedly do this until no nodes switch communities. The pseudocode is given in Algorithm 1.

In line 1, we shuffle the nodes randomly. According to [9], the order in which nodes are chosen might affect the computation time. The do-while loop of lines 4-18 iterates as long as total number of stable nodes are less than total number of nodes. In line 5, we choose the node for which we will calculate the best possible community. In for loop of lines 6-11, we compute the gain in modularity by moving singleton node v to adjacent community c using Equation 2.5. Here adjacent community means that node v has direct edge to any of the nodes in community c. If the node v is already assigned to some community then, before the next step, it is removed from that community & similar expression to Equation 2.5 is used to calculate this change in modularity. In

Algorithm 1 LocalMovingHeuristic(G)

```
1:  $V = \textsc{Shuffle}(V)$
2:  $j = 0$
3:  $update = false$
4:  do
5:      $v = V[j]$
6:      for each $c \in G.Adj[c_v]$ do
7:          compute $\delta Q$ for node $v$ and community $c$ using Equation 2.5
8:          if $\delta Q \geq maxQ$ then
9:              $maxQ = \delta Q$
10:             $bestCluster = c$
11:         end if
12:     if $bestCluster == c_v$ then
13:         $nStableNodes \leftarrow nStableNodes + 1$
14:     else
15:         $nStableNodes = 1$
16:         $update = true$
17:     end if
18: while $nStableNodes < V$
19: return $update$
```

practice, one considers this change and the gain obtained by moving node to community c. Lines 8-11 keeps track of the maximum gain in modularity and best community assignment for node v. Lines 12-13 checks whether the previous assignment of node v is the still the best. In such case, we increase the total number of stable nodes by 1. Otherwise, we initialize the update to true and the number of stable nodes to 1. This means that algorithm might consider the same nodes many times. This procedure keeps running until one does find the best possible assignment for every node.

Figure 4 shows an example of running local moving heuristic to the Karate club network [52] that has 34 nodes and 78 edges. After running the Local Moving Heuristic, we observe that there are six communities. The green and red communities contain most of the nodes.

2.3.2 Louvain Algorithm

The Louvain algorithm adds an additional level to the Local Moving Heuristic(LMH). Essentially, it takes the communities given by LMH and creates a reduced graph in which nodes correspond to the previously found communities. It then recursively finds communities in this reduced graph. The recursive calls stop when every node in the reduced graph stays in a singleton community when the LMH is applied. Finally, the nodes of the reduced graph are used to assign communities in the original graph. The pseudocode of the Louvain algorithm is given in Algorithm 2.

The Louvain algorithm works as follows: Lines 1-3 checks whether the graph has more than one nodes and returns false in such cases. In line 4, we read the network and assign every node to an individual singleton community. In line 5, we run the LocalMovingHeuristic and pass graph G as parameter. The aim of the local moving heuristic is to achieve the highest possible gain in modularity for every node. Lines 6-14 are executed only if the total number of communities are less than the total number of nodes. In line 7, we create reduced network with less number of nodes based on the resulting communities that are obtained from local moving heuristic. This is the first phase of an algorithm. From line 8, we can observe that the Louvain algorithm is written in recursive fashion. The reduced network is passed as a new instance for the Louvain algorithm. Lines 9-13 are executed only if through running the Louvain algorithm we have obtained positive gain in modularity. The for loop of lines 11-12 assigns the new communities to nodes based on the output of every phase. This is the merging step of the Louvain algorithm. After this we explain the construction of the reduced network in depth.

We give an example of running the Louvain algorithm on the Karate club network [52] which has 34 nodes and 78 edges. Figure 4 illustrates an example of running local moving heuristic, we observe that there are six communities. Algorithm 3 gives the pseudocode for the construction of the reduced network. In line 1, we create the nodes of the reduced network. Every community obtained after running the local moving heuristic will be considered as a node in reduced network. $V_v^{'}$ denotes the new supernode in the reduced network which containd node v. In the outer for loop, each community is considered one by one. For all the nodes which belongs to community under consideration, we check for every edge whose one of the endpoint is node v. The weight of the edges between two supernodes denote the sum of all the links between the nodes in the

Algorithm 2 LouvainAlgorithm(G)

```
1: if $G.V == 1$ then
2:     return $false$
3: end if
4: ReadInput($G$)
5: $update$ = LocalMovingHeuristic($G$)
6: if $|G.C| < |G.V|$ then
7:     $G^{'}$ = CreateReducedNetwork($G, C$)
8:     $newUpdate$ = LouvainAlgorithm($G^{'}$)
9:     if ($newUpdate$) then
10:        $update = true$
11:        for $v = 1$ to $V$ do
12:            $C[v] = C[C^{'}[v]]$
13:    end if
14: end if
15: return $update$
```

Algorithm 3 CreateReducedNetwork(G, C)

```
1: Let $V^{'} = 1, 2, ..C$
2: for $i = 0$ to $C$ do
3:     for each $v \in C_i$ do
4:         for each edge $(v, w) \in E$ do
5:             $e(V^{'}_v, V^{'}_w) \leftarrow e(V^{'}_v, V^{'}_w) + 1$
6: return $G^{'}$
```

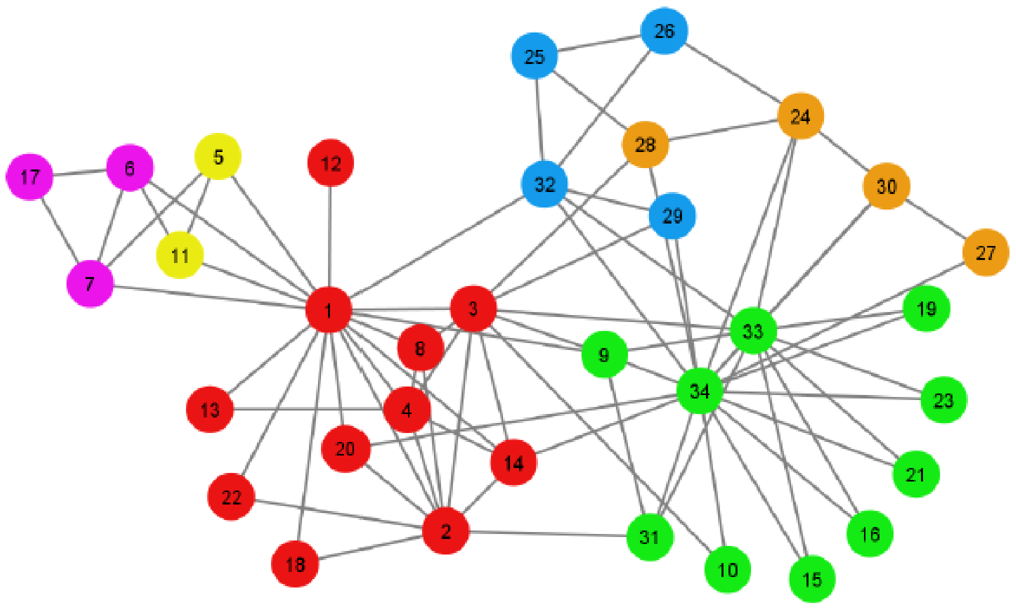


FIGURE 4: Result of applying local moving heuristic to karate club network [47].

corresponding two communities. Edges between the nodes of the same community are treated as self loops for this supernode. This is achieved by line 5. In line 6, we return the reduced network. Figure 5 gives the results of applying the Louvain algorithm for the karate network. Figure 5.a represents the reduced network of six communities after running local moving heuristic. We have not shown the self loops in it but they do exist. After running local moving heuristic again on this reduced network we get Figure 5.b. We can notice that nodes b and c now belong to same community. Similarly nodes e and f also now belong to same community. The new reduced network consists of four nodes. In the next phase, on running local moving heuristic, the change in modularity is not positive. We use these nodes from reduced network to assign communities in original graph. Hence, the final community structure is shown in Figure 5.c.

2.3.3 Louvain Algorithm with Multilevel Refinement

An extension of the Louvain algorithm was proposed in [41] in 2011. After running the Louvain algorithm we observed that no further gain in modularity is possible by merging communities. Actually, this is the stopping phase for the algorithm, in simple words, we find the locally optimal solution with respect to merging of the communities. In the Louvain algorithm, once the nodes are merged into supernode, individual nodes from the supernode cannot be moved to other communities. However, the final community structure obtained by Louvain, can further be improved by allowing individual movements for the nodes.

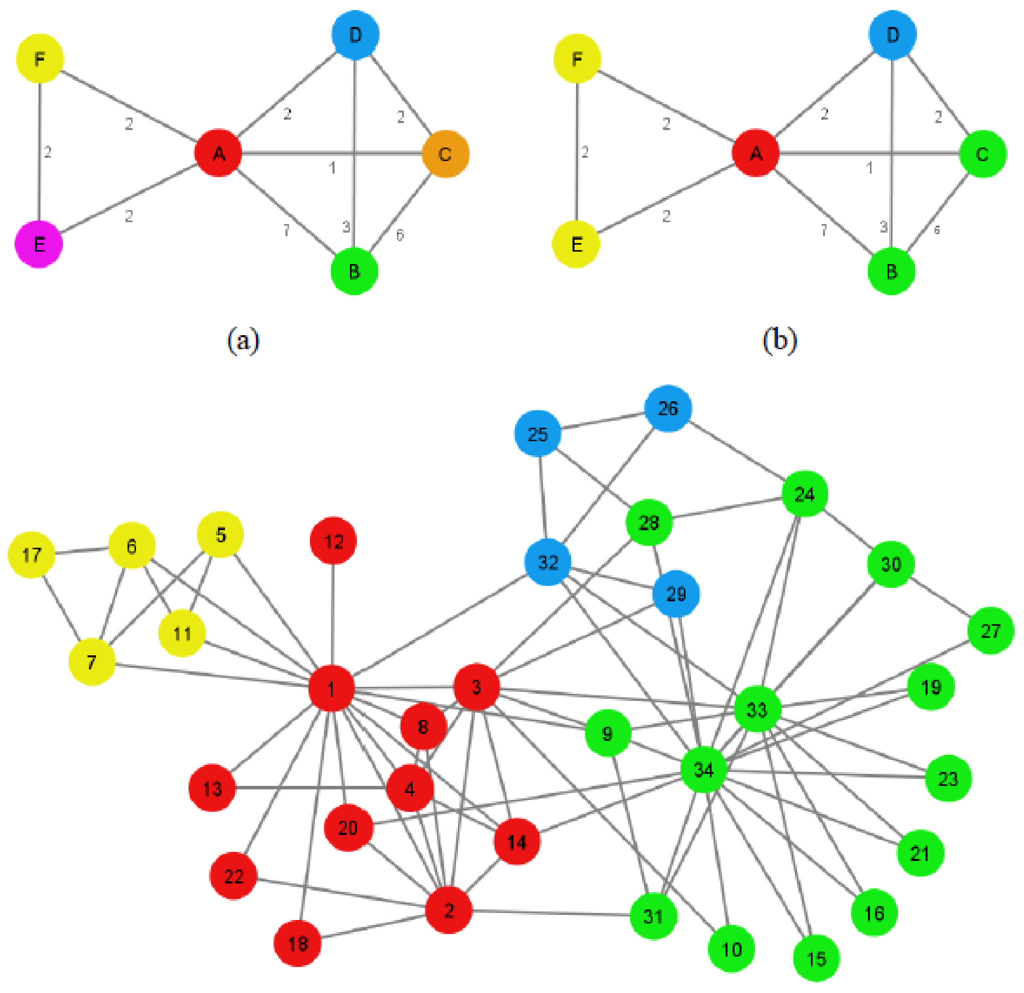


FIGURE 5: Result of applying the Louvain algorithm to the karate club network[47]. (a) Reduced network before applying LMH (b) Reduced network after applying the LMH (c) Final solution in the original network

The Louvain algorithm with multilevel refinement [41] improves the solution of the Louvain algorithm so that it becomes locally optimal with respect to individual node movements. In order to perform this, the authors run the local moving heuristic twice. First, to obtain the initial community structure for the reduced network, and after that, they run local moving heuristic again to allow individual node movements. This procedure is applied to each level of an algorithm. Hence, it is called as multilevel refinement. This algorithm gives a significant improvement over the Louvain algorithm in terms of modularity, but the running time was also increased.

2.3.4 SLM Algorithm

Another extension of the Louvain algorithm was proposed by [47]. Their solution is also locally optimal with respect to community merging and individual node movements. In addition to this, this solution checks for further improvements in modularity by splitting up communities and moving a set of nodes from one community to other communities. The idea of the algorithm can be summarized in 3 steps:

1. Run the Local Moving Heuristic on graph G to obtain an initial community structure $e = C_1, C_2, ..C_k$
2. Run separately the LMH on communities C_i to break (potentially) C_i into subcommunities $C_{i,1}, C_{i,2}, ..C_{i,j}$
3. Create a reduced graph with $C_{i,1}, C_{i,2}$ etc. as nodes (for all i), but use the original community structure e, and recursively call the algorithm on the reduced graph and community structure.

Algorithm 4 gives the pseudocode of the SLM algorithm.

Lines 1-3 check whether the graph contains a single node; in such a case, it returns false. In line 4, we read the graph and assign every node to individual community. In line 5, we run local moving heuristic as in to the Louvain algorithm. Lines 6-17 are executed only if number of communities obtained from local moving heuristic is less than total number of nodes. From line 7, authors take different approach. Instead of creating the reduced network right away, they construct a copy of subnetwork for every community present in the current community structure. This copy will contain the nodes belonging to particular community of interest of the original network. For loop of lines 9-13, runs local moving heuristic for each of the subnetwork in order to identify the communities inside the subnetwork. Similar to original procedure, every node of the subnetwork is assigned to individual community and then local moving heuristic is ran on it. The result after running the local moving heuristic on subnetwork might be a single big community including all the nodes of subnetwork or it might consists of multiple communities including some of the nodes from subnetwork. In line 13, nClusters will contain total number of communities obtained by running local moving heuristic on all the subnetworks. In line 14, one creates a reduced network where each node of reduced network represents a community obtained from one of the subnetworks. Nodes corresponding to communities in the same network are assigned to same community in the reduced network. Therefore, for every subnetwork one gets single community in the reduced network. This is achieved in lines 12-13. Once the reduced network is created, we recursively call SLM on this network.

To illustrate the SLM algorithm, the karate club network is reconsidered. Figure 6(a) shows the six communities which are generated after running local moving heuristic.

Each community is represented by different colour. For each community new subnetwork is created. On every subnetwork, local moving heuristic is applied. For green, blue, purple and yellow it results into all nodes being assigned to a single community. In the case of red and orange, subnetworks are split into two communities. This is represented by different shapes such as the squares and circles. Figure 6(b) shows the reduced network that is obtained. In this reduced network, we have 8 nodes. Each node represent a community in subnetwork. Nodes corresponding with communities in the same subnetwork are assigned to the same community initially. The result of applying local moving heuristic on it is shown in 6(c). We observe that nodes $A1$ and $A2$ remain in the same community but nodes $C1$ and $C2$ now belong to different communities. Afterwards, we again create subnetworks and run the local moving heuristic for every subnetwork. We do not show the result of this, since it turns out that the community structure shown in Figure 6(c) cannot be improved further.

2.3.5 Iterative variant of these algorithms

The authors of the [47], introduced an approach that aims to improve all of the stated algorithms above. The basic idea of this approach is to run in iterative fashion, where the output of previous iteration will be the starting community assignment for the next iteration. In any of the algorithms like Louvain, Multilevel Louvain or SLM, they start by assigning each node to an individual community. This approach will do the same, but after the first iteration, the result of the first iteration is given to second iteration. Therefore, for the second iteration, one does not start with every node as singleton community. Instead one starts with the community structure of the first iteration. The procedure is repeated for the number of iterations specified or one can stop the algorithm when an additional iteration is not giving a gain in the modularity. Also in the original paper of Louvain, it is mentioned that the order in which nodes are chosen is important, hence, certain number of random starts are provided to obtain the best result. Algorithm 5 explains the procedure. In line 6, by *runAlgorithm* we mean that you can run Louvain, Multilevel Louvain or SLM.

Algorithm 4 SLMAlgorithm(G)

```
1:  if $G.V == 1$ then
2:      return $false$
3:  end if
4:  ReadInput($G$)
5:  $update$ = LocalMovingHeuristic($G$)
6:  if $G.C < G.V$ then
7:      $G$ = CreateSubNetworks($G, C$)
8:      $nClusters = 0$
9:      for each edge subnetwork $G_i \in G$ do
10:         LocalMovingHeuristic($G_i$)
11:         for $j = 0$ to $V_i$ do
12:             $C[j] = nClusters + C_i[j]$
13:         $nClusters \leftarrow nClusters + C_i$
14:     $G^{'}$ = CreateReducedNetwork($G, C$)
15:     $update$ = SLMAlgorithm($G^{'}$)
16:     for $v = 1$ to $V$ do
17:         $C[v] = C[C^{'}[v]]$
18: end if
19: return $update$
```

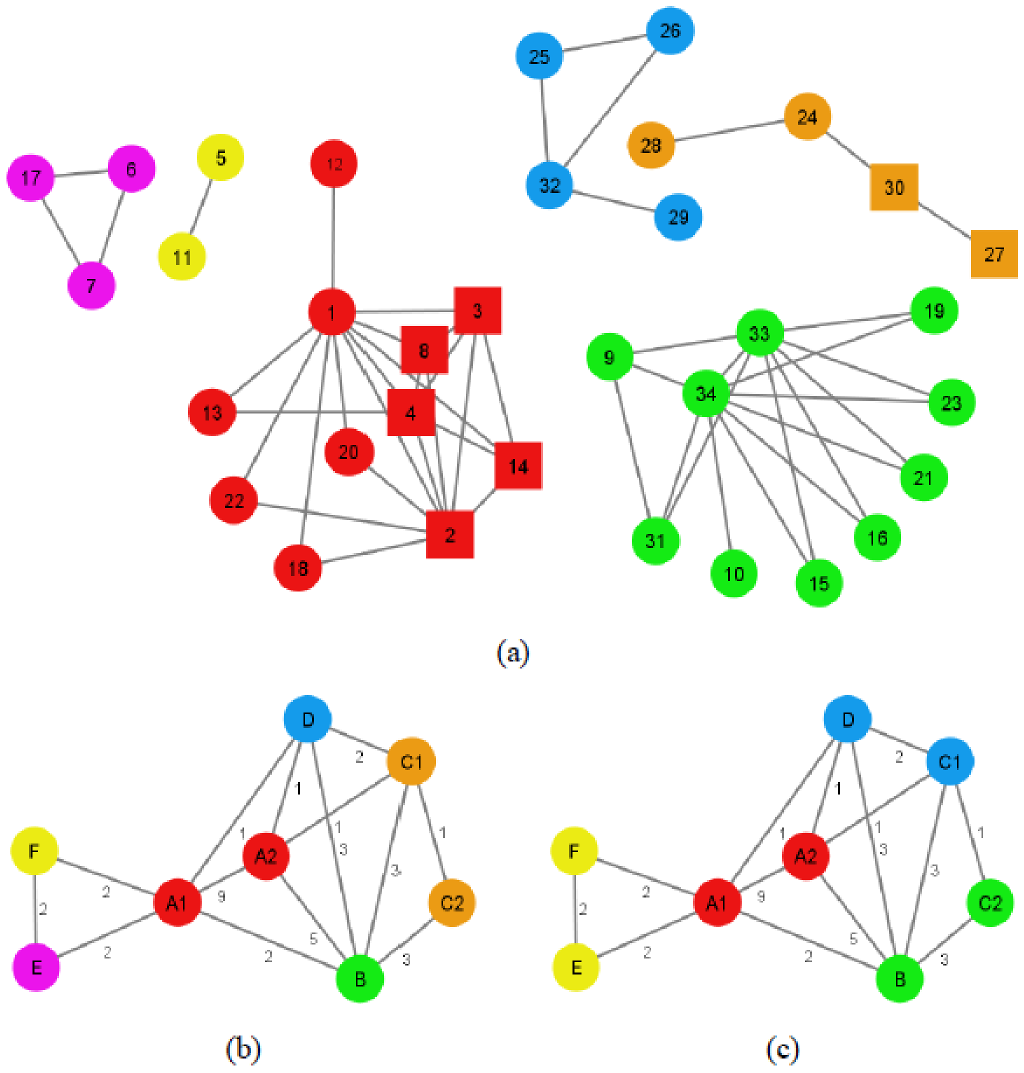


FIGURE 6: Result of applying the SLM algorithm to karate club network[47]. (a)LMH is applied on six subnetworks. Nodes in the subnetwork are displayed using either square or circle. (b) Reduced network before applying the LMH. (c) Reduced network after applying LMH.

Algorithm 5 ITERATIVE VARIANT FOR ALGORITHMS(G)

```
1: for $i = 0$ to $randomStarts$ do
2:     $update = true$
3:     $iteration = 0$
4:     do
5:         $IntializeCommunities(C)$
6:         $update = runAlgorithm(G)$
7:         $modularity = CalculateModularity(G, C)$
8:         $iteration \leftarrow iteration + 1$
9:     while $iteration < nIterationsandupdate$
10:    if $modularity \geq maxModularity$ then
11:        $maxModularity = modularity$
12:    end if
13: print $maxModularity$
```

2.4 Dynamic Community Detection for Evolving Networks

Dynamic community detection is still in its infancy. The reason for this is, as the problem is **NP**-hard, solving the problem of community detection for static graph is already difficult. Thus, most of the studies focus on the practically efficient static versions of this problem. In [18], it is mentioned that the main phenomena occuring in the lifetime of community are birth, growth, contraction, merger with other communities, split and death. In the literature, some of the studies like [44], [43], [10] have focused on predicting the evolution of networks and then they use that information for community detection. On the other hand some approaches like [3] focus on using the information from the previous snapshots and perform community detection based on it.

2.4.1 DSLM Algorithm

The authors of [3] have extended the SLM algorithm for dynamic community detection. The idea of the DSLM algorithm is summarized as follows:

1. Detect the new nodes which are added to the network and assign them as singleton communities.
2. Read the previous community structure for the given graph G and assign the current nodes to the communities based on it.
3. Run the SLM algorithm with this community structure as starting assignment.

Addition of edges, deletion of nodes and edges is handled by the SLM algorithm. This algorithm does very well in terms of running time as compared to running SLM from scratch on G_i. The main innovation in this algorithm is that rather than running SLM from scratch on each G_i, we use the community structure of G_{i-1} that was derived previously as a starting point for SLM. This is responsible for saving time.

The authors of DSLM do not indicate how often to run the algorithm. For example after the addition of how many nodes or edges should DSLM be run? In terms of modularity DSLM gives more or less the same result as SLM.

2.4.2 Community Evolution Prediction Algorithm

In [44], the authors have proposed a machine learning model to accurately predict the changes and transitions of the community based on structural and temporal properties.

Community transitions and events are considered as response variables. They have used features of communities such as density, clustering coefficient, number of nodes, cohesion, average closeness, average degree, variance closeness and variance degree which influence one of the response variables. Different snapshots of the networks are collected and analyzed. Next, they have used logistic regression and various classification functions to train the data and predict the community transitions. Their experiments show that defined features are non overlapping and community transitions and events are predicted accurately.

2.5 Limitations of existing algorithms

We have surveyed different definitions of communities and algorithms for community detection problem. First of all, the solutions given in the literature are heuristics hence, it is not possible to find the exact solution. Secondly, due to the plethora of application-based definitions, it is hard to use one particular algorithm for all the community detection problems. For example, the authors of [48], [50] have focused on finding overlapping communities, while according to the Louvain algorithm, one node cannot be part of two communities at the same time. In [5], the focus is on finding communities in bipartite graphs. The authors of [51] give an algorithm to find out 2-mode communities or hidden communities in which nodes might not have direct links to each other but they have links to other nodes in coordinate way. For example, a network of politicians where hidden community can be presidents of nations but in the network they might not be connected at all but their degree distributions are similar.

In this thesis, we will be focusing on modularity-based algorithms. As per [12], maximizing modularity is NP-hard. Also one of the major drawback for all the modularity based approach is the so-called *resolution limit* [18]. Due to this, in large networks sometimes such algorithms fail to resolve small communities even when they are well defined. The reason behind this is that, in the null model, it is assumed that nodes can get connected to any of other nodes in the graph. This assumption is a little unreasonable as the horizon of the node is limited to a small network. As a result, the expected number of edges between the two groups is decreased. In some of the cases, a single edge between two small clusters might lead to their merging. In [46], the authors have cited algorithms which do not suffer from the problem of resolution limit.

Finally, there are not many promising algorithms for the problem of dynamic community detection for evolving networks. One of the most promising approaches is [44] but it also predicts transition of communities with accuracy varying from 60 % to

90 %. In the next chapter, we propose two algorithms for solving community detection for evolving networks.

Chapter 3

Two Algorithms for Dynamic Community Detection

In this chapter, we propose two new algorithms for dynamic community detection. We assume that we are given a partition of a graph into communities and a set of new edges are subsequently added to the graph. Some existing approaches for dynamic community detection in evolving networks were described in the previous chapter. The analysis of dynamic communities is still in its infancy [18].

3.1 Notation and preliminaries

The following notations will be used throughout the chapter [43]. A dynamic evolving network is modeled as a sequence of graphs $G_0, G_1, ..., G_n$, where $G_j = (V_j, E_j)$ denotes the graph at snapshot j, which contains V_j nodes and E_j edges. Let $G_j.mod$ represent the modularity for the graph G_j. We start with the discussion of the number of edges that need to be added to the graph in order for the community structure to change.

Fix a graph $G = (V, E)$. The remaining discussion in this section pertains to any such graph G. Let $\mathcal{C}$ be a community structure for G obtained by running the SLM algorithm. [1]

We use $d(v)$ to denote the degree of node v, and $d_i(v)$ to denote the number of edges from node v to other nodes in the community C_i. Furthermore, let $D(i)$ be the

[1] As per our discussion in Section 2.1.4, finding maximum modularity is **NP**-hard. This community structure does not have maximum modularity.

sum of degrees of nodes in the community C_i, that is:

$$D(i) = \sum_{v \in C_i} d(v)$$

Finally, let m represent the total number of edges in the graph.

The gain in the modularity by moving an isolated node v to community C_i is obtained from Equation 2.5.

$$\delta Q = d_i(v) - \frac{d(v)D(i)}{2m} \tag{3.1}$$

Fact 1. *Let G be a graph and let $\mathcal{C}$ be a community structure for G computed by SLM or DSLM [47],[3]. Then $\mathcal{C}$ is locally optimal in the sense that for every node $v \in C_i$ and $C_j \neq C_i$, we have:*

$$d_i(v) - \frac{d(v)D(i)}{2m} > d_j(v) - \frac{d(v)D(j)}{2m}$$

Given a community structure $\mathcal{C}$, and a node $v \in C_i$, we say v wants to *switch communities* if there exists a community C_j with $j \neq i$ such that

$$d_j(v) - \frac{d(v)D(j)}{2m} > d_i(v) - \frac{d(v)D(i)}{2m}$$

Also, community C_j will be a *neighboring community* of node v, if there is a direct edge between node v and any node in community C_j. Node v might want to *switch communities* based on various factors like the addition of edges, deletion of edges, addition of new nodes or deletion of new nodes. In this thesis we will be focusing on the addition of edges. We can distinguish two types of edges. We call an edge an *intra-edge* if it connects two nodes in the *same community.* We call an edge an *inter-edge* if it connects two nodes in *different communities.*

3.2 Effect of edge additions on community structure

In this section, we study the effect of edge additions on community structure. We study first the effect of adding intra-edges, then the effect of adding inter-edges.

The main question we seek to answer is : how many edges need to be added to the graph before the community structure is changed. We focus on a single node v in a community C_i. How many and what kind of edge additions would cause v to switch communities? We study intra- and inter-edge additions separately in the next two sections.

3.2.1 Addition of intra community edges

In this section, our aim is to understand the effects of intra-edge additions on community structure $\mathcal{C}$ in the graph G. We would like to identify the maximum number of intra-community edges that can be added without affecting the community structure. Consider a node v in C_i. From $v^{'}$s vantage point, intra-edges can be further classified into 4 types, as shown in Figure 7.

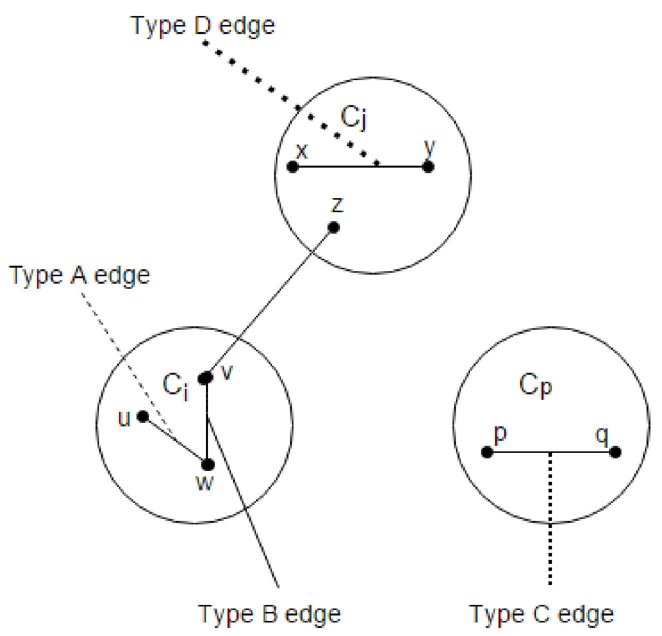


FIGURE 7: Types of intra edge additions-C_j is a neghboring community of node v, but C_p is not.

We call an intra-edge a type A edge if it connects two nodes in C_i but it is not incident on v itself. We call it a type B edge if it connects v to another node in C_i. We call it a type C edge if it connects two nodes in community C_p where C_p is not a neighboring community of node v. An intra-edge is a type D edge if it connects two nodes in community C_j where C_j is a neighboring community of node v. Our results for each type of intra edge addition are summarized in Table 1.

Type of edge	Proposition
A	Lemma 1
B	Lemma 1
C	Lemma 2
D	Lemma 3

TABLE 1: Propositions for different types of intra edge addition

Type A and B intra-edges

The following lemma considers the addition of type A and B intra-edges to the graph G.

Lemma 1. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let C_j be a neighboring community of node v with $j \neq i$. Then node v does not want to switch to community C_j, if at most κ_v^j new type A and B intra-edges are added between nodes in C_i, where κ_v^j is given by:*

$$\kappa_v^j = \left\lfloor \frac{2m(d_i(v) \; - \; d_j(v)) \; + \; d(v)(D(j) \; - \; D(i))}{2(d_j(v) \; - \; d_i(v) \; + \; d(v))} \right\rfloor$$

and there is no other change to the graph.

Proof. Since $\mathcal{C}$ is a locally optimal community structure for G, observe that by Fact 1, we have:

$$d_i(v) - \frac{d(v)D(i)}{2m} \geq d_j(v) - \frac{d(v)D(j)}{2m}$$

Now, suppose that $k \leq \kappa_v^j$ new edges between nodes in C_i are added to G. First v is not the endpoint of any of the k newly added edges that is, all the new edges are Type A edges with respect to node v. Observe that $d(v), d_i(v), d_j(v), D(j)$ will remain unchanged. However, $D(i)$ and m are both incremented by $2k$ as a result of the addition of the k new intra edges.

For node v, we will compare the change in modularity if v switches to C_j. We only consider a neighboring community C_j since only $D'(i)$ and m' have changed. So, in this new graph G', when we compare the change in modularity for an isolated node

v against community C_i and C_j, we have:

$$k \leq \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2(d_j(v) \ - \ d_i(v) \ + \ d(v))}$$

$$\implies 2d_j(v)k + 2d(v)k - 2d_i(v)k \ \leq \ d_i(v)2m - d(v)D(i) + d(v)D(j) - 2d_j(v)m$$

$$\implies 2d_j(v)m + 2d_j(v)k - d(v)D(j) \ \leq \ d_i(v)2m + 2d_i(v)k - d(v)D(i) - 2d(v)k$$

Dividing by $2m + 2k$ on both sides, we obtain:

$$\frac{2d_j(v)m \ + \ 2d_j(v)k \ - \ d(v)D(j)}{2m \ + \ 2k} \leq \frac{d_i(v)2m \ + \ 2d_i(v)k \ - \ d(v)D(i) \ - \ 2d(v)k}{2m \ + \ 2k}$$

$$\implies d_j(v) - \frac{d(v)D(j)}{2m+2k} \ \leq \ d_i(v) - \frac{d(v)(D(i)+2k)}{2m+2k} \tag{3.2}$$

This implies that node v does not want to switch to community C_j. Next, we consider the case when some of the newly added edges are incident to v, i.e type B intra-edges. We need the following technical claim.

Claim 1.
$$\frac{D(i) - D(j) - 2m}{d_j(v) \ - \ d_i(v) \ + \ d(v)} \leq 0$$

Proof. Observe that

$$d_i(v) \leq d(v)$$
$$\therefore d(v) - d_i(v) + d_j(v) \geq 0$$

On the other hand,

$$D(i) \leq 2m$$
$$\therefore D(i) - D(j) - 2m \leq 0$$

The claim follows. ∎

Now, suppose we add $k \leq \kappa_v^j$ new edges between pairs of nodes (u, w) where $u \in C_i$, $w \in C_i$ and $u \neq w$. Out of these k edges, let p be type B intra-edges, that is

$v = u$ or $v = w$. We know that

$$k \leq \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2(d_j(v) \ - \ d_i(v) \ + \ d(v))}$$

After the addition of k edges, we have:

$$\begin{aligned} d_i'(v) &= d_i(v) + p, \\ d'(v) &= d(v) + p, \\ D'(i) &= D(i) + 2k, \\ m' &= 2m + 2k \end{aligned}$$

$D(j)$ remains the same. Observe that

$$2k \leq \left(\frac{2m(d_i(v) \ - \ d_j(v) + d(v)(D(j) \ - \ D(i)))}{(d_j(v) \ - \ d_i(v) \ + \ d(v))} \right)$$

$$\implies 2k + p\left(\frac{D(i) - D(j) - 2m}{d_j(v) \ - \ d_i(v) \ + \ d(v)} \right) \leq \left(\frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{(d_j(v) \ - \ d_i(v) \ + \ d(v))} \right)$$

where the implication follows from Claim 1

Simplifying, we obtain:

$$2k(d_j(v) - d_i(v) + d(v)) + p(D(i) - D(j) - 2m) \leq 2m(d_i(v) - d_j(v)) + d(v)(D(j) - D(i))$$

Dividing by $2m + 2k$ on both sides, we obtain:

$$\frac{2md_j(v) + 2d_j(v)k - d(v)D(j) - pD(j)}{2m + 2k} \leq \left(\frac{2md_i(v) + 2mp + 2d_i(v)k}{2m + 2k} \right.$$

$$\left. + \frac{-d(v)D(i) - 2d(v)k - pD(i)}{2m + 2k} \right)$$

$$\equiv d_j(v) - \frac{(d(v) \ + \ p)D(j)}{2m + 2k} \leq d_i(v) \ + \ p - \frac{(d(v) \ + \ p)(D(i) + 2k)}{2m + 2k} \tag{3.3}$$

Once again, this means that node v does not want to switch to community C_j. ■

Type C intra-edges

The next lemma considers the addition of type C intra-edges to the graph G.

Lemma 2. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let $S(v)$ be the set of all the neighboring communities of node v and C_j be a neighboring community and C_p is non-neighboring community. Then node v does not want to switch to community C_j, if we add at most α_v^j new type C intra-edges to community C_p where, $C_j \in S(v)$, and α_v^j is given by:*

$$\alpha_v^j = \frac{2m(d_i(v) \; - \; d_j(v)) \; + \; d(v)(D(j) \; - \; D(i))}{2(d_j(v) \; - \; d_i(v))}$$

and there is no other change to the graph.

Proof. Suppose we add $k \leq \alpha_v^j$ new edges between pairs of nodes (u, w) where, $u \in C_p$, $w \in C_p$, and $C_p \notin S(v)$. For the node $v \in C_i$. Clearly, $d(v), d_i(v), d_j(v)$ will remain unchanged. Also, we have not added any edges to communities in $S(v)$ hence $D(i)$ and $D(j)$ will also remain unchanged.

In the new graph G', when we will compare the change in modularity for an isolated node v against community C_i and C_j, we have:

$$k \leq \frac{2m(d_i(v) \; - \; d_j(v)) \; + \; d(v)(D(j) \; - \; D(i))}{2(d_j(v) \; - \; d_i(v))}$$

$$\equiv 2(d_j(v) \; - \; d_i(v)) \; \leq \; 2d_i(v)m \; - \; d(v)D(i) \; - \; 2d_j(v)m \; + \; d(v)D(j)$$

$$\equiv 2d_j(v)m \; + \; 2d_j(v)k \; - \; d(v)D(j) \; \leq \; 2d_i(v)m \; + \; 2d_i(v)k \; - \; d(v)D(i)$$

Dividing by $2m + 2k$ on both sides, we get:

$$\frac{2d_j(v)m \; + \; 2d_j(v)k \; - \; d(v)D(j)}{(2m \; + \; 2k)} \quad \leq \quad \frac{2d_i(v)m \; + \; 2d_i(v)k \; - \; d(v)D(i)}{(2m \; + \; 2k)}$$

$$\equiv d_j(v) \; - \; \frac{d(v)D(j)}{(2m \; + \; 2k)} \; \leq \; d_i(v) \; - \; \frac{d(v)D(i)}{(2m \; + \; 2k)} \tag{3.4}$$

According to Equation (3.4), community C_i is still the best option for node v between C_i and C_j, hence, there is no change in the community structure. So unless we add more than α_v^j edges, node v will not switch its community to C_j. Similarly, we can calculate α_v^j for all the neighboring communities $C_j \in S(v)$. ■

Type D intra-edges

The following lemma considers the addition of type D intra-edges to the graph G.

Lemma 3. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let $S(v)$ be the set of all the neighboring communities of node v and $C_j \in S(v)$. Then node v does not want to switch to community $C_j \in S(v)$, if we add at most β_v^j new type D edges to community C_j where β_v^j is given by:*

$$\beta_v^j = \frac{2m(d_i(v) - d_j(v)) + d(v)(D(j) - D(i))}{2(d_j(v) - d_i(v) - d(v))}$$

and there is no other change to the graph.

Proof. Suppose we add $k \leq \beta_v^j$ new edges between pairs of nodes (u, w) where, $u \in C_j$, $w \in C_j$, and $C_j \in S(v)$. For the node $v \in C_i$, we have not added any edges incident on it, therefore, $d(v), d_i(v), d_j(v)$ will remain unchanged. Also, we have not added any edges to community C_i, hence, $D(i)$ will also remain unchanged. $D(j)$ is incremented by $2k$.

In the new graph G', when we compare the change in modularity for an isolated node v against community C_i and C_j, we have:

$$k \leq \frac{2m(d_i(v) - d_j(v)) + d(v)(D(j) - D(i))}{2(d_j(v) - d_i(v)\ d(v))}$$

$$\equiv 2k(d_j(v) - d_i(v) - d(v)) \leq 2d_i(v)m - d(v)D(i) - 2d_j(v)m + d(v)D(j)$$

$$\equiv 2d_j(v)m + 2d_j(v)k - d(v)D(j) - 2kd(v) \leq 2d_i(v)m + 2d_i(v)k - d(v)D(i)$$

Dividing by $2m + 2k$ on both sides, we obtain :

$$\frac{2d_j(v)m + 2d_j(v)k - d(v)D(j) - 2kd(v)}{(2m + 2k)} \leq \frac{2d_i(v)m + 2d_i(v)k - d(v)D(i)}{(2m + 2k)}$$

$$\equiv d_j(v) - \frac{d(v)(D(j) + 2k)}{(2m + 2k)} \leq d_i(v) - \frac{d(v)D(i)}{(2m + 2k)} \tag{3.5}$$

According to Equation (3.5), community C_i is still the best option for node v between C_i and C_j, hence, there is no change in community structure. So unless we add more than β_v^j edges, node v will not switch its community to C_j. Similarly, we can calculate β_v^j for all the neighboring communities $C_j \in S(v)$. ■

3.2.2 Inter community edge addition

Our objective in this subsection is to analyze the impact of inter community edge additions to community structure $\mathcal{C}$ in the graph G.

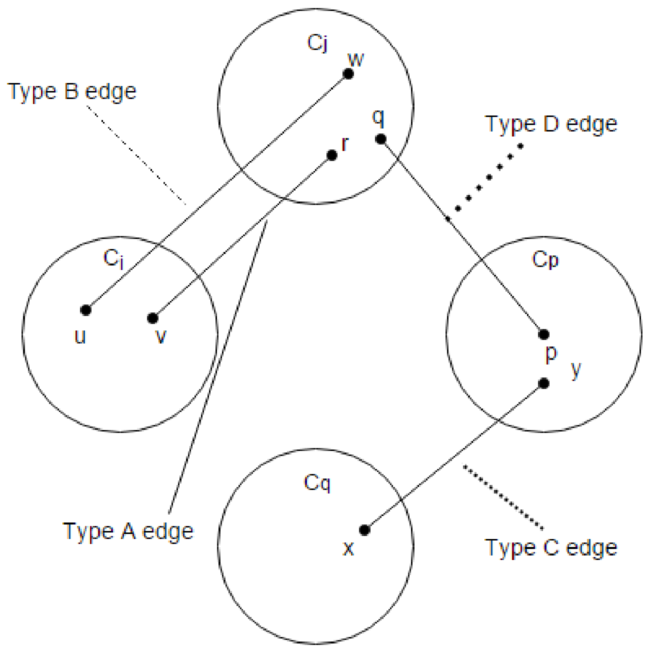


FIGURE 8: Types of inter edge additions

Consider node $v \in C_i$. There are four types of inter edge additions from the point of view of v, which are shown in Figure 8. We call an inter-edge a type A edge if it connects v to a node in any community except community C_i, C_j may or may not be a neighboring community of v, but C_p and C_q are not neighboring communities. We call it a type B edge if it connects two nodes from communities C_i and C_j, where C_j is neighboring community of node v. An inter-edge is a type C edge if it connects two nodes from communities C_p and C_q where C_p and C_q are not neighboring communities of node v. We call it a type D inter-edge if connects two vetices from communities C_j and C_P where C_j is neighboring community of node v and C_p is not. Our results for each type of inter edge addition in $\mathcal{C}$ are summarized in Table 2.

Type of edge	Proposition
A	Lemma 4
B	Lemma 5
C	Lemma 6
D	Lemma 7

TABLE 2: Propositions for different types of inter edge addition

Type A inter-edges

The following lemma considers the addition of type A inter-edges to the graph G.

Lemma 4. *Let $G = (V, E)$ be an undirected graph and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Then, v does not wish to switch to community C_j if we add at most $k \leq \gamma_v^j$ new type A inter-edges are added between v and nodes in C_j, where $C_j \neq C_i$, and γ_v^j is the solution to the quadratic*

$$\begin{aligned} k^2 + k(D(i) - 2d_i(v) + 2d_j(v) + 2m - d(v) - D(j)) \\ + d(v)(D(i) - D(j)) + 2m(d_j(v) - d_i(v)) \leq 0 \end{aligned}$$

and there is no other change to the graph.

Proof. Observe that adding k edges has the following effect. $D(j)$ and m are both incremented by $2k$ and $d_j(v)$ and $d(v)$ are incremented by k. On rearranging the quadratic above, we get:

$$\begin{aligned} D(i)k - 2d_i(v)k + 2kd_j(v) + 2km + k^2 - d(v)k - D(j)k \leq 2d_i(v)m \\ - d(v)D(i) - 2d_j(v)m + d(v)D(j) \end{aligned}$$

Dividing by $2m + 2k$ on both sides, we obtain:

$$d_j(v) + k - \frac{(d(v) + k)(D(j) + k)}{(2m + 2k)} \leq d_i(v) - \frac{(d(v) + k)D(i)}{(2m + 2k)} \tag{3.6}$$

Therefore, according to Equation 3.6, node v would not change its community to C_j as long as $k \leq \gamma_v^j$ edges are added between node v and nodes in C_j. Similarly, we can calculate γ_v^j for all the other communities apart from C_i. ■

Type B inter-edges

The following lemma considers the addition of type B inter-edges to the graph G.

Lemma 5. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let $S(v)$ be the set of all the neighboring communities of node v. Let C_j be a neighboring community of node v such that $C_j \in S(v)$. Then node v does not want to switch communities to C_j, if we add at most ζ_v^j new type B inter-edges to communities C_i and C_j where ζ_v^j is given by:*

$$\zeta_v^j = \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2(d_j(v) \ - \ d_i(v))}$$

and there is no other change to the graph.

Proof. Suppose we add $k \leq \zeta_v^j$ new edges between pairs of nodes (u, w) where, $u \in C_i$, $w \in C_j$, $u \neq v$ and $C_j \in S(v)$. Clearly, $d(v), d_i(v), d_j(v)$ will remain unchanged. However, $D(i)$ and $D(j)$ are both incremented by k as a result of addition of k new inter edges.

In the new graph G', when we the compare change in modularity for an isolated node v against community C_i and C_j, we have:

$$k \leq \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2(d_j(v) \ - \ d_i(v))}$$

$$\equiv 2k(d_j(v) \ - \ d_i(v)) \ \leq \ 2d_i(v)m \ - \ d(v)D(i) \ - \ 2d_j(v)m \ + \ d(v)D(j)$$

Subtracting $kd(v)$ from both sides, we obtain :

$$2d_j(v)m \ + \ 2d_j(v)k \ - \ d(v)D(j) \ - \ kd(v) \leq 2d_i(v)m \ + \ 2d_i(v)k \ - \ d(v)D(i) \ - \ kd(v)$$

Dividing by $2m + 2k$ on both sides, we get :

$$\frac{2d_j(v)m \ + \ 2d_j(v)k \ - \ d(v)D(j) \ - \ kd(v)}{(2m \ + \ 2k)} \leq \frac{2d_i(v)m \ + \ 2d_i(v)k \ - \ d(v)D(i) \ - \ kd(v)}{(2m \ + \ 2k)}$$

$$\equiv d_j(v) \ - \ \frac{d(v)(D(j) \ + \ k)}{(2m \ + \ 2k)} \ \leq \ d_i(v) \ - \ \frac{d(v)(D(i) \ + \ k)}{(2m \ + \ 2k)} \tag{3.7}$$

According to Equation (3.7), community C_i is still the best option for node v between C_i and C_j, hence, there is no change in community structure. So, unless we add more than ζ_v^j edges, node v will not switch its community to C_j. Similarly, we can calculate ζ_v^j for all the neighboring communities $C_j \in S(v)$. ■

Type C inter-edges

The following lemma considers the addition of type C inter-edges to the graph G.

Lemma 6. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let $S(v)$ be the set of all the neighboring communities of node v. Let C_j is a neighboring community and C_p and C_q are non neighboring communities. Then node v does not want to switch to community C_j, if we add at most η_v^j new type C inter-edges to communities C_p and C_q where η_v^j is given by:*

$$\eta_v^j = \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2(d_j(v) \ - \ d_i(v))}$$

and there is no other change to the graph.

Proof. The proof is exactly similar to Lemma 2. ■

Type D inter-edges

The following lemma considers the addition of type D inter-edges to the graph G.

Lemma 7. *Let $G = (V, E)$ be an undirected graph, and $\mathcal{C}$ be a locally optimal community structure for G. Fix $C_i \in \mathcal{C}$ and let $v \in C_i$ be an arbitrary node. Let $S(v)$ be the set of all the neighboring communities of node v. Let C_j be a neighboring community of node v and C_p is non neighboring community such that $C_j \in S(v)$, $C_p \notin S(v)$. Then, node v does not want to switch to community C_j, if we add at most τ_v^j new type D inter-edges to communities C_j and C_p, where τ_v^j is given by:*

$$\tau_v^j = \frac{2m(d_i(v) \ - \ d_j(v)) \ + \ d(v)(D(j) \ - \ D(i))}{2d_j(v) \ - \ 2d_i(v) \ - \ d(v)}$$

and there is no other change in the graph.

Proof. Suppose we add $k \leq \tau_v^j$ new edges between pairs of nodes (u, w) where, $u \in C_j$, $w \in C_p$, $C_p \notin S(v)$ and $C_j \in S(v)$. For the node $v \in C_i$, we have not added any edges incident on it, therefore, $d(v), d_i(v), d_j(v)$ will remain unchanged. Also we have not added any edges to community C_i hence $D(i)$ does not change. However, $D(j)$ is incremented by k as a result of addition of k new inter edges.

In the new graph G', when we will compare the change in modularity for an isolated node v against community C_i and C_j, we have:

$$k \leq \frac{2m(d_i(v) - d_j(v)) + d(v)(D(j) - D(i))}{2(d_j(v) - d_i(v))}$$

$$\equiv 2k(d_j(v) - d_i(v)) - d(v)k \leq 2d_i(v)m - d(v)D(i) - 2d_j(v)m + d(v)D(j)$$

$$2d_j(v)m + 2d_j(v)k - d(v)D(j) - kd(v) \leq 2d_i(v)m + 2d_i(v)k - d(v)D(i)$$

Dividing by $2m + 2k$ on both sides, we get :

$$d_j(v) - \frac{d(v)D(j)}{(2m + 2k)} \leq d_i(v) - \frac{d(v)(D(i) + k)}{(2m + 2k)} \tag{3.8}$$

According to Equation (3.8), community C_i is still the best option for node v between C_i and C_j, hence, there is no change in community structure. So, unless we add more than τ_v^j edges, node v will not switch its community to C_j. Similarly, we can calculate τ_v^j for all the neighboring communities $C_j \in S(v)$. ■

3.2.3 Merging of communities threshold

In the next claim, we calculate the minimum number of edges that are required to be added to the graph, in order to successfully merge the two communities.

Claim 2. *In graph $G = (V, E)$, we claim that the community C_j will not merge with the community C_i, if at most k new edges are added between pairs of nodes (v, w), where $v \in C_p$, $w \in C_q$, $p \in \mathcal{C}$, $q \in \mathcal{C}$, $i \in C$, $p \neq i$, $q \neq i$, $p \neq j$ and $q \neq j$*

$$k_{merge} = \min_{i|d_i(v)\neq 0} \frac{d(v)D(i) - 2md_i(v)}{2d_i(v)}$$

and there is no other change in the graph.

Proof. We consider a community as a supernode. Therefore, when we talk about merging of the two communities, we mean that the isolated supernodes change the stable community structure $\mathcal{C}$ of the graph G by joining some other supernodes. At the final stage of the algorithm, change in the modularity for every supernode is negative, hence the structure is stable.

After adding $k \leq k_{merge}$ new edges, we get the new graph $G'(V, E + 2k)$. For this new graph, let us fix supernode v, where $v \in C_j$. We have not added any edges to nodes present in community C_i and C_j, so $d(v), d_i(v), d_j(v), D(i)$ will remain unchanged. We have to check whether after adding k edges, the change in the modularity for supernode v will become positive. We have:

$$k < \frac{d(v)D(i) \; - \; 2md_i(v)}{2d_i(v)}$$

$$\equiv 2k < \frac{d(v)D(i)}{d_i(v)} \; - \; 2m$$

$$\equiv 2m + 2k < \frac{d(v)D(i)}{d_i(v)}$$

$$\equiv 2d_i(v)m \; + \; 2d_i(v)k < d(v)D(i)$$

$$\equiv d_i(v)(2m \; + \; 2k) < d(v)D(i)$$

$$\equiv d_i(v) < \frac{d(v)D(i)}{2m \; + \; 2k}$$

$$\equiv d_i(v) \; - \; \frac{d(v)D(i)}{(2m \; + \; 2k)} < 0 \tag{3.9}$$

L.H.S of Equation (3.9) is less than zero, hence the gain in modularity is negative and supernode v, i.e, community C_j will not merge with C_i. Therefore, in order to find the minimum number of edge additions, up to which the communities C_j will not merge with any other community, we choose the minimum value of all the possible values of k. ■

3.3 Edge Distribution Analysis algorithm for dynamic community detection

We have computed the minimum number of different types of intra and inter edges that would be need to be added to the network to cause v to switch communities. However, the analysis assumes that there is only one type of edge being added. Analyzing the effect of different types of edges being added to the same G is a much more complex problem and we have not attempted it in this thesis.

In this section, we propose a heuristic to decide when the community structure has changed significantly enough to justify running a community detection algorithm such as DSLM. While the community structure can change, even if a single threshold is crossed, in practice, the difference in the modularity is not very high. Thus, the key idea of our algorithm is to perform the community detection algorithm only if a *certain percentage* of nodes have crossed their thresholds. This percentage is determined empirically.

To summarize, we first calculate the distribution of edges in different communities. Next, we calculate the intra-edge and inter-edge thresholds κ_v^j and γ_v^j for every node. Subsequently, we use the edge distribution to determine the number of nodes whose thresholds have been crossed. Finally, if the percentage of nodes whose threshold is crossed is over a certain specified $MaxPercentCrossing$, then we call a community detection algorithm such as SLM or DSLM.

Algorithm 6 ThresholdPercentage($G_j, G_{j-1}, MaxPercentCrossing$)

```
1: GenerateThresholds($G_j, G_{j-1}.cluster$)
2: GenerateEdgeDistribution($G_j, G_{j-1}, G_{j-1}.cluster$)
3: $noOfNodesCrossingThr = 0$
4: for $v = 1$ to $V_j$ do
5:     if $intraDistribution[cluster[v]] > thrIntra[v]$ or
         ($interDistribution[v] > thrInter[v]$ and $!(intraPerNode[v] > 0)$) then
6:         $noOfNodesCrossingThr \leftarrow noOfNodesCrossingThr + 1$
7:     end if
8: $percentageCrossing = \frac{noOfNodesCrossingThr}{V_j} * 100$
9: if $percentageCrossing > MaxPercentCrossing$ then
10:    Dslm($G_j$)
11: else print no need to run DSLM Algorithm
12: end if
```

The ThresholdPercentage algorithm can be described as follows: In Line 1, we call GenerateThresholds algorithm which calculates different thresholds for every node. In line 2, we call GenerateEdgeDistribution algorithm which computes the intra vs. inter edge distribution for newly added edges. The **for** loop of lines 4-7 considers each node and checks whether intra threshold is violated. Basically the for loop checks whether the community that has node v gets more intra edges than $thrIntra[v]$. If yes, then we increase the number of nodes crossing the threshold by one. We also check whether

Algorithm 7 GenerateThresholds($G_j, G_{j-1}.cluster$)

```
1: ReadInput($G_j$) // Reads the graph and computes $V_j, E_j, d_v[]$
2: let $cluster[1..V_j]$, $thrIntra[1..V_j]$ and $thrInter[1..V_j]$ be new arrays
3: for $v = 1$ to $V_j$ do
4:     $cluster[v] = G_{j-1}.cluster[v]$
5: for $v = 1$ to $V_j$ do
6:     $thrIntra[v] = \infty$
7:     for $l = 1$ to $S(v)$ do
8:         if $thrIntra[v] > k_v^l$ then  //$k_v^l$ is # of intra edges needed for $v$ to switch to
9:             $thrIntra[v] = k_v^l$        //community $l$ as per Lemma 1
10:        end if
11:    $thrInter[v] = \infty$
12:    for $l = 1$ to $C$ do
13:        if $l \neq cluster[v]$ and $thrInter[v] > \gamma_v^l$ then  //From Lemma 4
14:            $thrInter[v] = \gamma_v^l$
15:        end if
16: return thrIntra and thrInter
```

inter threshold is violated or not. We modify the calculation of inter threshold for experimental purposes. In principle, $thrInter[v]$ computes minimum number of edges that are added from node v to a particular community C_j to *switch communities*. So for a large network, which has more than 0.1 million communities, storing such information for every node will slow down the algorithm. Also, according to Lemma 4 we assume that there is no other change in the graph, but, in reality, there might be other changes in the graph. If node v is the recipient of newly added intra edges, then we state that the node v will not switch communities. For similar reasons, we have only considered thresholds from Lemma 4 and 1. So, once we have calculated the total number of nodes changing their communities, in line 8, we calculate their percentage with respect to V_j. Based on the comparison of $MaxPercentCrossing$ and percentage of nodes crossing threshold, lines 9 to 12 decide whether to run the $DSLM$ algorithm or not. If the percentage of nodes, which are changing their communities, is less than $MaxPercentCrossing$, we save a lot of running time.

Algorithm GenerateThresholds calculates different thresholds given G_j and $G_{j-1}.cluster$.

Algorithm 8 GENERATEEDGEDISTRIBUTION(G_j, G_{j-1})

```
1: let $intraDistribution[1..G_j.cluster]$ and $interDistribution[1..V_j]$ be new arrays.
2: $intraPerNode[1..V_j]$ be new array.
3: $E_{diff} = E_j - E_{j-1}$ // Store distinct edges which are present in $E_j$ but not in $E_{j-1}$
4: for each edge $(u, v) \in E_{diff}$ do
5:     if $cluster[u] == cluster[v]$ then
6:         $intraDistribution[cluster[u]] \leftarrow intraDistribution[cluster[u]] + 1$
7:         $intraPerNode[u] \leftarrow intraPerNode[u] + 1$
8:         $intraPerNode[v] \leftarrow intraPerNode[v] + 1$
9:     else
10:        $interDistribution[u] \leftarrow interDistribution[u] + 1$
11:        $interDistribution[v] \leftarrow interDistribution[v] + 1$
12:    end if
13: return $intraDistribution$, $interDistribution$ and $intraPerNode$
```

Lines 1-2 reads the graphs G_j and computes total number of nodes V_j, total number of edges E_j and $d_v[1..V_j]$ stores the degree of every node. It also creates three new arrays, $thrIntra[]$ and $thrInter[]$ in which we store the computed thresholds. Lines 3-4 assign every node to community based on $G_{j-1}.cluster$. The **for** loop of the lines 5-15 considers every node and computes different thresholds. In line 6, we initialize the $thrIntra$ array to infinity. The **for** loop of the lines 7-10 calculates the intra threhshold using Lemma 1. $S(v)$ is the set of neighboring communities. We consider isolated node v against all its neighboring communities one by one and calculate the number of edges that need to be added to $v^{'}$s community so that the node v will switch communities. After the running of the for loop, we have computed the minimum threshold and stored it in $thrIntra[v]$. In line 11, we initialize the $thrInter$ array to infinity. Similarly Lines 12-15 compute inter threshold using Lemma 4. At this point, we do not restrict the calculation only to $S(v)$ but consider all the communities in the graph, apart from the community that has the node v. We compute the minimum of all these thresholds and assign it to $thrIntra[v]$ in line 14. Finally, we return intra and inter threshold arrays.

Algorithm GenerateEdgeDistribution works as follows. In Line 1, we create two new arrays in which we will store the number of intra-edges per community and number of inter edges per node for the newly added edges. In line 2, we create another array of size equal to the number of nodes. We want to mark the nodes which received intra edges. In line 3, we take the difference of E_j and E_{j-1} to store the newly added edge in E_{diff}. The **for** loop of lines 4-12 considers each edge from E_{diff} one by one. If both the endpoints of an edge belong to same cluster then we increase the count of intra-edges for that cluster and the count of intra-edges received for both the nodes. Otherwise, we increment the count of inter-edges for both the endpoints. Finally, we return the $intraDistribution$ and $interDistribution$ array containing the edge distribution per community and per node along with the $intraPerNode$ array.

3.4 Modularity Change Rate algorithm for Dynamic Community Detection

In this section, we describe another approach to determine for which G_i, the community structure has changed significantly enough to justify re-running a community detection algorithm such as DSLM. Consider the change in modularity as new edges are added to the graph, but using the same community structure. After a certain number of edges are added, we would obtain an improvement in modularity by computing a new community structure. It is to be expected that this improvement will increase as more and more edges are added to the graph. We hypothesize that the increase in improvement is actually a linear function of the number of edges added to graph. [2] To this end, we define the function,

$$\delta_{i,k} = (Q(G_i, C') - Q(G_i, C_0)) - (Q(G_k, C'') - Q(G_k, C_0))$$

where C' and C'' are the community structures obtained by running DLSM on G_i and G_k respectively, C_0 is some baseline community structure obtained by running DSLM on some earlier version of the graph, and for any graph G and any community structure C for it, $Q(G, C)$ is the modularity value for the graph G with respect to the community structure C.

We assume that a fixed percentage P_0 of the edges in G_0 is added to obtain G_i from G_{i-1} for every i. To obtain the rate of modularity change, we run DSLM on the graphs G_1 and G_2 and obtain $\delta_{1,0}$ and $\delta_{2,0}$ and divide their difference by P_0. Let δ_m be a value which is the minimum significant difference in modularity for two community structures for the same graph. That is, suppose we run DSLM on the graph G_0, and obtain a community structure C_0, and subsequently perform a number of edge additions. Once the difference between the modularity of the changed graph with respect to community structure C_0 and the modularity obtained by re-running DSLM on the

[2]This hypothesis is confirmed by our experiments as described in Section 5.4

new graph exceeds δ_m, we would like to run a community detection algorithm. The value of δ_m is determined empirically as described in Chapter 5. Using the predicted rate of modularity change, and the value of δ_m, we can predict in which phase to run the community detection algorithm.

Algorithm 9 DeltaModDslm(δ_m, G_j, P_0)

```
1: $j = 0$
2: $phase = 0$
3: while $true$
4:     if $acsj == phase$ or $j == phase + 1$ or $j == phase + 2$ then
5:         $G_j.mod = \text{DSLM}(G_j)$
6:     else print no need to run DSLM
7:     end if
8:     if $j == phase + 2$ then
9:         $phase = phase + \text{PhaseCalculation}(\delta_m, G_j, G_{j-1}, G_{j-2}, P_0)$
10:    end if
11:    $j \leftarrow j + 1$ do
```

Algorithm 10 PhaseCalculation($\delta_m, G_j, G_{j-1}, G_{j-2}, P_0$)

```
1: if $(\delta_{j-1,j-2} == 0$ and $\delta_{j,j-2} == 0)$ then
2:     return false
3: end if
4: $slope = \dfrac{\delta_{j,j-2} - \delta_{j-1,j-2}}{P_0}$
5: $c = \delta_{j-1,j-2} - slope * (P_0)$
6: $phase = \left\lceil \dfrac{\delta_m - c}{slope * (P_0)} \right\rceil$
7: return phase
```

Algorithm DeltaModDSLM works as follows: Since we want to find modularities for graphs G_0 onward, lines 1-2 initialize j and *phase* to zero. Line 3 indicates that we will always run DeltaModDSLM for every snapshot G_j. Lines 4-5 run the dynamic community detection algorithm by [50] for the snapshots $G_{phase}, G_{phase+1}$ and $G_{phase+2}$. As per Line 6, we do not run the DSLM algorithm when current snapshot does not satisfy constraint specified in line 4. Line 8-9 points out the right phase to run the algorithm by calling procedure **PhaseCalculation**.

The working of PhaseCalculation is as follows: Lines 1-2 evaluates whether the $\delta_{j,j-2}$ and $\delta_{j-1,j-2}$ for these pair of graphs G_{j-1}, G_{j-2} and G_j, G_{j-2} is zero. In such case, we predict that we will not run the DSLM algorithm in any of the future phases, and hence, return false.

On other hand, we predict that, the change in modularity grows linearly with the percentage of new edges added. Change in modularity is a linear function where x axis is the percentage of new edges added with respect to Graph G_0 and the y axis is the difference in modularity as $\delta_{phase,j}$. Lines 4-5 calculates the slope of this function using following pair of points $(P_j, \delta_{j-1,j-2}), (2 * P_j, \delta_{j,j-2})$. Using this slope and point $(P_j, \delta_{j-1,j-2})$, it further computes the constant c. Lines 6-7 rewrites the the function in terms of x and then divides it by P_j to evaluate correct phase in which the change in modularity will be greater than δ_m. In this chapter we proposed two new methods for dynamic community detection.

Chapter 4

Modeling Evolution of Communities

In the previous chapter, we proposed two algorithms for dynamic community detection. Both algorithms assume that we are given a sequence of graph $G_0, G_1, G_2 ... G_j$ where G_j is the result of adding edges to G_{j-1}. This begs the question of what the characteristics of these added edges are.

While there are many real-world data sets for social networks available in the public domain, such as Facebook, collaboration networks such as DBLP etc., it is not easy to find time stamped data on *evolving networks*. There are very few studies of such real evolving networks [27],[25].

Instead, in order to test our algorithms, we study models for evolving social networks in this chapter. Sociologists studying social networks have pointed out that the formation of new friendships/relationships are governed by a complex combination of factors including homophily, triadic closure, and focal closure [14] shown in Figure 9.

Homophily is the idea that people are more likely to form ties with others like them according to some attribute(eg. gender, race, or profession). Focal closure is the idea that if A and B have a common focus (eg. an activity, or workplace), they are more likely to form a tie. Triadic closure is the idea that if B and C share a common friend A, they are more likely to "close the triangle" and form a direct tie. Triadic closure can be further generalized to *cyclic closure*, which can be said to happen when the formation of a new tie creates a cycle of length > 2.

In this chapter, we propose three models for edge addition in social networks :Random, the EdgeDistance model(based on cyclic closure) and the Geometric Probability model(based on homophily)

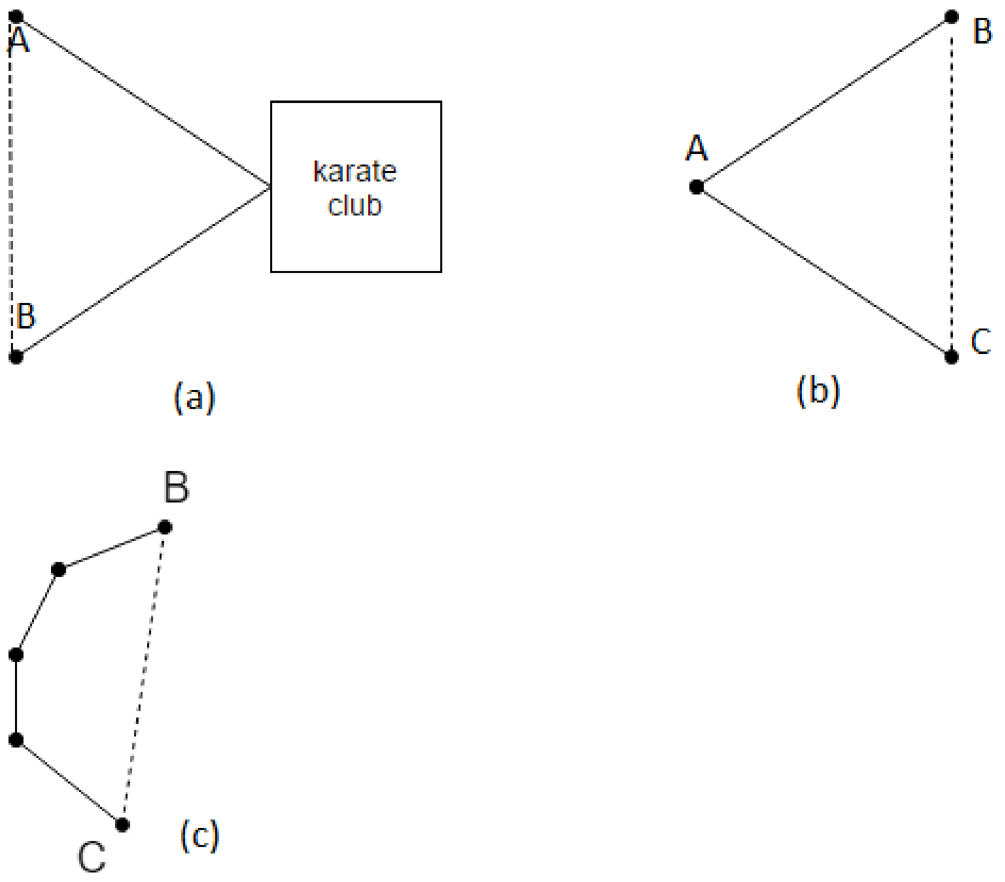


FIGURE 9: Effective factors for edge formation (a)Homophily (b)Triadic closure (c) Cyclic closure

4.1 Random Model

In social networks, edge addition is not in general random. Rather, in any network which possesses strong community structure, edges are added far from randomly. However, we also study the effect of random edge addition as a baseline model. The pseudocode is given in Algorithm 11.

Algorithm 11 RandomModel($G_j, percentageOfEdges$)

```
1: $nEdgesToAdd = \frac{E_j * percentageOfEdges}{100}$
2: let $newEdges[1..nEdgesToAdd]$ be a new list.
3: while $nEdgesToAdd > 0$ do
4:     $validEdge = false$
5:     $source = \text{Random}(1, V_j)$
6:     do
7:         $dest = \text{Random}(1, V_j)$
8:         for each $v \in G_j.Adj[source]$ do
9:             if $dest == v$ or $dest == source$ then
10:                $validEdge = false$
11:            end if
12:    while $(!validEdge)$
13:    if $validEdge$ then
14:        $nEdgesToAdd \leftarrow nEdgesToAdd - 1$
15:        if $source < dest$ then
16:            add $e(source, dest)$ to $newEdges$
17:        else
18:            add $e(dest, source)$ to $newEdges$
19:        end if
20:    end if
```

```
21: $newEdges = \text{Sort}(\text{newEdges})$
22: $E_{j+1} = \text{Merge}(E_j, newEdges)$
23: return $G_{j+1}$
```

The working of Random model is as follows. Lines 1-2 creates a new list in which we store the newly created edges and calculates the total number of edges to be added to the graph using the given percentage of edges. The while loop of lines 3-20 iterates as long as there are still new edges left to be added. Line 5 chooses the source node randomly between 1 to V_j. The do-while loop of lines 6-12 iterates as long as it does not find a valid edge for source node chosen in line 5. In line 7, we choose the destination node randomly between 1 to V_j. Then, we check whether the source and destination node are different and are not connected directly. Lines 13-19 are executed only if we find a valid edge to add to the network. We decrement the number of new edges to add by 1. We perform the necessary check that source should be smaller than destination and then add the new pair to the list. In line 21, we sort the list of newly added edges. In line 22, we merge the edges from E_j and newEdges list in a sorted manner. Finally, We return the newly added inter edges as a list.

4.2 EdgeDistance Model

As mentioned earlier, one phenomenon thought to influence the formation of new links in a social network is the phenomenon of cyclic closure- people are more likely to friends with friends of friends, and then their friends and so on. In this section, we propose the EdgeDistance model which is based [24] based on the idea that the probability of adding edge (u, v) is inversely proportional to the distance between them. For example, suppose node v is two hops away and node w is three hops away from node u, then the probability of adding a new edge from node u to node v is more than the probability of adding a new edge from node u to node w. We consider cycles of length at most 6; this means we create new edges between pair of nodes which are separated by at least two hops or at most five hops.

For hop distance i (where $2 \leq i \leq 5$), we define the probability of adding a new edge with hop distance i to be p_i where:

$$p_i = \frac{\frac{1}{i}}{\sum_{i=2}^{5} \frac{1}{i}}$$

Then clearly $\sum_{i=2}^{5} p_i = 1$, and the probability of picking an edge with hop distance i is inversely proportional to i. The pseudocode for adding new edges according to the EdgeDistance model is given in Algorithm 12.

Lines 1-2 creates new list in which we will store the newly created edges and calculates the total number of edges to be added to the graph using the given percentage of edges. The while loop of lines 3-42 iterates as long as there are still new edges left to be added. Lines 4-14 choose the distance between nodes of newly added edge based on probability which is inversely proportional to the distance between the nodes. Line 15 chooses the source node randomly between 1 to V_j. Line 16 runs Breadth First Search algorithm starting from source node and store the distance to every node from source. Line 18 initializes counter to 12. The while loop of lines 19-39 iterates as long as it does not find a valid edge from source node chosen in line 15 or until counter goes below zero. We are using the counter because it is possible that for the chosen distance, source does not have any neighbor with that distance. In line 20, we choose destination node randomly between 1 to V_j. Lines 21-24 determine whether source and node are separated by the distance chosen in previous steps. In case this is not true, we decrement the counter. Lines 26-31 checks whether the source node already has an edge to destination node. Lines 32-39 are executed only if we find a valid edge to add to the network. We decrement the number of new edges to add by 1. We perform the necessary check that source should be smaller than destination and then add the new pair to the list. In line 40, we sort the list of newly added edges. In line 41, we merge the edges from E_j and newEdges list in a sorted manner. Finally, We return the newly added inter edges as a list.

The running time of the algorithm can be improved further. In line 16, we know the currentEdgeDistance in advance, so we can perform Breadth First Search to that level only. Furthermore, we can store the nodes in that level in temporary array. In line 20, we are choosing destination node randomly from 1 to V_j. Instead of that, we can choose destination from the temporary array we created. Also, we might not need counter in this case. If all the nodes in this temporary array are invalid as destination node, then we choose new source node.

Algorithm 12 EDGEDISTANCE($G_j, percentageOfEdges$)

```
1: $nEdgesToAdd = \frac{E_j * percentageOfEdges}{100}$
2: let $newEdges[1..nEdgesToAdd]$ be a new list.
3: **while** $nEdgesToAdd > 0$ **do**
4:     $currentEdgeDistance = -1$
5:     $edgeDistance = \text{RANDOM}(1, 154)$
6:     **if** $edgeDistance \leq 60$ **then**
7:         $currentEdgeDistance = 2$
8:     **else if** $edgeDistance > 60$ and $edgeDistance \leq 100$ **then**
9:         $currentEdgeDistance = 3$
10:    **else if** $edgeDistance > 100$ and $edgeDistance \leq 130$ **then**
11:        $currentEdgeDistance = 4$
12:    **else if** $edgeDistance > 130$ and $edgeDistance \leq 154$ **then**
13:        $currentEdgeDistance = 5$
14:    **end if**
15:    $source = \text{RANDOM}(1, V_j)$
16:    MODIFIED-BFS($G_j, source$)
17:    $invalidEdge = true$
18:    $counter = 12$
```

```
    while $invalidEdge$ and $counter > 0$ do
        $dest = \textsc{Random}(1, V_j)$
        if $dest.d == currentEdgeDistance$ then
            $invalidEdge = false$
        else
            $counter \leftarrow counter - 1$
        end if
        if $(!invalidEdge)$ then
            for each edge $(u, v) \in newEdges$ do
                if $(u == source$ and $v == dest)$ or $(u == dest$ and $v == source)$ then
                    $invalidEdge = true$
                end if
        end if
        if $(!invalidEdge)$ then
            $nEdgesToAdd \leftarrow nEdgesToAdd - 1$
            if $source < dest$ then
                add $e(source, dest)$ to $newEdges$
            else
                add $e(dest, source)$ to $newEdges$
            end if
        end if
$newEdges = \textsc{Sort}(\textsc{newEdges})$
$E_{j+1} = \textsc{Merge}(E_j, newEdges)$
return $G_{j+1}$
```

4.3 Geometric Probability Model

In chapter 3, we classified edges into 2 categories intra-edges and inter-edges. The Geometric Probability model is based on the idea that people are more likely to form ties within their community than the outside community [25]. In the absence of data about other attributes of the individuals in the network, we can infer that if two nodes belong to the same community(as identified algorithmically), they share the same value for the "community attribute". Thus homophily can be said to dictate their bias towards tie formation with other members of the community. To this end, when choosing new edge, to add to the network, we pick an inter-edge with probability p and intra-edge with probability $1 - p$. We use the values $p = 0, 0.4, 0.6$ in our experiments, described in the next chapter.

We mention a small technical point here. We want to compare the performance of various algorithms like SLM, DSLM, Modularity-Change-Rate algorithm and Edge-Distribution-Analysis algorithm on dynamic evolving networks. For this reason, we want to use the same snapshots to compare the performance of these algorithms. For a given snapshot, resulting community structure might vary based on which algorithm is ran on graph. On that account, we chose to generate G_1, G_2... onwards from G_0. Hence, the new edges which are added in every snapshot are not with respect to $G_{previous}.cluster$ but based on $G_0.cluster$. By following this we make sure that the same snapshots are used to compare different algorithms. The pseudocode for our algorithm to implement this model is given in Algorithm 13.

The functioning of GeometricProbability algorithm is as follows. After initializing nInterEdges and nIntraEdges to zero in lines 1-2, line 3 calculates the total number of new edges to be added. Lines 4-10 generate random number between 0 to 1 and depending on upon p which is probability of an inter-edge, it assigns every edge either as inter or intra. Lines 11-13 calls procedure AddIntraEdges and AddInterEdges and store the result in two new lists. In line 14, we merge these two lists in a sorted order and return as a single list called newEdges. In line 15, we merge the edges from E_j and newEdges list in a sorted manner. We return the new snapshot G_{j+1} with newly added edges.

The procedure AddIntraEdges works as follows. Lines 1-4 store the graph G_j in adjacency list format and assign every node to community based on $G_0.cluster$. The while loop of lines 5-24 iterates as long as there are still new edges left to be added. Line 6 chooses the source node randomly between 1 to V_j. The do-while loop of lines 7-24 iterates as long as it does not find a valid edge from source node chosen in line 6. In line 9, we choose the destination node randomly from the given nodes V_j. Lines 10-13 checks

Algorithm 13 GeometricProbability($G_j, p, percentageOfEdges$)

```
1: $nInterEdges = 0$
2: $nIntraEdges = 0$
3: $nEdgesToAdd = \frac{E_j * percentageOfEdges}{100}$
4: for $v = 1$ to $nEdgesToAdd$ do
5:     $rand$ =Random$(0, 1)$
6:     if $rand < p$ then
7:         $nInterEdges \leftarrow nInterEdges + 1$
8:     else
9:         $nIntraEdges \leftarrow nIntraEdges + 1$
10:    end if
11: let $newEdgesIntra[1..nIntraEdges]$ and $newEdgesInter[1..nInterEdges]$ be new
    lists.
12: $newEdgesIntra$ = AddIntraEdges$(nIntraEdges, G_j, G_0)$
13: $newEdgesInter$ = AddInterEdges$(nInterEdges, G_j, G_0)$
14: $newEdges$ = MergeAndSort$(newEdgesIntra, newEdgesInter)$
15: $E_{j+1}$ = MergeAndSort$(E_j, newEdges)$
16: return $G_{j+1}$
```

whether the source node already has an edge to destination node or do they belong to same community or they are the same node. In any of such cases, we will discard this destination node and choose a new candidate. Lines 14-23 are executed only if we find a valid edge to add to the network. We add source and destination node to each others adjacency list, in order to avoid choosing the same pair again. We decrement the number of new intra edges to add by 1. In all the variants of the Louvain algorithm, the unweighted graph is stored in such fashion that all the edges are in ascending order based on nodes. Hence, we perform the necessary check that source should be smaller than destination and then add the new pair to list. We return the newly added intra edges as a list.

The procedure AddInterEdges works as follows. Lines 1-3 store the graph G_j in adjacency list format and creates a new adjacency list of nodes per cluster. Line 4-5 assign every node to community based on $G_0.cluster$. The while loop of lines 5-28

Algorithm 14 AddIntraEdges($nIntraEdges, G_j, G_0$)

```
1: let $newIntraEdges[1..nIntraEdges]$ be a new list.
2: store adjacency list representation of $G_j$ as $edgesPerNode[1..V_j][1..Adj[v]]$
3: for $v = 1$ to $V_0$ do
4:     $cluster[v] = G_0.cluster[v]$
5: while $nIntraEdges > 0$ do
6:     $source = \textsc{Random}(1, V_j)$
7:     do
8:         $validEdge = true$
9:         $dest = \textsc{Random}(1, V_j)$
10:        for each $v \in G_j.Adj[source]$ do
11:            if $dest == v$ or $cluster[source]! = cluster[dest]$ or $dest == source$ then
12:                $validEdge = false$
13:            end if
14:        if $validEdge$ then
15:            $nIntraEdges \leftarrow nIntraEdges - 1$
16:            add $source$ to $G_j.Adj[dest]$
17:            add $dest$ to $G_j.Adj[source]$
18:            if $source < dest$ then
19:                add $e(source, dest)$ to $newIntraEdges$
20:            else
21:                add $e(dest, source)$ to $newIntraEdges$
22:            end if
23:        end if
24:    while (!$validEdge$)
25: return $newIntraEdges$
```

Algorithm 15 AddInterEdges($nInterEdges, G_j, G_0$)

```
1: let $newInterEdges[1..nInterEdges]$ be a new list.
2: store adjacency list representation of $G_j$ as $edgesPerNode[1..Vj][1..Adj[v]]$
3: store adjacency list representation of $G_j.cluster\ as\ nNodesPerCluster[1..C][1..Adj[c]]$
4: for $v = 1$ to $V_0$ do
5:     $cluster[v] = G_0.cluster[v]$
6: while $nInterEdges > 0$ do
7:     $source$ = Random($1, V_j$)
8:     do
9:         do
10:            $destCluster$ = Random($1, C$)
11:        while $destCluster == cluster[source]$
12:        $randomDest$ = Random($1, nNodesPerCluster[source].length$)
13:        $dest = nNodesPerCluster[source][randomDest]$
14:        $validEdge = true$
15:        for each $v \in G_j.Adj[source]$ do
16:            if $dest == v$ or $dest == source$ then
17:                $validEdge = false$
18:            end if
19:        if $validEdge$ then
20:            $nInterEdges \leftarrow nInterEdges - 1$
21:            add $source$ to $G_j.Adj[dest]$ and add $dest$ to $G_j.Adj[source]$
22:            if $source < dest$ then
23:                add $e(source, dest)$ to $newInterEdges$
24:            else
25:                add $e(dest, source)$ to $newInterEdges$
26:            end if
27:        end if
28:    while (!$validEdge$)
29: return $newInterEdges$
```

iterates as long as there are still new inter edges left to be added. Line 7 chooses the source node randomly between 1 to V_j. The do-while loop of lines 8-28 iterates as long as it does not find a valid edge from source node, chosen in line 6. The do-while loop of lines 9-11 chooses one cluster randomly apart from the cluster to which the source node belongs. Lines 12-13 draws destination node randomly from the adjacency list of cluster chosen in line 10. Lines 15-18 checks whether the source node already has an edge to destination node or whether they are the same node. In any of such cases, we will discard this destination node and choose a new candidate. Lines 19-27 are executed only if we find a valid edge to add to the network. We add source and destination node to each others adjacency list, in order to avoid choosing the same pair again. We decrement the number of new inter edges to add by 1. We perform the necessary check that source should be smaller than destination and then add the new pair to the list. We return the newly added inter edges as a list.

Chapter 5

Empirical Analysis

In this chapter, we describe our experimental results. First we describe the characteristics of data sets such as percentage of edges added, number of intra vs. inter edges added, edge to node ratio etc, and the computing environment. Then we explain our performance metrics for the experiments. Next, we present the analysis of edge addition models and results of Modularity-Change-Rate algorithm. After this we explain the problems faced while running the experiments for Edge-Distribution-Analysis algorithm and its results. We have combined the results of different phases and provided a comparative analysis over the time, modularity as well as the accuracy for the proposed and existing algorithms.

5.1 Experimental Setup

In this section we describe the standard benchmark networks we have used, and mention the computational environment for our experiments.

5.1.1 Data Sets

There are many standard benchmark networks available for research in social network analysis. Most studies focus on the social networks, biological networks and citation networks. The performance of the proposed algorithms was tested on five small and medium sized networks and two large networks. The following networks are considered:

- *Football network*: Models American football games between NCAA Div IA colleges in Fall 2000. Each node represents a football team, which belongs to a specific conference(Big Ten, Conference USA, Pac-10, etc.). An edge between two nodes $v1$ and $v2$ means that the two teams played each other [40].

- *Email network*: This is the email communication network at the University Rovira i Virgili in Tarragona in the south of Catalonia in Spain. Nodes are users and each edge represents that at least one email was sent. The emails during the first three months of 2002 are considered. The direction of emails or the number of emails are not stored [2], [21].

- *Facebook network*: This network consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using the Facebook app. The dataset includes node features (profiles), circles, and ego networks.

  Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. Also, while the feature vectors from this dataset have been provided, the interpretation of those features has been obscured. For instance, where the original dataset may have contained a feature "political=Democratic Party", the new data would simply contain "political=anonymized feature 1". Thus, using the anonymized data, it is possible to determine whether two users have the same political affiliations, but not what their individual political affiliations represent [28].

- *PGP network*: This is the interaction network of users of the Pretty Good Privacy (PGP) algorithm. The network contains only the giant connected component of the network [1], [11].

- *Condmat2003 network*: The network Cond-mat-2003 is an updated version of Cond-mat, the collaboration network of scientists posting preprints on the condensed matter archive at www.arxiv.org. This version is based on preprints posted to the archive between January 1, 1995 and June 30, 2003. The largest component of this network, which contains 27519 scientists, has been used by several authors as a testbed for community-finding algorithms for large networks [33].

- *DBLP network*: This is the collaboration graph of authors of scientific papers from DBLP computer science bibliography. An edge between two authors represents a common publication. Edges are annotated with the date of the publication [49].

- *Livejournal network*: LiveJournal is a free online community with almost 10 million members; a significant fraction of these members are highly active. (For example, roughly 300,000 update their content in any given 24-hour period.) LiveJournal allows members to maintain journals, individual and group blogs, and it allows people to declare which other members are their friends [49].

Table 3 displays some salient characteristics of the networks such as name, number of nodes and edges, edges to nodes ratio and modularity of the network. Modularity of the network is obtained by running the SLM algorithm with 10 random starts and we run 10 iterations for every random start. The best result obtained from one of these 100 iterations is reported. The only exception to this is the Livejournal network. Due to its huge size, we run a single random start with 10 iterations.

Network	*#nodes*	*#edges*	E/V	*Modularity*
Football	115	613	5.330434782	0.6042
Email	1133	5411	4.775816417	0.5822
Facebook	4039	88234	20.47667672	0.8381
PGP	10680	24316	2.276779026	0.8860
Condmat2003	27519	116181	4.221846724	0.7695
DBLP	425957	1049866	2.464722965	0.8366
Livejournal	4036538	34681189	8.591815313	0.7707

Table 3: Characteristics of benchmark networks

5.1.2 Computational environment

Computations were made on the supercomputer Briarée from Concordia University, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), the ministére de l'Économie, de la science et de l'innovation du Québec (MESI) and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT). All calculations reported below were performed on a system with an Intel Xeon X5650 Westmere (2.67 GHz) and 48 GB internal memory. The code is written in Java and is available on https://github.com/tejaspuranik/EDA-MCR.git.

5.2 Analysis of edge distribution

In this section, we study the distribution of edges being added according to the different edge addition models described in chapter 4, to the 7 networks described in Subsection 5.1.1.

5.2.1 Generating evolving networks

To run our experiments, we use each of the real world networks described above as network G_0. We then add edges according to the 3 edge addition models described in Chapter 4. We take 5 snapshots of the network, each snapshot after 2% of the number of

edges in the original graph G_0 has been added. That is, $|E_{i+1}| = 1.02|E_0|$ for $0 \leq i \leq 4$. We call each of the five rounds of edge additions a *phase*.
For each G_0, and for each edge addition model, we run 10 different random runs, so that we obtain 10 different versions of $G_i(1 \leq 5)$. Thus, at the end of i phases, we have 10 different versions of G_i, each representing a different possible evolution of the graph G_0 according to a specific model of edge additions. All our results for any given phase are an average over the 10 graphs.

5.2.2 Distribution of edges for the Geometric Probability Model

For each of the seven networks, we generated 5 phases using the Geometric Probability model with inter-edge probability $p = 0, 0.4$ and 0.6. We computed the intra vs. inter distribution of newly added edges in each phase. We have taken the average of the results obtained from 10 graphs for every phase. In the case of the Email network, the inter probability = 0.4 gives, on an average 43.6 inter edges are added out of 110 newly added edges. On the other hand, 66.4 intra edges are added in phase 1. We observe similar results for the other phases. $\frac{43.6}{110} \approx 0.4$, which tells us that our algorithm has generated the desired result. We also confirm this looking at the results of the other networks.

5.2.3 Distribution of edges for the EdgeDistance model

In the case of the EdgeDistance, we computed the intra vs. inter edge distributions and plotted graphs for each network. The x-axis represents different phases for different models and y-axis represents a percentage of intra/inter edges for 10 graphs in each phase. Based on Figures 10 to 16, we observe that the EdgeDistance model does not generate the edges as we expected it. We expected that the majority of the edges being added would be intra but from Figures 10, 11, 12, 13, 14, 15, and 16 we observe that the percentage of intra-edges, is on an average 7, 18, 20, 15, 8, 35 and 29 respectively which is shown in Table 4. So in reality, the majority of the edges added are inter edges. In fact, along with the defined probability which is inversely proportional to distance there should be another conditional probability for inter edges with lower value. For example, consider node u is source node which is two hops away from node v and node w. Also lets assume that the nodes u and v belong to same community while node v and w belong to different communities. Then in such case, the probability that node u will form a direct edge with node v should be more than the probability of forming a direct edge with node w. Our model does not take care of this. Also from Table 4, we observe that its hard to find direct relation between modularity and percentage of intra-edges added.

Network	*Modularity*	% of intra edges added
Email	0.5822	18
Football	0.6042	7
Condmat2003	0.7695	8
Livejournal	0.7707	29
DBLP	0.8366	35
Facebook	0.8381	20
PGP	0.8860	15

Table 4: Edge distribution analysis over modularity for EdgeDistance model

Network	*Modularity*	% of intra edges added
Email	0.5822	11
Football	0.6042	5
Condmat2003	0.7695	3
Livejournal	0.7707	4
DBLP	0.8366	1
Facebook	0.8381	9
PGP	0.8860	2

Table 5: Edge distribution analysis over modularity for Random model

5.2.4 Distribution of edges for the Random model

We can observe that in Figures 10, 11, 12, 13, 14, 15, and 16 percentage of intra-edges, is on average 5, 11, 9, 2, 3, 1 and 4 which is shown in Table 5. We can confirm that on adding edges randomly to network, 90 percent of the edges will be inter community edges. As a deduction the real model which portrays the evolution of social networks will have very less percentage of inter community edges as compared to intra community edges. Also from Table 5, we observe that its hard to find direct relation between modularity and percentage of intra-edges added.

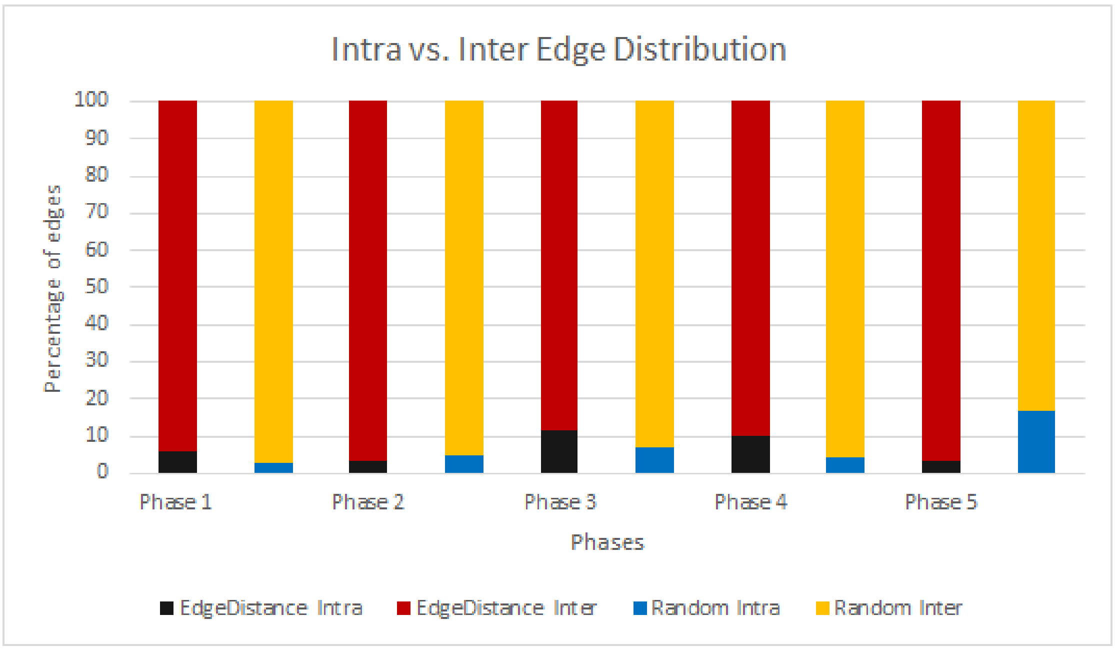


FIGURE 10: Football network edge distribution

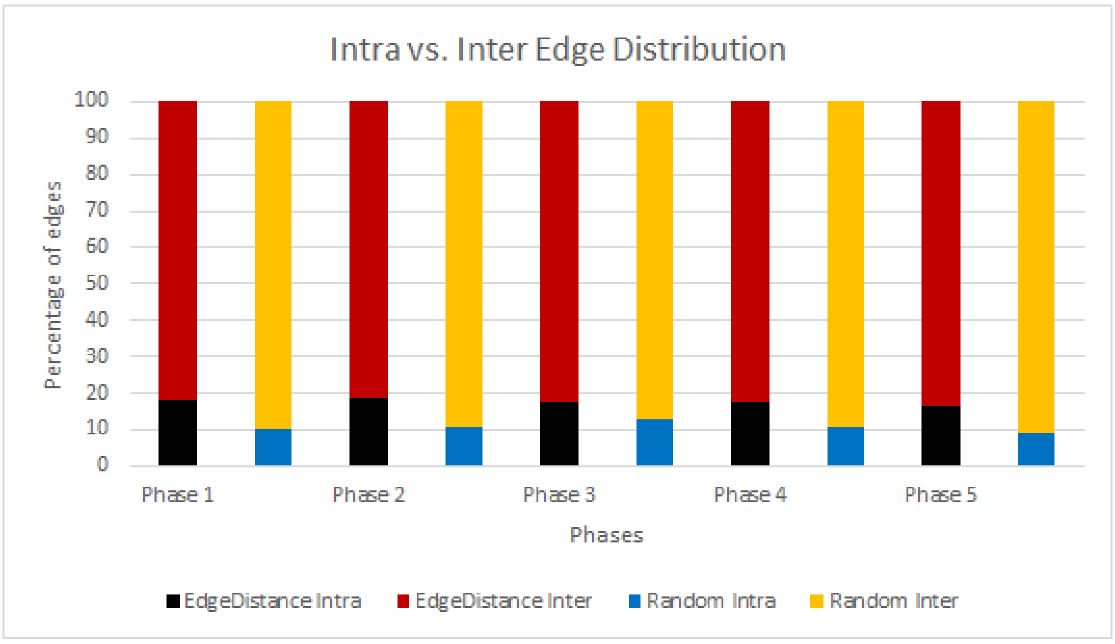


FIGURE 11: Email network edge distribution

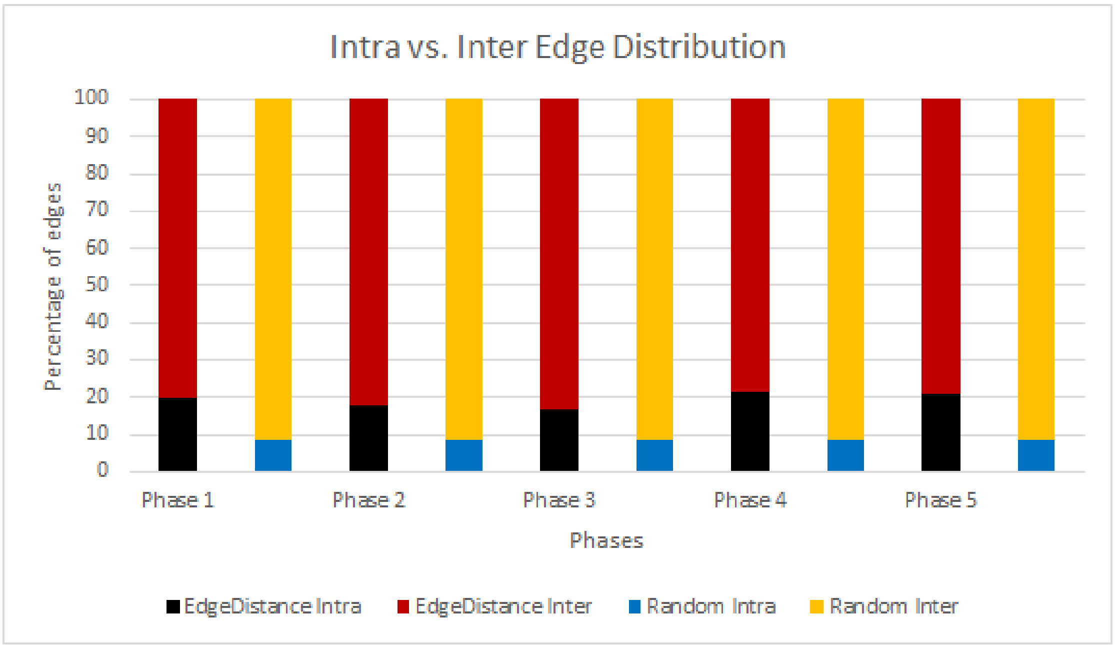


FIGURE 12: Facebook network edge distribution

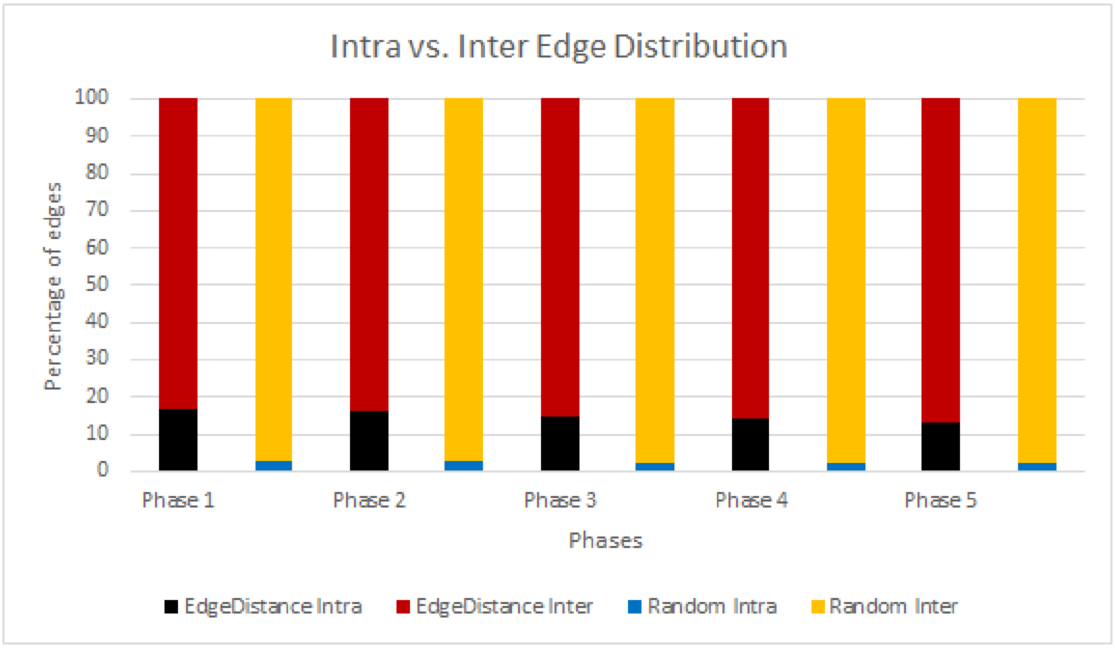


FIGURE 13: PGP network edge distribution

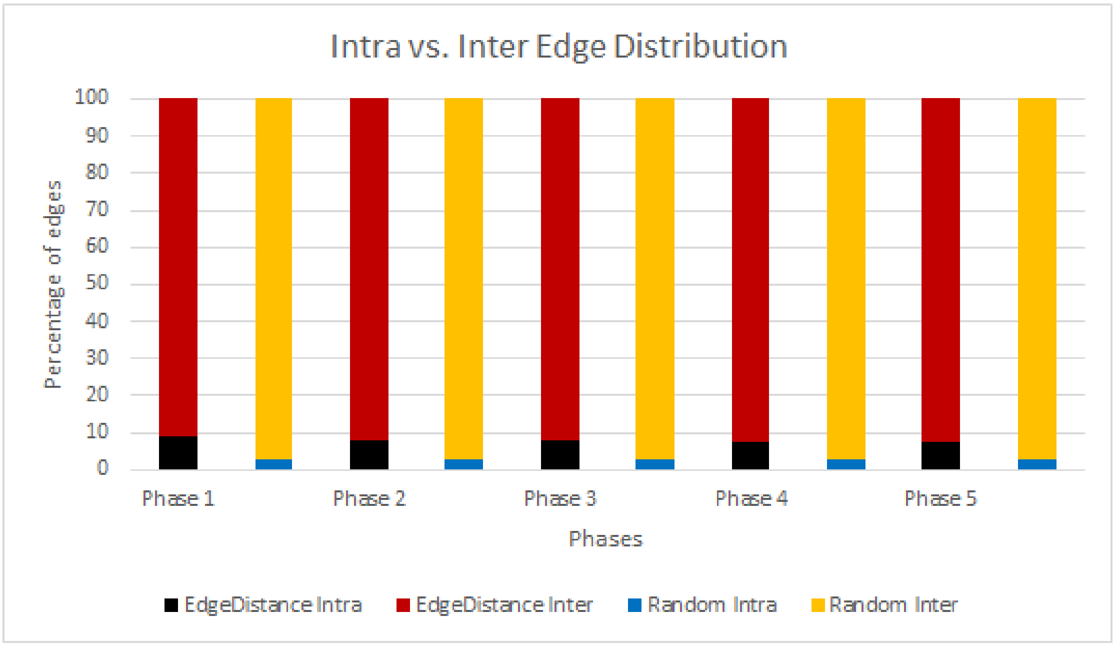


FIGURE 14: Condmat network edge distribution

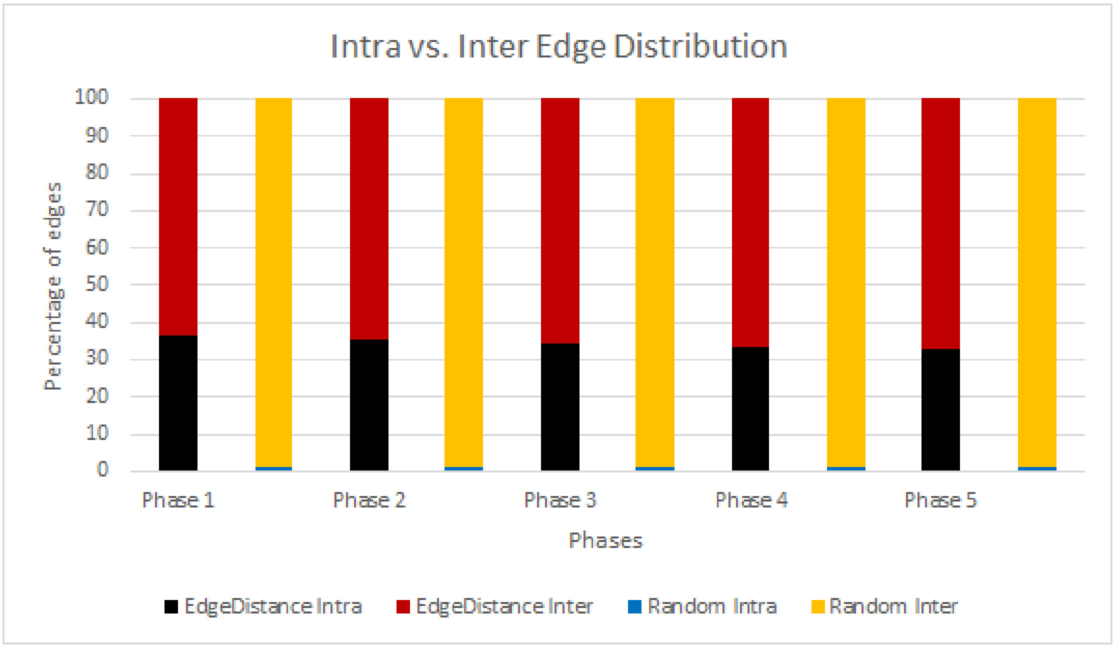


FIGURE 15: DBLP network edge distribution

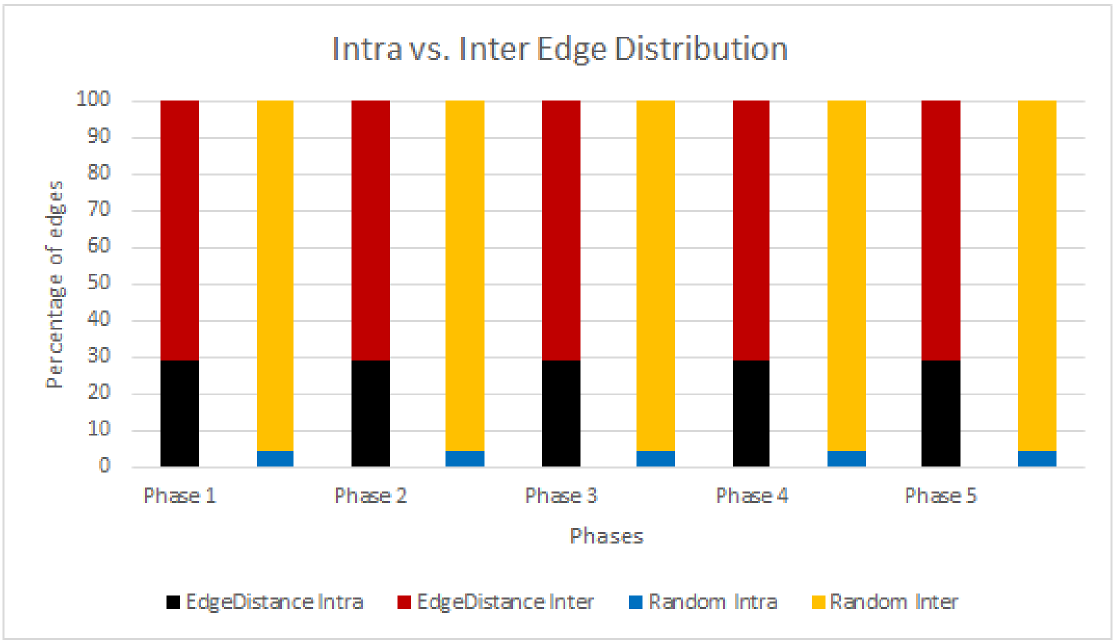


FIGURE 16: Livejournal network edge distribution

5.3 Performance evaluation approach

The goal of the community detection algorithms described in Chapter 3 is to determine for each snapshot G_i whether or not to update the community structure by running a community detection algorithm such as DSLM. Ideally, we should not update if the change in modularity is not significant, but we should indeed update if the change in modularity is significant. We choose $\delta_m = 0.005$ to be the minimum change of modularity necessitating an update of community structure.

In order to know whether such an update is really needed in our given data sets, we run DSLM after every phase for every graph. We denote by $Q(G_i, C_i)$ the modularity of graph G_i with respect to the new community structure computed by DSLM. We denote by $Q(G_i, C_0)$ the modularity of the graph G_i with respect to the *old* community structure computed by DSLM for G_0. In other words, $Q(G_i, C_0)$ is the modularity obtained by *not* running an update for G_i. Finally we define δ_{i0} to be the *observed* actual difference in modularity obtained by running an update for G_i versus not running an update for G_i and for other snapshots upto G_i. That is,

$$\delta_{i0} = Q(G_i, C_i) - Q(G_i, C_0)$$

We conclude that an update to the community structure is required if and only if $\delta_{i0} > \delta_m$.

In Figures 17 to 23, we plot the value of δ_{i0} for all our data sets. The value of δ_m is shown in blue. Observe that for Football, Email, Facebook and Condmat networks,

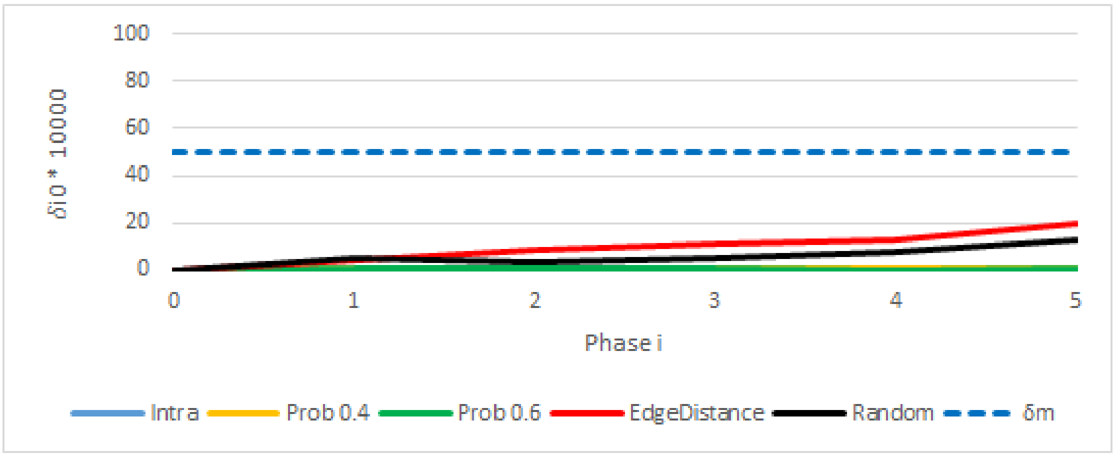


FIGURE 17: Difference in modularity with DSLM and without DSLM for Football n/w

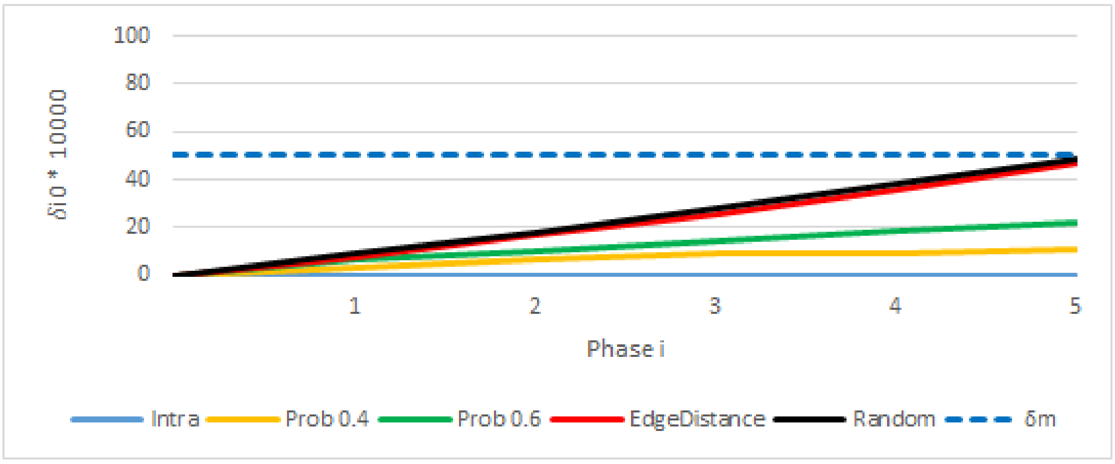


FIGURE 18: Difference in modularity with DSLM and without DSLM for Email n/w

the value δ_{i0} stays below δ_m for all phases. In other words, there is no need to update the community structure for any of the phases. On the other hand, for PGP, DBLP and LiveJournal, the modularity changes sufficiently for some of the edge addition models and some phases for an update to be desirable.

We summarize the desired update decisions in Table 6. A notation of Y implies that an update is required and a notation of N implies that an update is not required. Observe that for the PGP network, in the 5th phase, we say an update is not required even though $\delta_{50} > \delta_m$ in this case. This is because if an update is performed in the fourth phase, then the difference in modularity $\delta 54$ is expected to be less than δ_m. The same reasoning is applied to derive the ideal update decision for the 4^{th} and 5^{th} phase and 5^{th} phase of Random model of Livejournal network.

In the following sections, we analyze the results of our algorithms and compare them with the desired update decisions in Table 6.

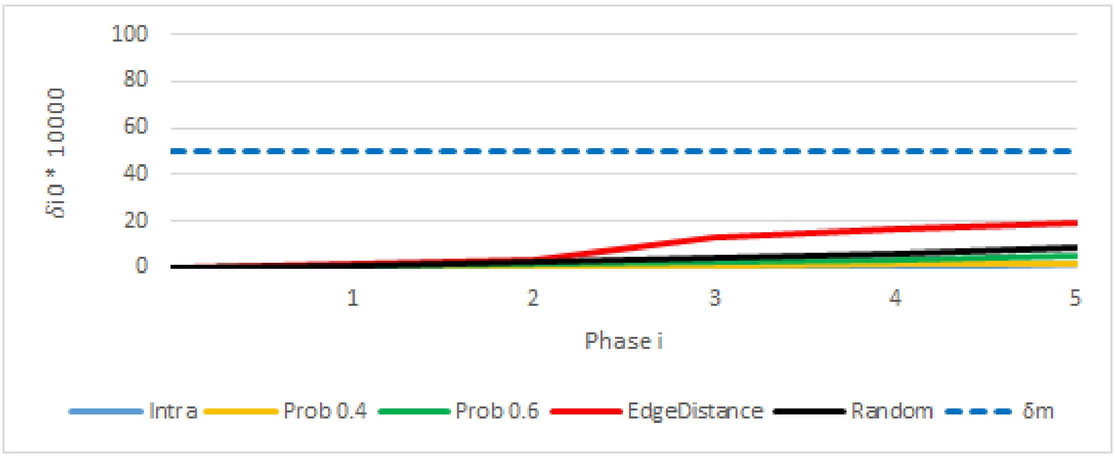


FIGURE 19: Difference in modularity with DSLM and without DSLM for Facebook n/w

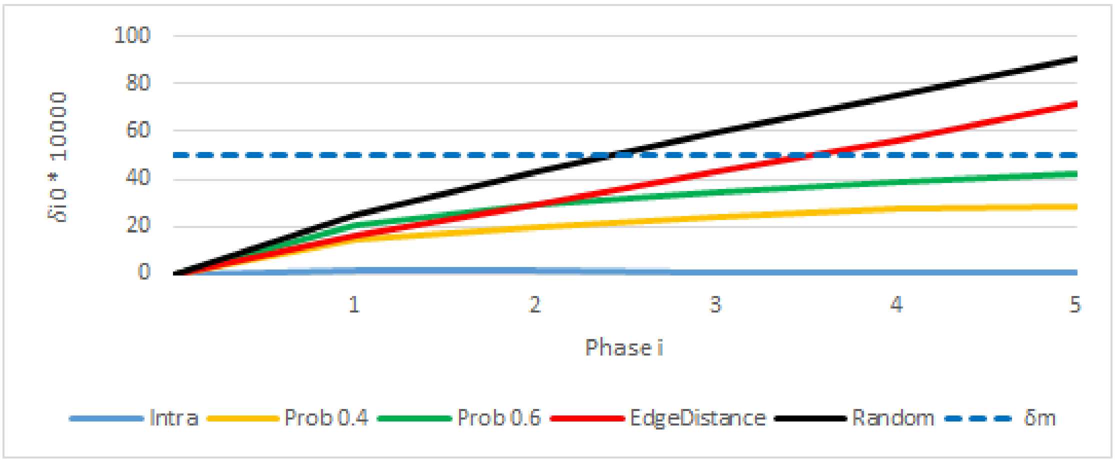


FIGURE 20: Difference in modularity with DSLM and without DSLM for PGP n/w

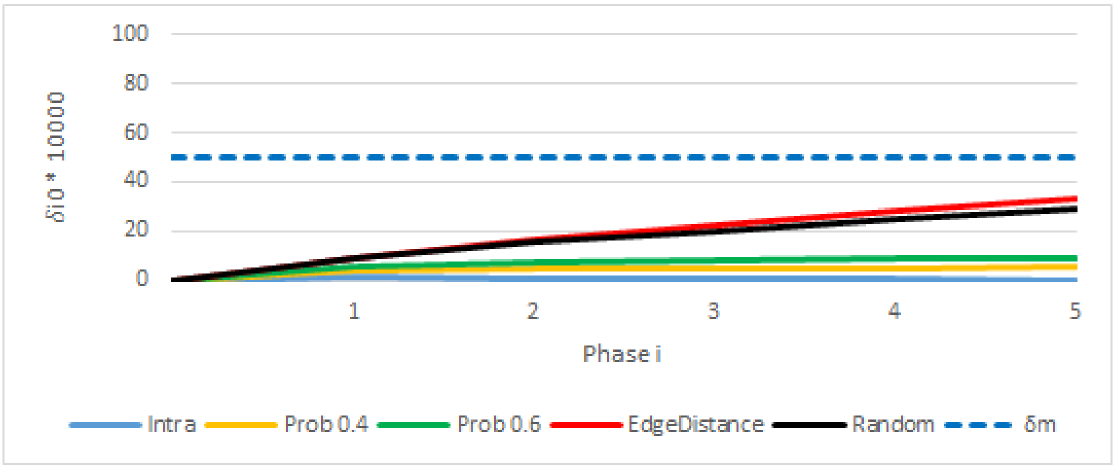


FIGURE 21: Difference in modularity with DSLM and without DSLM for Condmat n/w

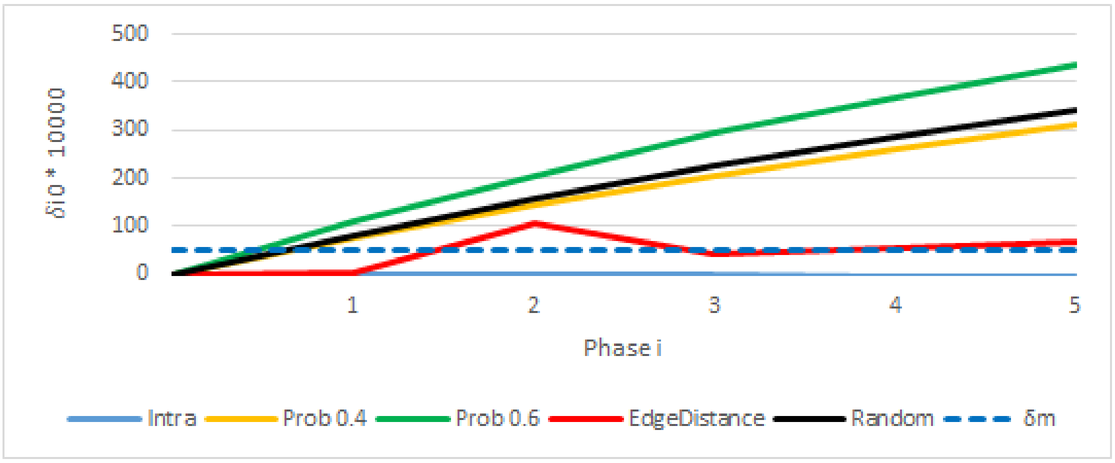


FIGURE 22: Difference in modularity with DSLM and without DSLM for DBLP n/w

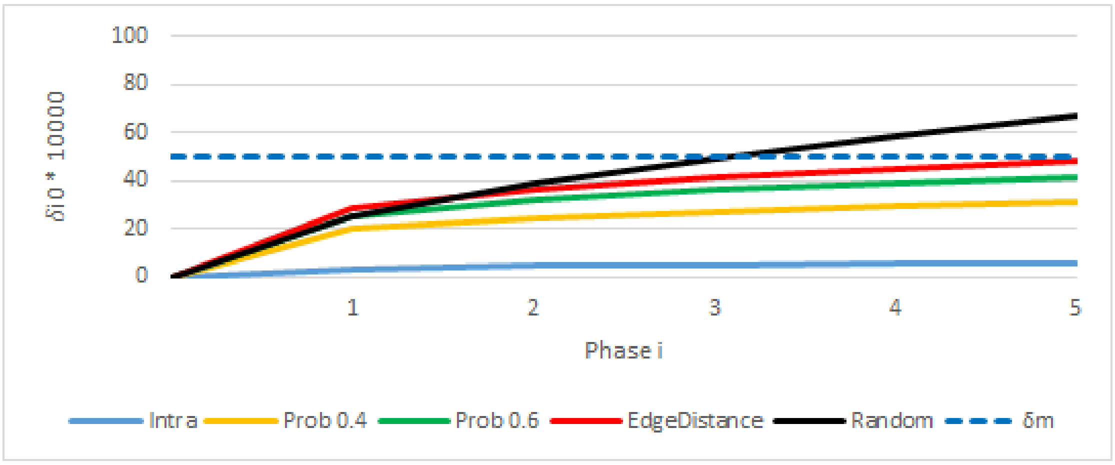


FIGURE 23: Difference in modularity with DSLM and without DSLM for Livejournal n/w

Network	Model	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
Football	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	N	N
	Random	N	N	N	N	N
Email	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	N	N
	Random	N	N	N	N	N
Facebook	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	N	N
	Random	N	N	N	N	N
PGP	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	Y	N
	Random	N	N	Y	N	N
Condmat	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	N	N
	Random	N	N	N	N	N
DBLP	P=0	N	N	N	N	N
	P=0.4	Y	Y	Y	Y	Y
	P=0.6	Y	Y	Y	Y	Y
	EdgeDistance	N	Y	N	Y	N
	Random	Y	Y	Y	Y	Y
Livejournal	P=0	N	N	N	N	N
	P=0.4	N	N	N	N	N
	P=0.6	N	N	N	N	N
	EdgeDistance	N	N	N	N	N
	Random	N	N	N	Y	N

Table 6: Ideal phases to update community structure

5.4 Modularity-Change-Rate algorithm

In this section, we report the results of running our modularity-based algorithm for our 7 base networks. Recall that according to this algorithm, we run the DSLM algorithm for the first two phases, and based on the calculation of the change in modularity, we predict the next phase in which to update the community structure. The results are shown in Table 7. In all the intervening phases, we obtain an advantage in terms of running time by not updating the community structure while being reasonably certain that it has not changed significantly.

On that note, in case of an $P = 0$ edge addition model, we do not run DSLM algorithm at all after the first 2 phases as the $\delta_{1,0}$ and $\delta_{2,0}$ are zero. Hence, the slope of the line and constant c from algorithm 10 is also zero. When $p = 0.4$ or $p = 0.6$, we get an advantage of not running the algorihthm for more than 100 phases in case of the Football network and Facebook network. Also in the Condmat2003 network we get a significant advantage of not running until 77^{th} and 29^{th} phases respectively. In the Email network, we do not run DSLM until 16^{th} and 12^{th} phases respectively. In the PGP network, we are saving time by not running in a few phases. However, the DBLP network is an exception to this. We do not get any advantage; rather, on top of running the DSLM algorithm, we are running our algorithm, hence, it increases the running time. In the Livejournal network, we are getting a significant advantage by not performing an update for a few phases. The Livejournal network is very large and the running time of DSLM, even after using memory clusters, is not small. So even if we save a few phases, we are saving a lot of time. For EdgeDistance and Random model also our algorithm gives good results but as these models have majority of inter-community edges, $\delta_{i,j}$ crosses 0.005 faster than other edge addition strategies. As a result, we do not get an advantage like we got in case of $p - 0$ or $p = 0.4$ model but still we get some advantage. For example, in the case of the Facebook network we do not run DSLM until 36^{th} and 27^{th} phase but for large networks like DBLP and Livejournal we do not get much advantage.

For the first 5 phases of edge additions, Table 8 summarizes the positive and negative errors made by our algorithm as compared to the ideal update decisions shown in Table 6. FP indicates a false positive error, that is, our algorithm suggests an update but the ideal decision would be not to update. FN indicates a false negative; our algorithm suggests not to update, but a ideal decision would be to update. TP indicates a true positive; our algorithm suggests to update and so does the ideal decision. TN indicates a true negative, that is, our algorithm suggests not to run the update and so does the ideal decision. We remind the reader that in the first two phases, our algorithm always runs, resulting in many false positive decisions, but in most cases, this would result in a huge gain in time far beyond the 5 phases we have shown here.

Table 9 summarizes total number of TP, TN, FP, FN and accuracy of the Modularity-Change-Rate algorithm for each network. Each network has 5 phases and 5 different edge addition models. We observe that an accuracy for each network varies in between 36 to 60. We have low accuracy rate due to the fact that there are lot of FPs for the first two phases.

Network	Model	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Phase 6
Football	P=0	1	2	N/A			
	P=0.4	1	2	125	126	127	
	P=0.6	1	2	97	98	99	
	EdgeDistance	1	2	21	22	23	
	Random	1	2	11	12	13	
Email	P=0	1	2				
	P=0.4	1	2	16	17	18	
	P=0.6	1	2	12	13	14	
	EdgeDistance	1	2	6	7	8	
	Random	1	2	6	7	8	
Facebook	P=0	1	2	N/A			
	P=0.4	1	2	101	102	103	
	P=0.6	1	2	99	100	101	
	EdgeDistance	1	2	36	37	38	
	Random	1	2	4	5	6	
PGP	P=0	1	2	N/A			
	P=0.4	1	2	7	8	9	
	P=0.6	1	2	5	6	7	
	EdgeDistance	1	2	3	4	5	
	Random	1	2	4	5	6	
Condmat	P=0	1	2	N/A			
	P=0.4	1	2	77	78	79	
	P=0.6	1	2	29	30	31	
	EdgeDistance	1	2	7	8	9	
	Random	1	2	7	8	9	
DBLP	P=0	1	2	N/A			
	P=0.4	1	2	3	4	5	6
	P=0.6	1	2	3	4	5	6
	EdgeDistance	1	2	3	4	5	6
	Random	1	2	3	4	5	6
Livejournal	P=0	1	2	29	30	31	
	P=0.4	1	2	8	9	10	
	P=0.6	1	2	5	6	7	
	EdgeDistance	1	2	4	5	6	
	Random	1	2	3	4	5	6

Table 7: Modularity-Change-Rate algorithm results

Network	Model	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
Football	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	TN
	EdgeDistance	FP	FP	TN	TN	TN
	Random	FP	FP	TN	TN	TN
Email	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	TN
	EdgeDistance	FP	FP	TN	TN	TN
	Random	FP	FP	TN	TN	TN
Facebook	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	TN
	EdgeDistance	FP	FP	TN	TN	TN
	Random	FP	FP	TN	TN	TN
PGP	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	FP
	EdgeDistance	FP	FP	FP	TP	FP
	Random	FP	FP	FN	FP	FP
Condmat	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	TN
	EdgeDistance	FP	FP	TN	TN	TN
	Random	FP	FP	TN	TN	TN
DBLP	P=0	FP	FP	TN	TN	TN
	P=0.4	TP	TP	TP	TP	TP
	P=0.6	TP	TP	TP	TP	TP
	EdgeDistance	FN	TP	FN	TP	FN
	Random	TP	TP	TP	TP	TP
Livejournal	P=0	FP	FP	TN	TN	TN
	P=0.4	FP	FP	TN	TN	TN
	P=0.6	FP	FP	TN	TN	FP
	EdgeDistance	FP	FP	TN	FP	FP
	Random	FP	FP	FP	TP	FP

TABLE 8: Modularity-Change-Rate algorithm false positives and false negatives

			Predicted		
			T	N	Accuracy(%)
Football	Actual	T	0	0	60
		N	10	15	
Email	Actual	T	0	0	60
		N	10	15	
Facebook	Actual	T	0	0	60
		N	10	15	
PGP	Actual	T	1	1	36
		N	15	8	
Condmat	Actual	T	0	0	60
		N	10	15	
DBLP	Actual	T	17	3	80
		N	2	3	
Livejournal	Actual	T	1	0	40
		N	15	9	

Table 9: Confusion Matrix and Accuracy for Modularity-Change-Rate algorithm

5.5 Edge-Distribution-Analysis algorithm

In this section, we present the results of our Edge-Distribution-Analysis algorithm . To run the ThresholdPercentage algorithm, we need to know the correct value of *MaxPercentCrossing*. In order to find the correct value of *MaxPercentCrossing*, we run the lines 1-8 of ThresholdPercentage for every snapshot and calculate the percentage of nodes crossing their thresholds. We made the following modification while running this code: We use the community structure of graph G_0 in all the snapshots instead of $G_{previous}$. The results of these experiments are displayed in following figures. For Figures 24, 25, 26, 27, 28, 29, and 30, we have plotted the phases on x-axis and percentage of nodes on y-axis. Five different lines represents the five different edge addition models. After careful observation, we reached the conclusion that we cannot generalize a particular value of $MaxPercentCrossing$ to all the networks, the primary cause of this is that the percentage of nodes crossing their thresholds vary from network to network. For example, in case of the Email network, the percentage of nodes crossing thresholds varies between 0 to 3, on the other hand, in case of a PGP network it varies from 0 to 1.5. So, if we choose $MaxPercentCrossing$ as 2 or above, we will not run DSLM algorithm in any phase of PGP network for any of the edge addition strategies. We believe that choosing correct value of $MaxPercentCrossing$ depends on various factors such as modularity of the network, total number of edges to total number of nodes ratio, average degree of the nodes, number of triangle in the graph etc. Due to the scope of our thesis, we have chosen $\frac{E}{V}$ ratio as the single deciding parameter. We assign $\alpha * \frac{E}{V}$ as $MaxPercentCrossing$ where Table 10 displays the value of α based on percentage of nodes crossing their thresholds.

Percentage of nodes crossing their thresholds	0 to 4	4 to 8	8 to 12	12 to 16	16 to 20	20 to 24
α	0.3	0.6	0.7	0.75	0.8	0.825

TABLE 10: Deciding parameter α

Next, we calculate the different values of $MaxPercentCrossing$ of each network for all the phases. The values are displayed in Table 11. By following this strategy, we can observe that for the PGP network, the values of $MaxPercentCrossing$ vary between 0.6966 to 0.7513, on other hand in case of an Email network it varies in between 2.9237 to 2.9819. Hence, for every network, we have a predefined range. Using these values of $MaxPercentCrossing$, we run our experiments. The results are displayed in Table 12. There are the five phases for each edge addition strategy and each phase have 10 graphs. For any of the graphs if the percentage of nodes crossing their thresholds exceeds $MaxPercentCrossing$ then we run the DSLM algorithm. Otherwise we do not run DSLM algorithm.

We want to observe, whether the Edge-Distribution-Analysis algorithm gives the desired results specified in Table 6. For the first 5 phases of edge additions, Table 12 summarizes the positive and negative errors made by our algorithm as compared to the ideal update decisions shown in Table 6. FP indicates a false positive error, that is, our algorithm suggests an update but the ideal decision would be not to update. Comparing with Table 6, we observe that we obtain a best result in case of an $P = 0, P = 0.4$ and

Network	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
Football	3.2608	3.3234	3.3860	3.4486	3.5113
Email	2.9237	2.9819	3.0402	3.0984	3.1567
Facebook	18.4310	18.8395	19.2480	19.6566	19.6566
PGP	0.6966	0.7103	0.7239	0.7376	0.7513
Condmat2003	2.5837	2.6344	2.6850	2.7357	2.7863
DBLP	0.7535	0.7675	0.7816	0.7957	0.809846
Livejournal	6.1345	6.2548	6.3751	6.4954	6.6156

TABLE 11: Values of $MaxPercentCrossing$ for different phases

$P = 0.6$ for all of the networks except DBLP. In all the networks apart from DBLP, the change in modularity is less than 0.005 and our algorithm advises not to run DSLM. In case of DBLP network, the change in modualarity is quite high. For example, when $P = 0.4$ it varies from 0.05 to 0.30, but we advise not to run the algorithm. In all the other cases, we save a lot of time by not running the DSLM algorithm. In case of an EdgeDistance model, for PGP network the change in modularity is greater than 0.005 from the 4^{th} phase, but we fail to catch it. For the Random model of an PGP network and Email network, the change in modularity is approximately 0.005 in the $2nd$ phase and the 5^{th} phase but in both the cases we advise to run from the 2nd phase onwards hence we have 2 false negatives in Table 12. For the DBLP network, we should run in all the phases but we run from the 3^{rd} phase onwards. For Random model of Football, Condmat, and Facebook network, our algorithm gives correct results. For Livejournal network, we get one false positive.

Table 13 summarizes total number of TP, TN, FP, FN and accuracy of the Edge-Distribution-Analysis algorithm for each network. Each network has 5 phases and 5 different edge addition models. For Football, Email, Facebook and Condmat networks, we have 100 % accuracy. We also get good results for Livejournal and PGP network, only exception is DBLP network in which we predict the correct snapshot to run a DSLM with lower accuracy rate.

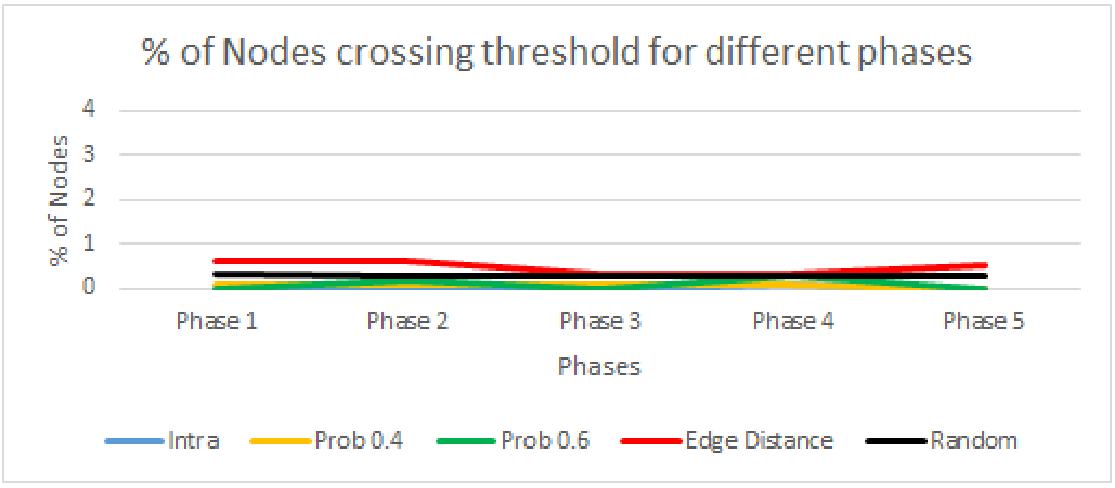


FIGURE 24: Percentage of nodes crossing thresholds per phase for Football n/w

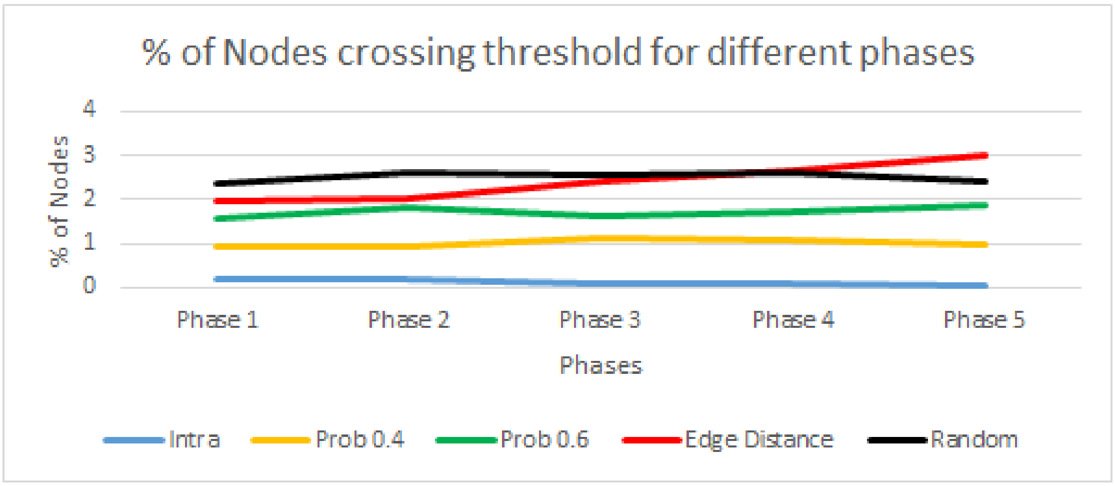


FIGURE 25: Percentage of nodes crossing thresholds per phase for Email n/w

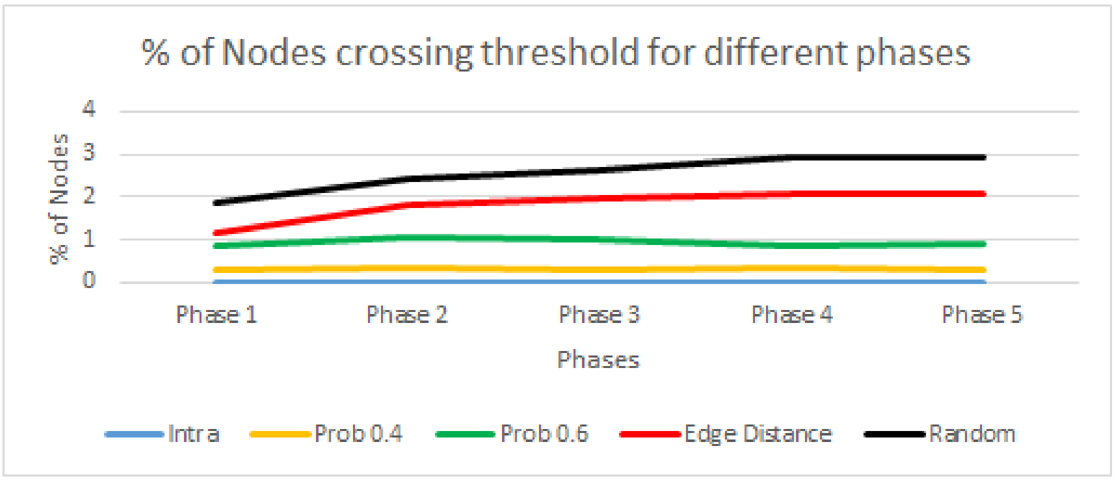


FIGURE 26: Percentage of nodes crossing thresholds per phase for Facebook n/w

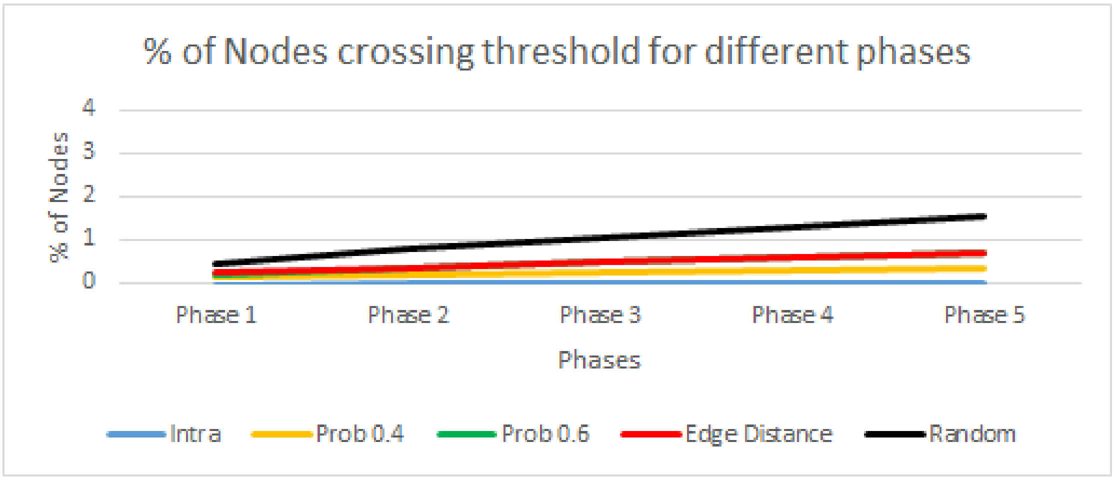


FIGURE 27: Percentage of nodes crossing thresholds per phase for PGP n/w

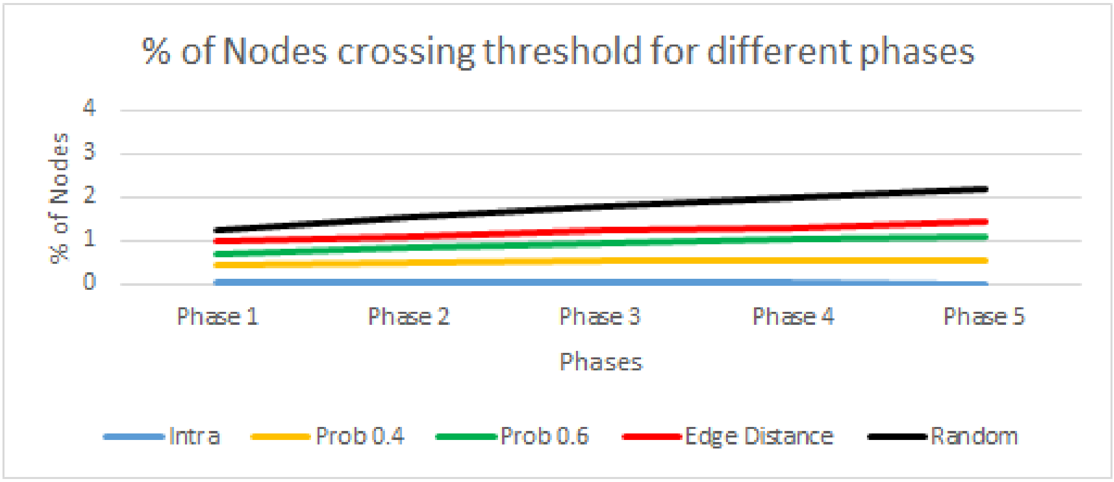


FIGURE 28: Percentage of nodes crossing thresholds per phase for Condmat n/w

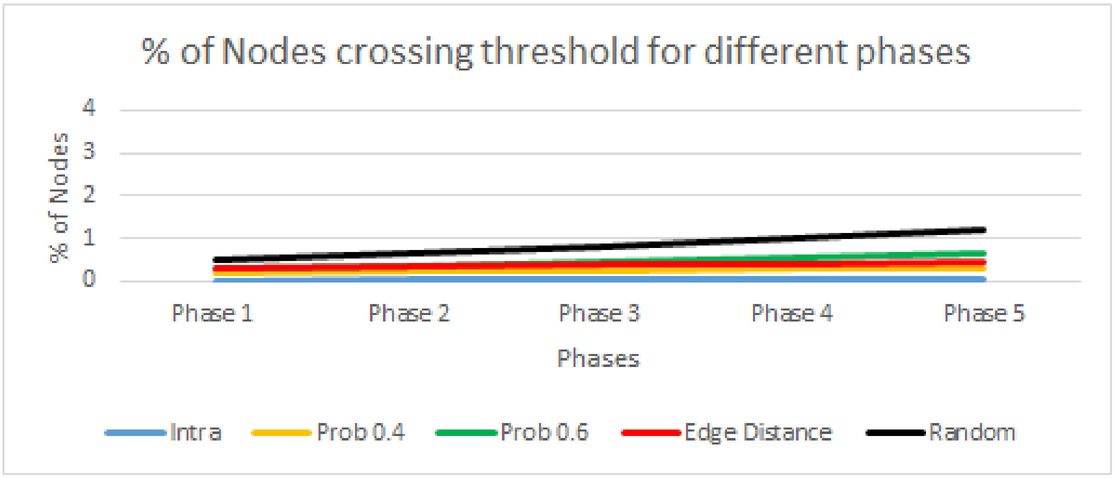


FIGURE 29: Percentage of nodes crossing thresholds per phase for DBLP n/w

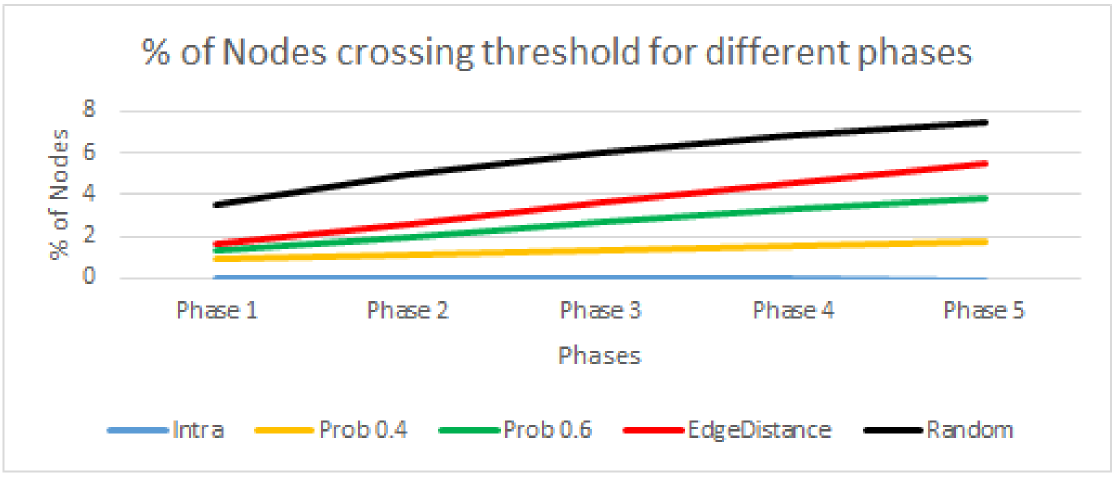


Figure 30: Percentage of nodes crossing thresholds per phase for Livejournal n/w

Network	Model	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5
Football	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	TN	TN
	Random	TN	TN	TN	TN	TN
Email	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	TN	TN
	Random	TN	TN	TN	TN	TN
Facebook	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	TN	TN
	Random	TN	TN	TN	TN	TN
PGP	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	FN	TN
	Random	TN	FP	TP	FP	FP
Condmat	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	TN	TN
	Random	TN	TN	TN	TN	TN
DBLP	P=0	TN	TN	TN	TN	TN
	P=0.4	FN	FN	FN	FN	FN
	P=0.6	FN	FN	FN	FN	FN
	EdgeDistance	TN	FN	TN	FN	TN
	Random	FN	FN	TP	TP	TP
Livejournal	P=0	TN	TN	TN	TN	TN
	P=0.4	TN	TN	TN	TN	TN
	P=0.6	TN	TN	TN	TN	TN
	EdgeDistance	TN	TN	TN	TN	TN
	Random	TN	TN	TN	TP	FP

TABLE 12: Edge-Distribution-Analysis algorithm false positives and false negatives

			Predicted		
			T	N	Accuracy (%)
Football	Actual	T	0	0	100
		N	0	25	
			Predicted		
			T	N	Accuracy(%)
Email	Actual	T	0	0	100
		N	0	25	
			Predicted		
			T	N	Accuracy(%)
Facebook	Actual	T	0	0	100
		N	0	25	
			Predicted		
			T	N	Accuracy(%)
PGP	Actual	T	1	1	84
		N	3	20	
			Predicted		
			T	N	Accuracy(%)
Condmat	Actual	T	0	0	100
		N	0	25	
			Predicted		
			T	N	Accuracy(%)
DBLP	Actual	T	3	14	44
		N	0	8	
			Predicted		
			T	N	Accuracy(%)
Livejournal	Actual	T	1	0	96
		N	1	23	

TABLE 13: Confusion Matrix and Accuracy for Edge-Distribution-Analysis algorithm

5.6 Comparison of algorithms: Modularity and Time

In this section, we evaluate the performance of our two algorithms to two approaches from the literature in terms of both modularity and time. In the first approach, denoted SLM, we simply run SLM [47] from scratch for each of the graphs G_0, G_1, G_2, G_3, G_4. In the approach denoted DSLM, we run the DSLM algorithm [3] after every phase; recall that this is essentially running SLM but starting with the known community structure for G_0. Using the two algorithms we proposed in Chapter 3, after every phase, we decide whether or not to update the community structure. If we decide to update, we run DSLM. Clearly, by deciding not to update, we may end up with a lower modularity value; however we hope to save in terms of time. In the following sections, we present our experimental results comparing the modularity and time taken by the four approaches.

5.6.1 Comparative Performances: Modularity

For Figures 31 to 37, the x-axis represents phases and y-axis represents modularity. From these figures, it can be seen that regardless of the model for edge addition, the difference in modularity obtained by different algorithms is generally less than 0.005 for all the networks.

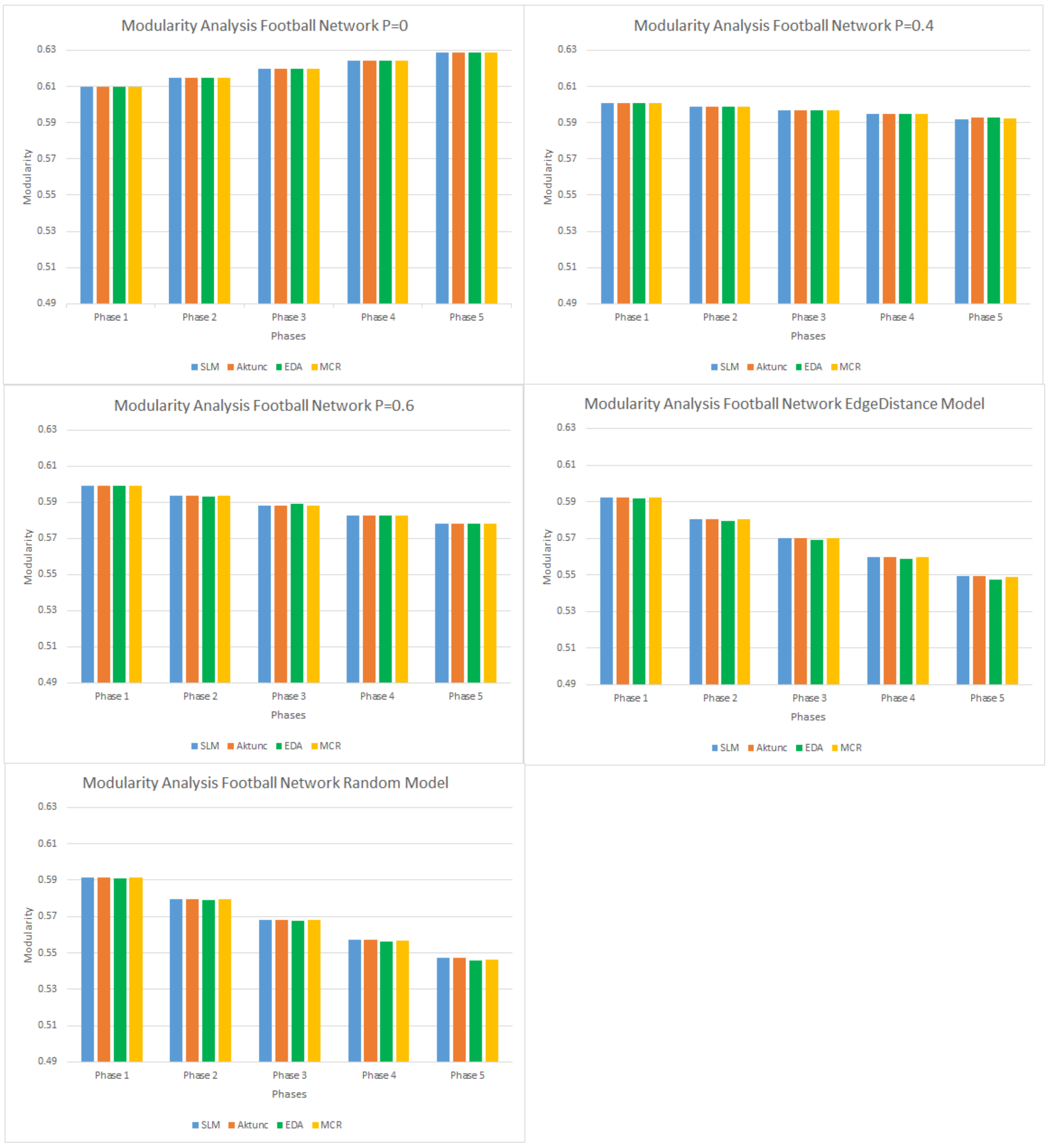


FIGURE 31: Modularity Analysis for Football Network for different edge addition models

The exceptions are PGP and DBLP network. For PGP network, from Figure 34, we observe the difference is 0.0064 only for 5th phase of an EdgeDistance model. For other edge addition models, we observe that it is less than 0.005. In case of the DBLP network, the modularities obtained from our Edge-Distribution-Analysis algorithm are significantly lower than DSLM and SLM for some of the phases. As previously explained, the reason for low modularity is the approximate prediction of threshold cut off value. We also observe that PGP and DBLP networks have lower E/V ratio; this could be the reason for the anomalous behavior.

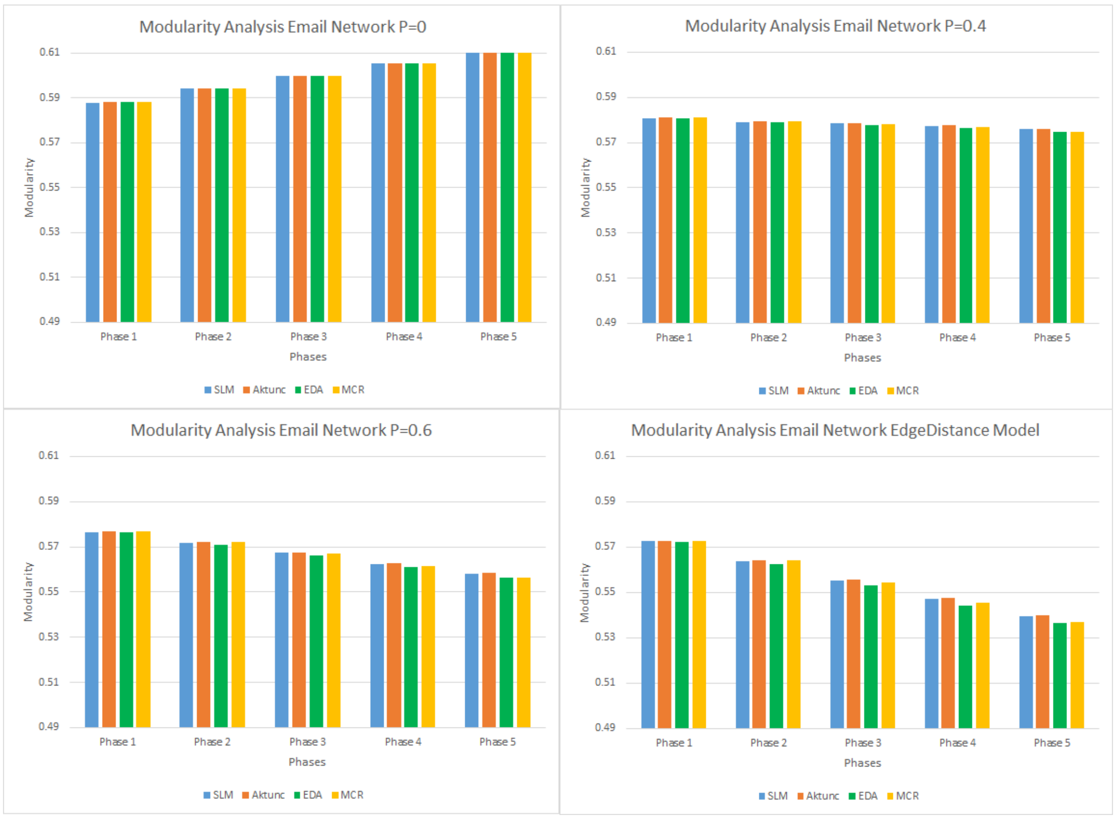


FIGURE 32: Modularity Analysis for Email Network for different edge addition models

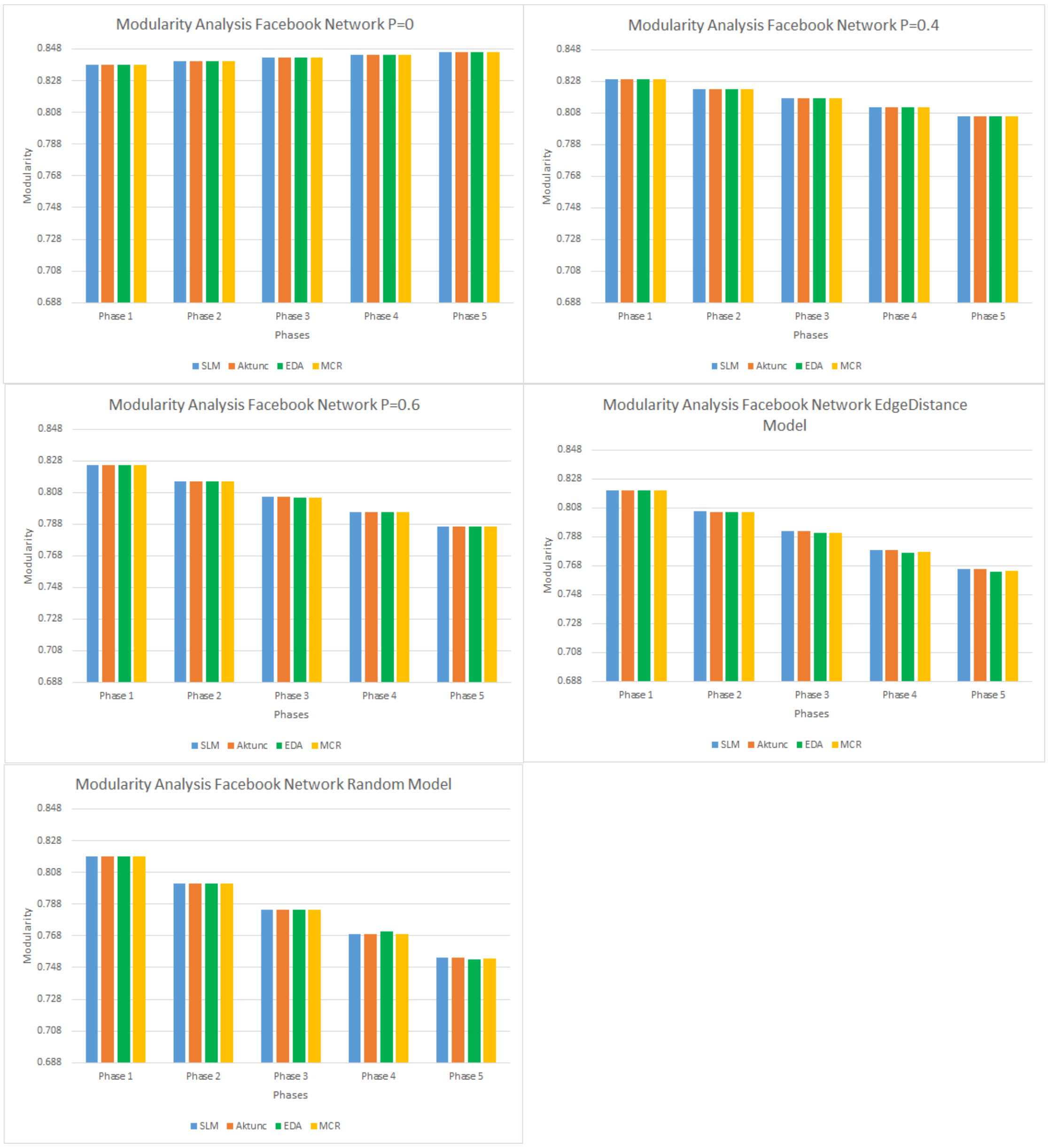


FIGURE 33: Modularity Analysis for Facebook Network for different edge addition models

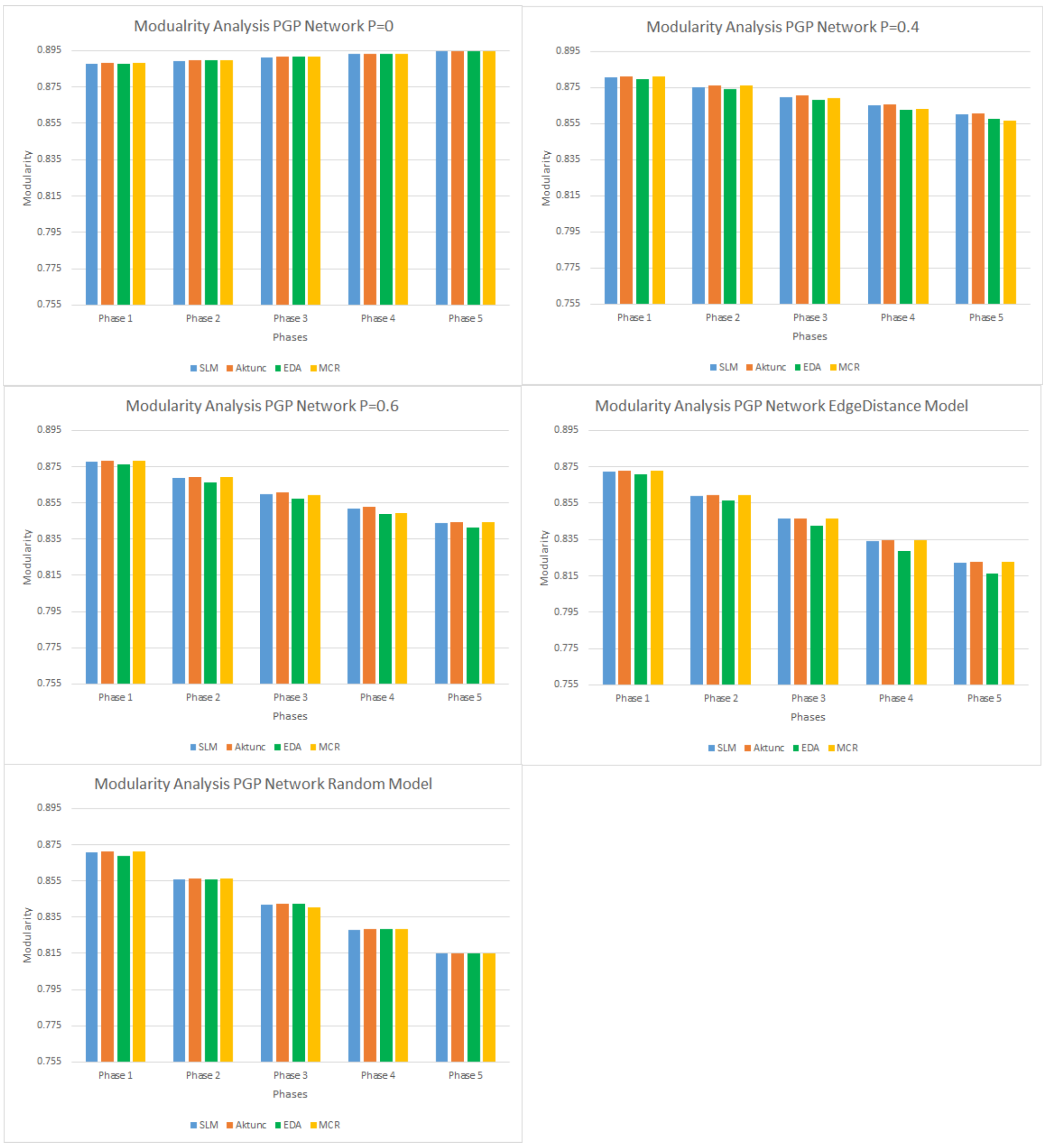


FIGURE 34: Modularity Analysis for PGP Network for different edge addition models

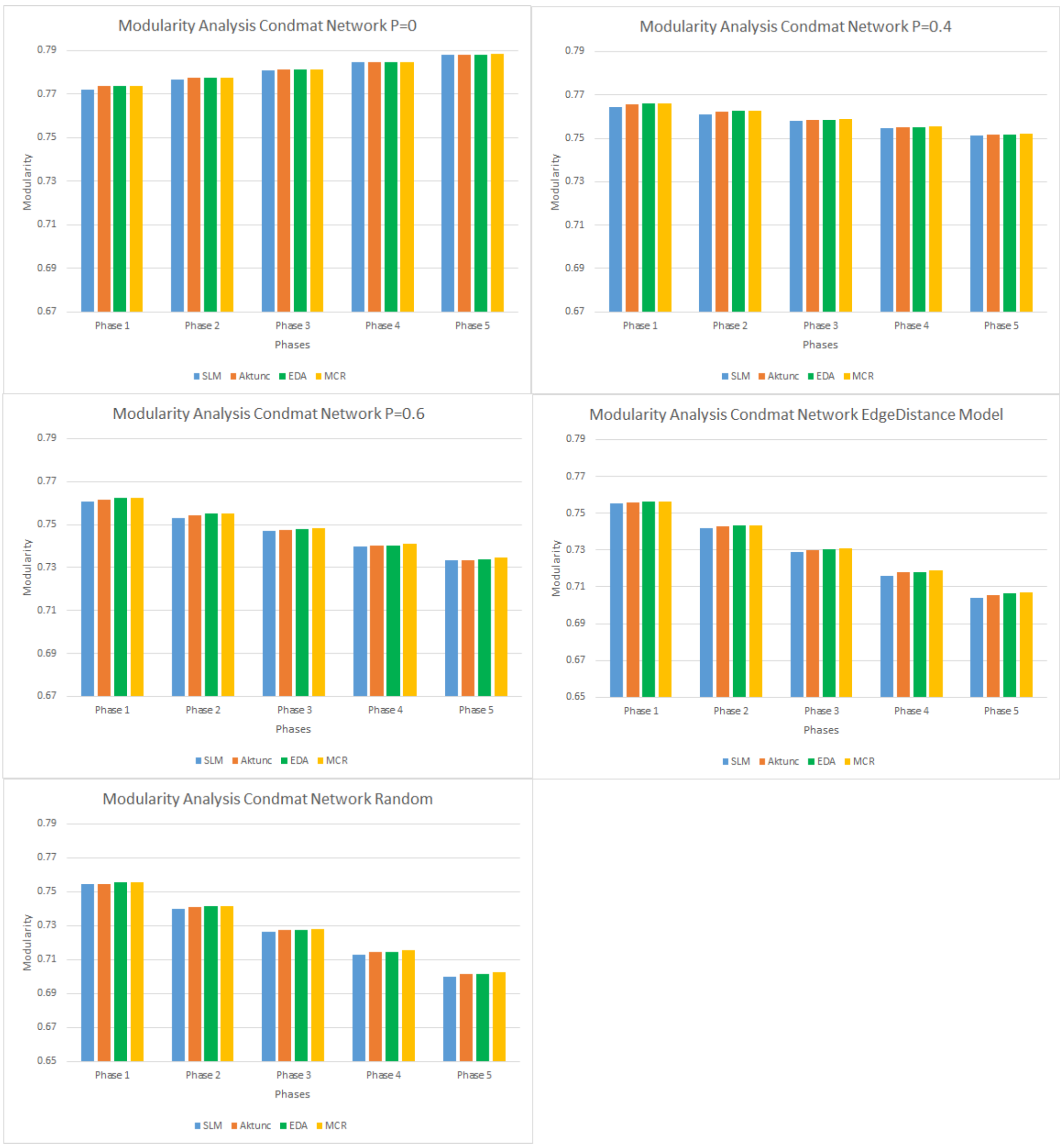


FIGURE 35: Modularity Analysis for Condmat Network for different edge addition models

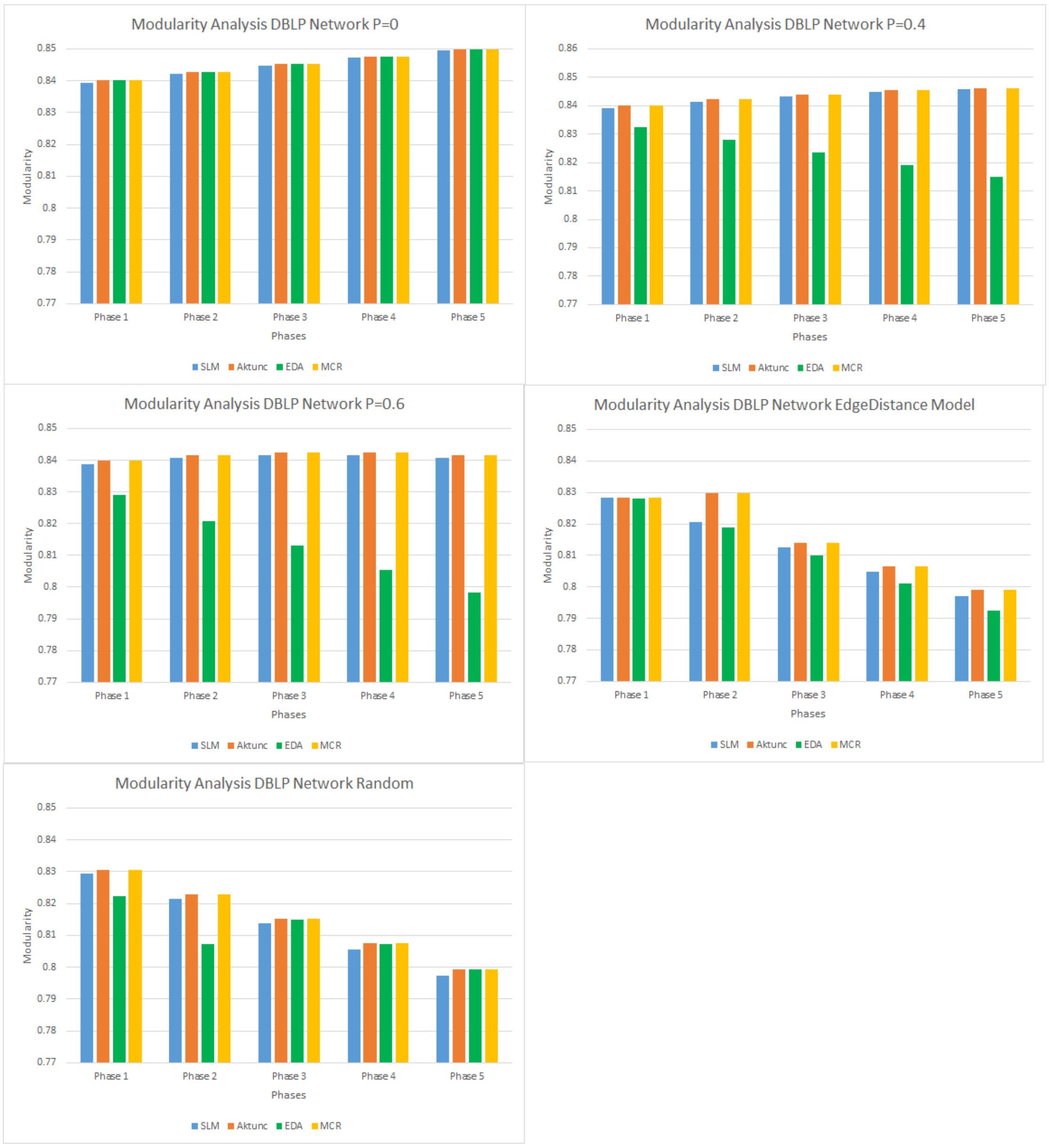


FIGURE 36: Modularity Analysis for DBLP Network for different edge addition models

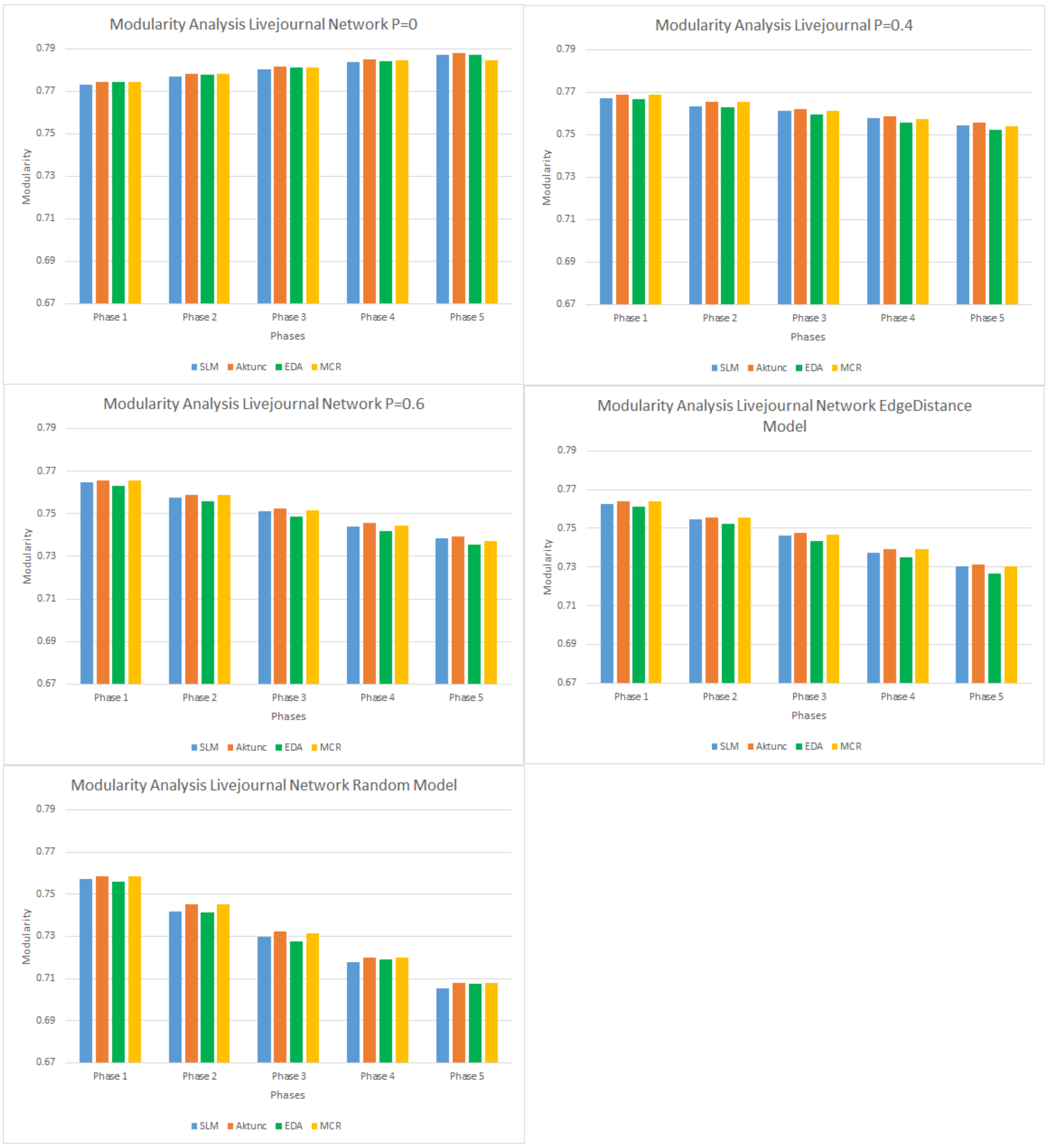


FIGURE 37: Modularity Analysis for Livejournal Network for different edge addition models

5.6.2 Comparative Performances: Time

For Figures 38, 39, 40, 41, 42, 43, and 44 x-axis represents phases and y-axis represents time in seconds. For the Football network, we can observe from Figure 38 that the Modularity-Change-Rate algorithm gives the least running time as it is not running from the 3^{rd} phase onwards. The time taken by SLM and DSLM is almost equal whereas the Edge-Distribution-Analysis algorithm gives the worst running time. The reason behind this is that we are storing the generated thresholds in separate file, so, even though we are not running the DSLM algorithm in Edge-Distribution-Analysis algorithm , the I/O time to write the results is at least 0.05 seconds. This happens only for this network.

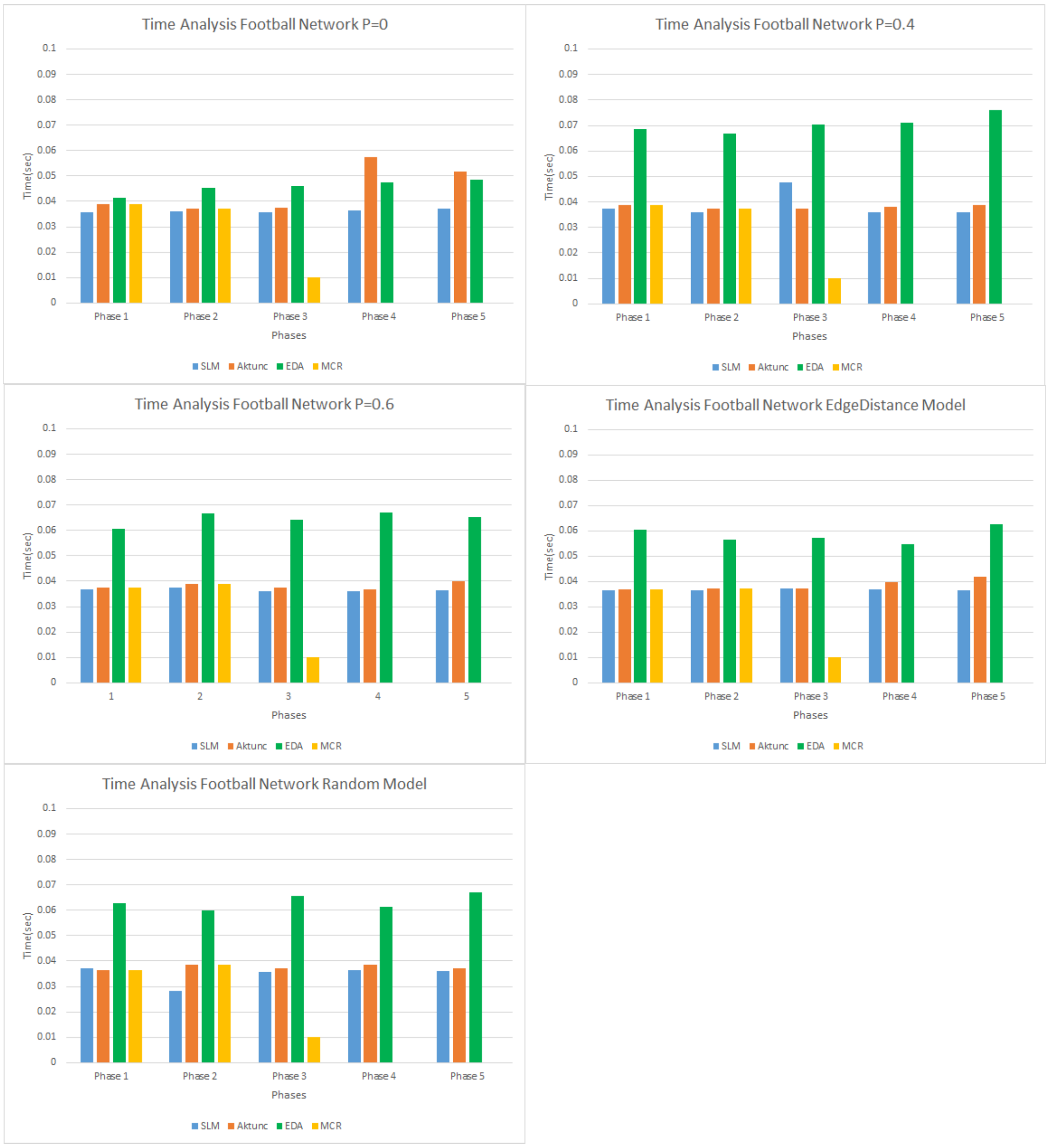


FIGURE 38: Time Analysis for Football network with different models

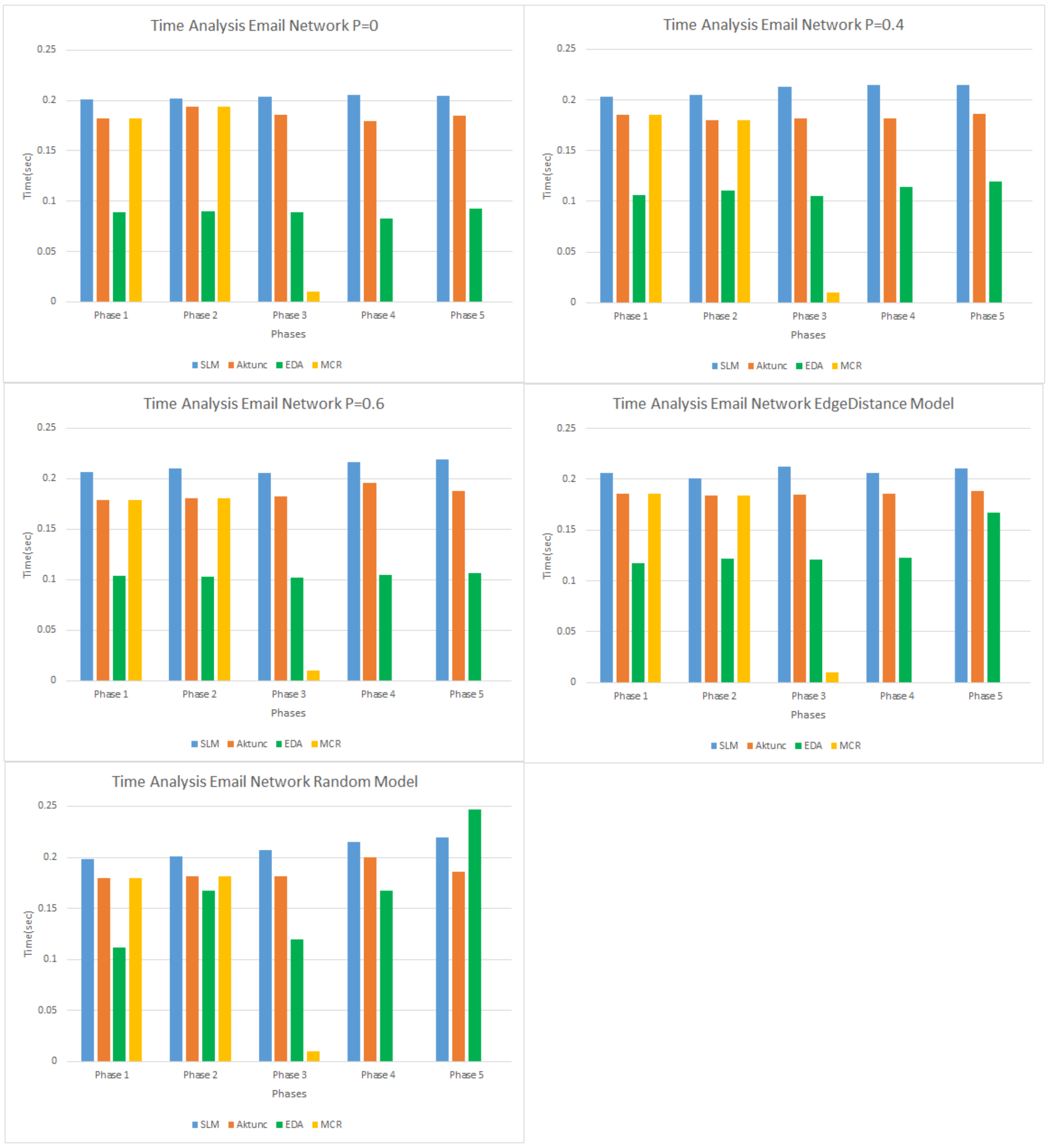


FIGURE 39: Time Analysis for Email network with different models

For the Email, Facebook, and the Condmat networks, from Figures 39, 40, and 42, we observe that the Edge-Distribution-Analysis algorithm takes the least time for first two phases and that the Modularity-Change-Rate algorithm takes the least time from 3rd phase onwards. As expected, the SLM algorithm takes the highest running time for all the edge addition models for all of the phases.

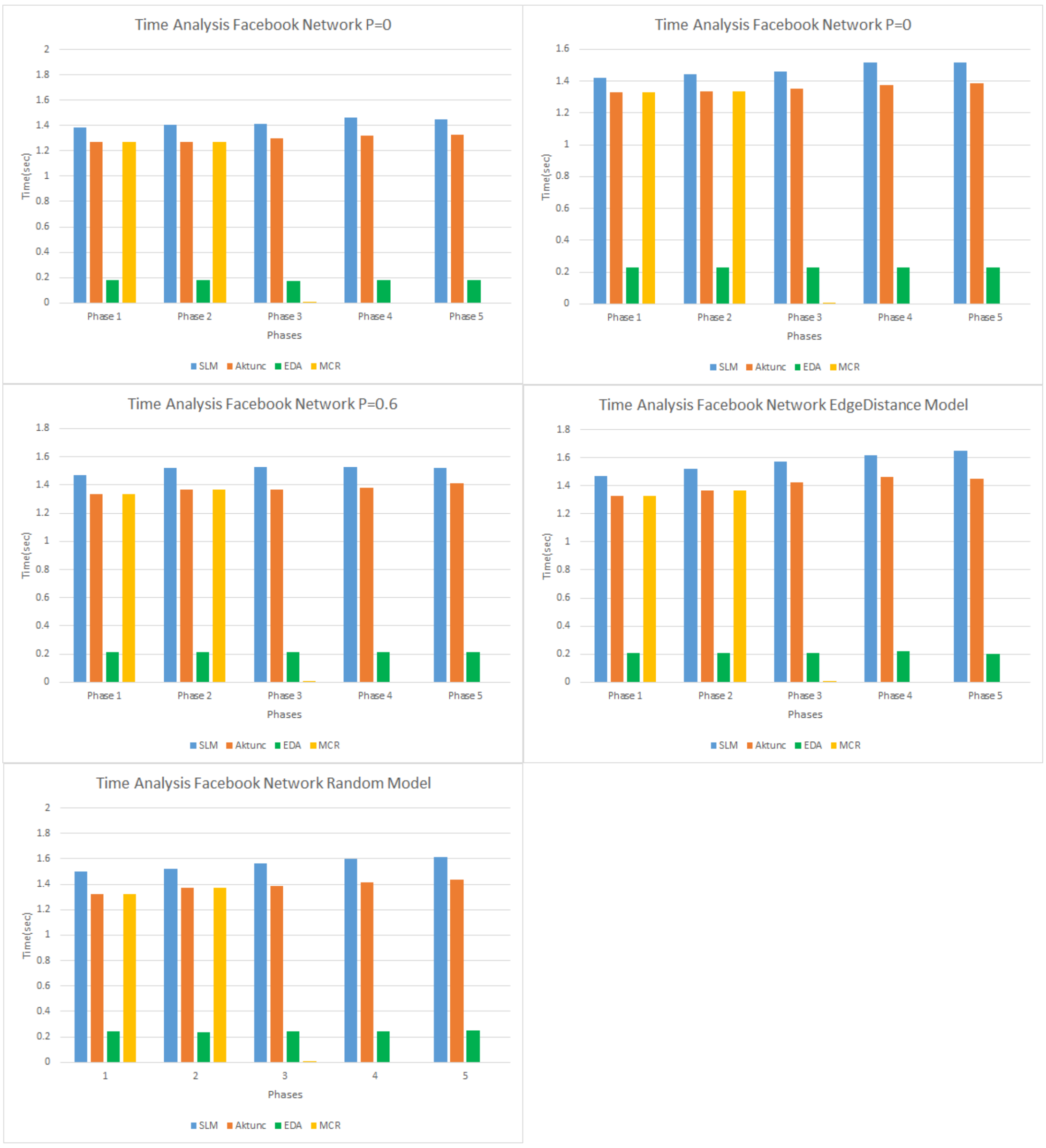


FIGURE 40: Time Analysis for Facebook network with different models

For PGP network from Table 7, we observe that we are running the DSLM algorithm in Modularity-Change-Rate algorithm for the 1^{st}, 2^{nd} & the 5^{th} phase of the $p = 0.6$, for all the phases of EdgeDistance and for the 1^{st}, 2^{nd}, 4^{th} and the 5^{th} phase of Random model. Similarly, from Table 12 we see that we are running the DSLM algorithm in Edge-Distribution-Analysis algorithm for the 5th phase of $P = 0.6$ and EdgeDistance model. We also run DSLM in all the phases of the Random model. Hence we will have different run-time behavior for PGP network as compared to other networks.

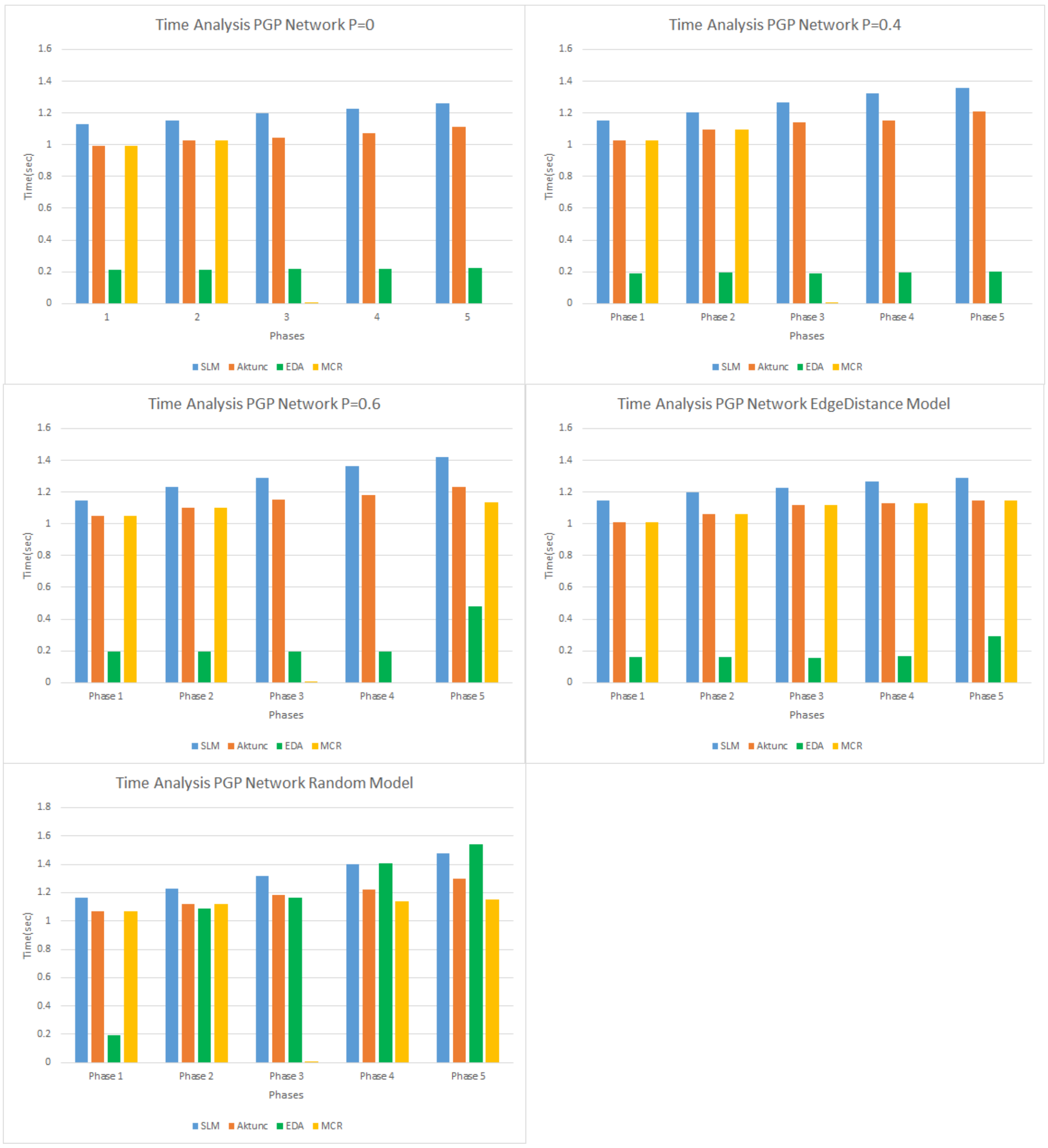


FIGURE 41: Time Analysis for PGP network with different models

In terms of overall running time, from Figure 41, we realize that Edge-Distribution-Analysis algorithm gives better results in general as compared to all the other algorithms. Modularity-Change-Rate algorithm is still running the DSLM algorithm in some of the phases where the difference in modularity is actually not greater than 0.005. The SLM algorithm gives the worst running time.

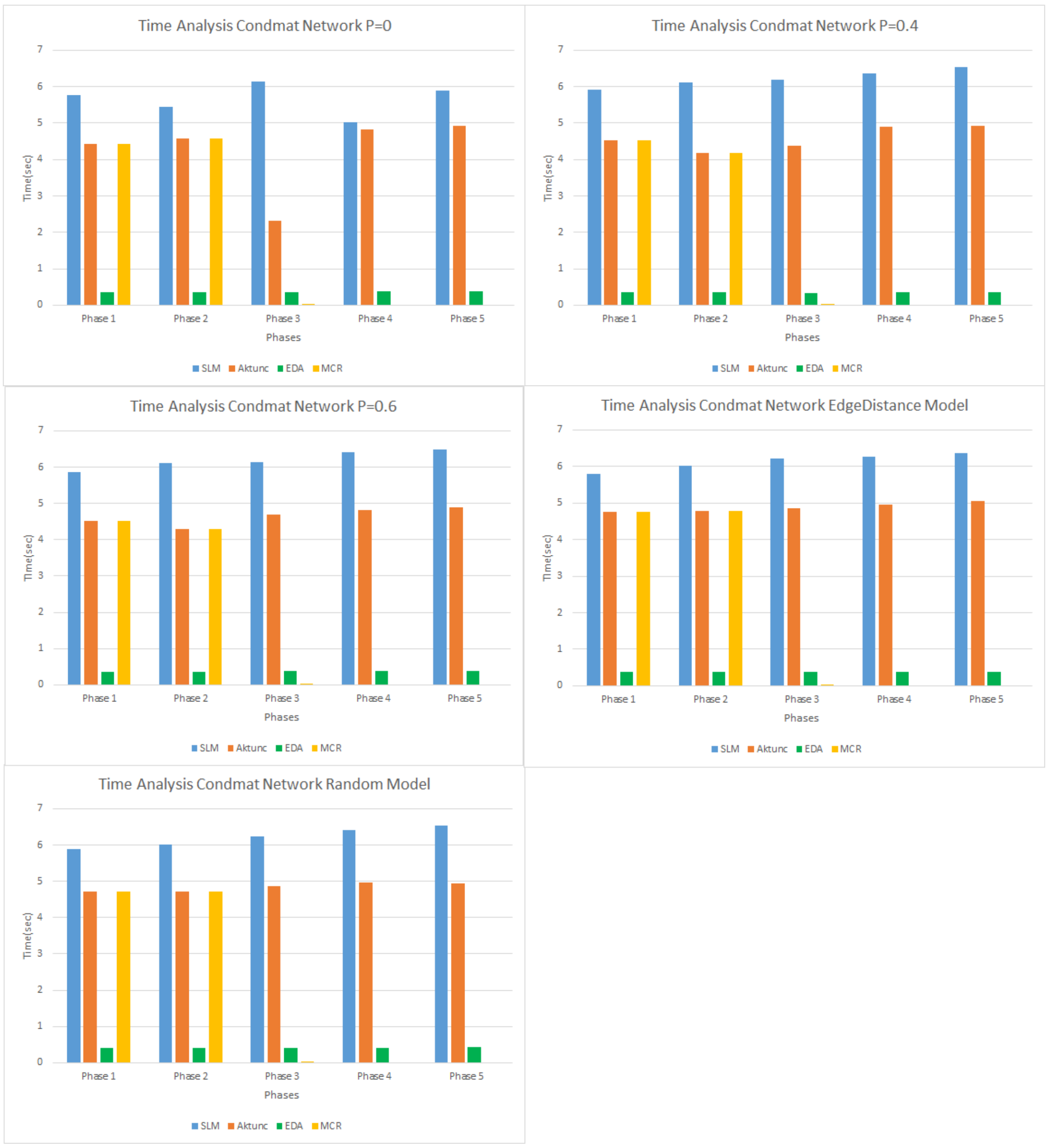


FIGURE 42: Time Analysis for Condmat network with different models

For DBLP network, DSLM and Modularity-Change-Rate algorithm gives better results. Edge-Distribution-Analysis algorithm fails to predict correct phase in which it should run an update. The SLM algorithm takes the highest running time. For $p = 0$ Modularity-Change-Rate algorithm gives better results than DSLM algorithm.

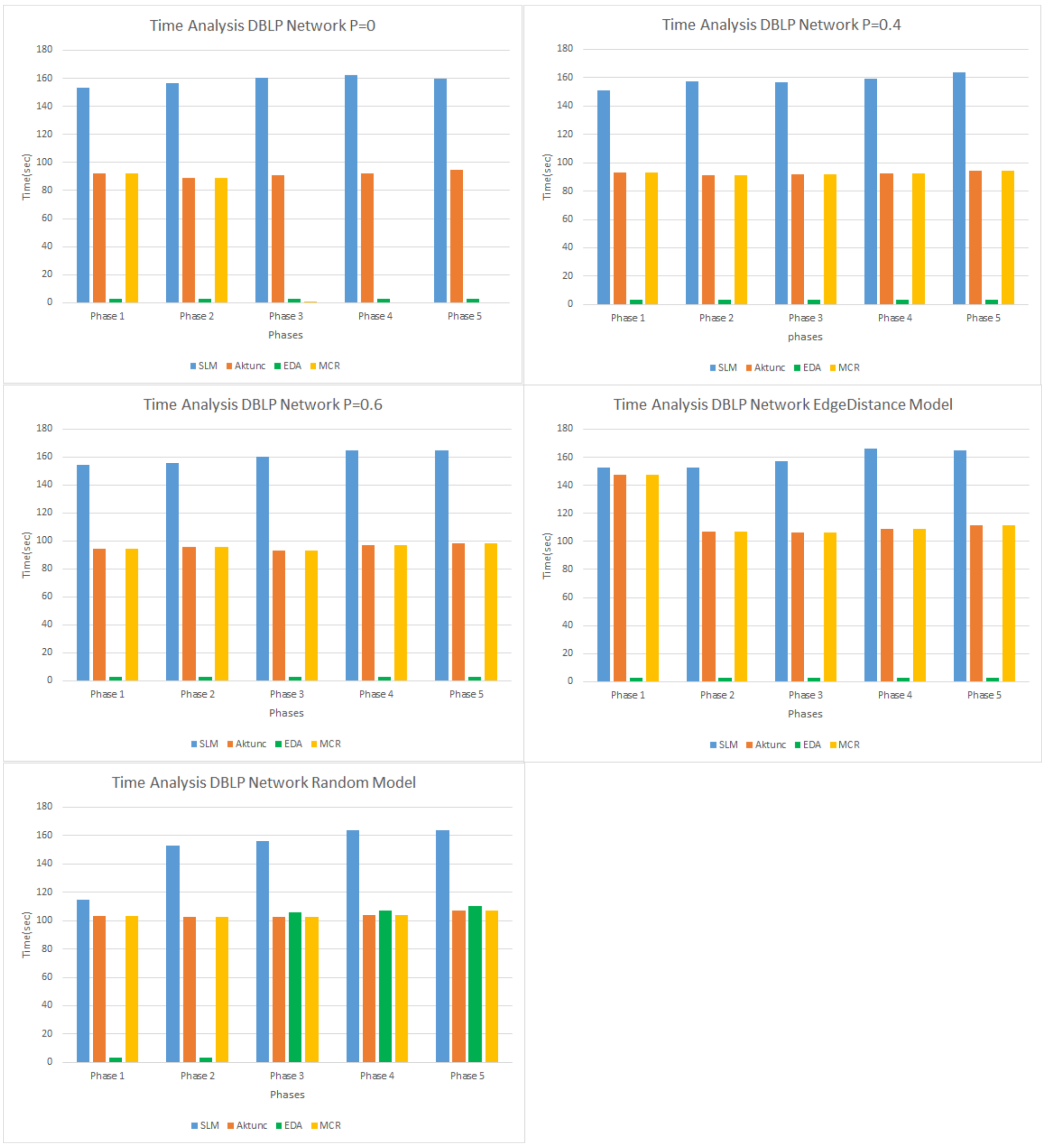


FIGURE 43: Time Analysis for DBLP network with different models

From Figure 44, we understand that, on an average, it takes about 1200 seconds to run SLM algorithm and approximately 550 seconds to run DSLM algorithm for a single snapshot of Livejournal network. From Figure 23,we can observe that the change in modularity is greater than 0.005 for the 4^{th} and 5^{th} phase of the Random model. From Figure 44, we realize that the Edge-Distribution-Analysis algorithm runs only in the 4^{th} and 5^{th} phase of Random model hence giving the best running time as compared to other models. The Modularity-Change-Rate algorithm gives good results for $P = 0$ and $P = 0.4$. When $P = 0.6$ it runs the DSLM algorithm even though the actual change in modularity is less than 0.005. The same thing happens for the 4^{th} and the 5^{th} phase of EdgeDistance model. For the Random model, it runs DSLM in all the phases. The SLM

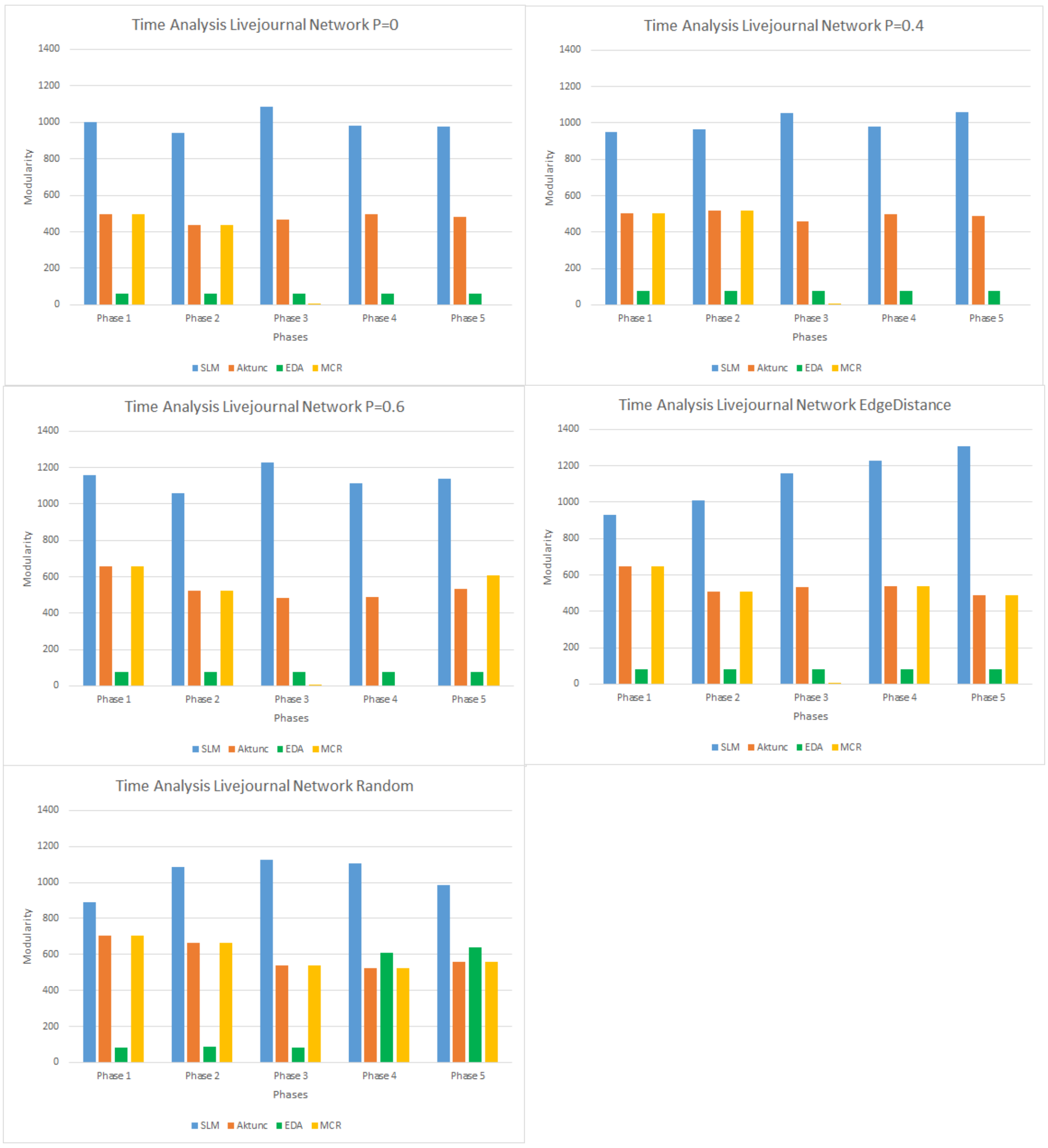


FIGURE 44: Time Analysis for Livejournal network with different models

algorithm gives the worst running time performance as compared to other algorithms.

Chapter 6

Conclusion and Future Work

In this thesis, we studied the problem of updating community structure for evolving social networks. In particular, we assume that a modularity-maximization based algorithm for community detection, such as DSLM, is being used. After the addition of a certain number of edges to the network, we attempt to determine whether or not the community structure has changed sufficiently to justify running a community detection algorithm. We give two algorithms to make this determination. The Edge-Distribution-Analysis algorithm analyzes the newly added edges to estimate how many nodes would switch communities as a result of the newly added edges. It then uses the percentage of such nodes to determine if the community structure would change significantly, and decides whether or not to re-run DSLM. The Modularity-Change-Rate algorithm is based on the idea that the rate of change of modularity is network-specific and linear in the number of edges added. It finds the rate of modularity change in a given network, and uses it to predict when an update is required.

We proposed three models to generate evolving networks: the Random model, the Geometric-Probability model, which is based on the well-known phenomenon of homophily in social networks, and the Edge Distance model, that is based on the phenomenon of triadic and cyclic closure. Starting with real-world data sets, we used these models to generate evolving networks. We implemented SLM, DSLM, Edge-Distribution-Analysis algorithm , and Modularity-Change-Rate algorithm on these data sets. Our results show that both the Edge-Distribution-Analysis algorithm and Modularity-Change-Rate algorithm predict quite well when the community structure should be updated. They result in significant computational savings compared to running SLM or DSLM after a fixed number of edge additions, while ensuring that the quality of the community structure is comparable.

6.1 Future Work

There is lot of scope for future work on this topic. As social networks are getting larger day by day there is an increasing need for effective strategies that can handle a lot of changes in the network. We have only focused on edge additions. In a real network, apart from edge addtions there are edge deletions, node additions and node deletions as well. In this thesis, we proposed three models for edge additions. It would be interesting to see how well the EdgeDistance model and the GeometricProbability models match the real-life evolution of networks. In general, finding an accurate model for social network evolution would be an interesting avenue for research.

For the Edge-Distribution-Analysis algorithm , we have given a few threshold calculations for adding edges of specific types. A study of how the simultaneous addition of different types of edges affects whether a node would switch communities needs to be undertaken. Also, it is far from clear how to determine the maximum percentage of nodes that would cross their thresholds before the community structure changes. We have suggested that it is based on the E/V ratio, but in reality, there are many other factors which could be considered, such as the betweenness centrality, degree of the node, number of triangles the node participates in, etc. An approach using machine learning to find which parameters are most important in compelling a node to switch communities might be useful. In the Modularity-Change-Rate algorithm, we run the DSLM algorithm for the first two phases and then predict the next phase in which we should run the DSLM. Once we reach that phase, we run the DSLM for that phase and for the two more subsequent phases. We suggest that we might not have to run the algorithm for those two subsequent phases. Instead we could run for a single phase and predict the change in modularity, and obtain even more savings in time.

Bibliography

[1] Pretty good privacy network dataset – KONECT, September 2016.

[2] U. rovira i virgili network dataset – KONECT, January 2016.

[3] R. Aktunc, I. H. Toroslu, M. Ozer, and H. Davulcu. A dynamic modularity based community detection algorithm for large-scale networks: Dslm. In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, ASONAM '15, pages 1177–1183, 2015.

[4] H. Alvari, S. Hashemi, and A. Hamzeh. Discovering overlapping communities in social networks: A novel game-theoretic approach. *AI Commun.*, 26(2):161–177, April 2013.

[5] T. Alzahrani and K. J. Horadam. *Community Detection in Bipartite Networks: Algorithms and Case studies*, pages 25–50. 2016.

[6] M. J. Barber and J. W. Clark. Detecting network communities by propagating labels under constraints. *Phys. Rev. E*, 80:026129, Aug 2009.

[7] J. J. Bechtel, W. A. Kelley, T. A. Coons, M. G. Klein, D. D. Slagel, and T. L. Petty. Lung cancer detection in patients with airflow obstruction identified in a primary care outpatient practice. *CHEST Journal*, 127(4):1140–1145, 2005.

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *NIPS*, volume 14, pages 585–591, 2001.

[9] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 10(10008), October 2008.

[10] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424(45):175 – 308, 2006.

[11] M. Bogu, R. Pastor-Satorras, A. Daz-Guilera, and A. Arenas. Models of social networks based on social distance attachment. *Phys. Rev. E*, 70(5):056122, 2004.

[12] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. Maximizing modularity is hard. *physics.data-an*, 2006.

[13] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(066111), Dec 2004.

[14] Easley D. and Kleinberg J. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World.* 2010.

[15] P. Erdos and A. Renyi. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[16] E. Estrada and N. Hatano. Communicability graph and community structures in complex networks. *Appl. Math. Comput.*, 214(2):500–511, August 2009.

[17] C. Fan, K. Xiao, B. Xiu, and G. Lv. A fuzzy clustering algorithm to detect criminals without prior information. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 238–243, Aug 2014.

[18] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(35):75 – 174, 2010.

[19] K. A. Fredericks and M. M. Durland. The historical evolution and basic concepts of social network analysis. *New Directions for Evaluation*, 2005(107):15–23, 2005.

[20] A. Friggeri, R. Lambiotte, M. Kosinski, and E. Fleury. Psychological aspects of social communities. In *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Conference on Social Computing*, pages 195–202, Sept 2012.

[21] Roger Guimer, Leon Danon, Albert Daz-Guilera, Francesc Giralt, and Alex Arenas. Self-similar community structure in a network of human interactions. *Phys. Rev. E*, 68(6):065103, 2003.

[22] R. Guimerà, M. Sales-Pardo, and L. A. N. Amaral. Modularity from fluctuations in random graphs and complex networks. *Phys. Rev. E*, 70:025101, Aug 2004.

[23] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307, Feb 1970.

[24] G. Kossinets and D. J. Watts. Empirical analysis of an evolving social network. *science*, 311(5757):88–90, 2006.

[25] G. Laurent, J. Saramäki, and M. Karsai. From calls to communities: a model for time-varying social networks. *The European Physical Journal B*, 88(11):1–10, 2015.

[26] J. Lee, S. P. Gross, and J. Lee. Modularity optimization by conformational space annealing. *Phys. Rev. E*, 85:056702, May 2012.

[27] J. Leskovec, L. Backstrom, R. Kumar, and A. Tomkins. Microscopic evolution of social networks. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 462–470, 2008.

[28] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection, June 2014.

[29] X. Liu and T. Murata. Advanced modularity-specialized label propagation algorithm for detecting communities in networks. *Physica A: Statistical Mechanics and its Applications*, 389(7):1493 – 1500, 2010.

[30] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti. Generalized louvain method for community detection in large networks. In *2011 11th International Conference on Intelligent Systems Design and Applications*, pages 88–93, 2011.

[31] W. E Moore. Man, time and society. 1963.

[32] M. E. J. Newman. Analysis of weighted networks. *Phys. Rev. E*, 70:056131, Nov 2004.

[33] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69:066133, Jun 2004.

[34] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

[35] M.E.J. Newman. Detecting community structure in networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):321–330, 2004.

[36] A. Y. Ng, M. I. Jordan, Y. Weiss, et al. On spectral clustering: Analysis and an algorithm. In *NIPS*, volume 14, pages 849–856, 2001.

[37] W.F. Pan, B. Jiang, and B. Li. Refactoring software packages via community detection in complex software networks. *International Journal of Automation and Computing*, 10(2):157–166, 2013.

[38] S. Papadopoulos, Y. Kompatsiaris, A. Vakali, and P. Spyridonos. Community detection in social media. *Data Mining and Knowledge Discovery*, 24(3):515–554, 2012.

[39] J. Reichardt and S. Bornholdt. Statistical mechanics of community detection. *Phys. Rev. E*, 74:016110, Jul 2006.

[40] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[41] R. Rotta and A. Noack. Multilevel local search algorithms for modularity clustering. *J. Exp. Algorithmics*, 16:2.3:2.1–2.3:2.27, July 2011.

[42] A. J. Seary and W. D. Richards. Partitioning networks by eigenvectors. In *Proceedings of the International Conference on Social Networks*, volume 1, pages 47–58, 1995.

[43] M. Takaffoli, J. Fagnan, F. Sangi, and O. R. Zaane. Tracking changes in dynamic information networks. In *2011 International Conference on Computational Aspects of Social Networks (CASoN)*, pages 94–101, Oct 2011.

[44] M. Takaffoli, R. Rabbany, and O. R. Zaane. Community evolution prediction in dynamic social networks. In *2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014)*, pages 9–16, 2014.

[45] J. Tang, X. Hu, and H. Liu. Social recommendation: a review. *Social Network Analysis and Mining*, 3(4):1113–1133, 2013.

[46] V. A. Triag, P. Van Dooren, and Y. Nesterov. Narrow scope for resolution-limit-free community detection. *Phys. Rev. E*, 84:016114, Jul 2011.

[47] L. Waltman and N. J. van Eck. A smart local moving algorithm for large-scale modularity-based community detection. *The European Physical Journal B*, 86(471), 2013.

[48] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: The state-of-the-art and comparative study. *ACM Comput. Surv.*, 45(4):43:1–43:35, August 2013.

[49] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *2012 IEEE 12th International Conference on Data Mining*, pages 745–754, Dec 2012.

[50] J. Yang and J. Leskovec. Overlapping community detection at scale: A nonnegative matrix factorization approach. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, WSDM '13, pages 587–596, 2013.

[51] J. Yang, J. McAuley, and J. Leskovec. Detecting cohesive and 2-mode communities indirected and undirected networks. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM '14, pages 323–332, 2014.

[52] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.