COMMUNITY DETECTION IN COMPLEX NETWORKS USING LOCAL METHODS AND INFORMATION FLOW

by

Mürsel Taşgın

B.S., Computer Engineering, Middle East Technical University, 2002

M.S., Computer Engineering, Boğaziçi University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Computer Engineering

Boğaziçi University

2019

COMMUNITY DETECTION IN COMPLEX NETWORKS USING LOCAL METHODS AND INFORMATION FLOW

APPROVED BY:

Prof. Haluk Bingöl (Thesis Supervisor)	.....
Prof. Ali Taylan Cemgil	.....
Assoc. Prof. Arzucan Özgür	.....
Keziban Orman, Ph.D.	.....
Prof. Şule Gündüz Öğüdücü	.....

DATE OF APPROVAL: 21.12.2018

ACKNOWLEDGEMENTS

I am a proud member of Boğaziçi University. It has a unique atmosphere where I have the opportunity to meet great people who influenced me during my Ph.D. First, I thank my supervisor, Haluk Bingöl, for his guidance and all the great discussions we have on interesting topics; and above all introducing me with the great research area of complex systems and networks. I want to thank the jury members; Şule Gündüz Öğüdücü, Arzucan Özgür, Keziban Günce Orman and Ali Taylan Cemgil who helped me with their invaluable comments and feedbacks. I want to thank Suzan Üsküdarlı for the discussions and talks we had on topics ranging from complex networks to semantic web and artificial intelligence. I want to thank Ali Taylan Cemgil for teaching me basics of machine learning and Bayesian statistics. I want to thank all my teachers and classmates in Boğaziçi University who enriched my vision on many interdisciplinary topics ranging from complex systems to artificial intelligence and machine learning. Thanks to TAM project and all the facilities provided by TETAM which helped me much on concentrating and speeding up my research.

I thank to my wife, Zehra Taşgın, who always encouraged me to pursue research and complete my Ph.D., even in the hardest times. With her great support and sacrifice, I am able to proceed with my research. My heart is with my beloved father Hasan Taşgın, my mother Elif Taşgın and all the family members. It was a pleasure to see my son, Kerem, drawing his first network on a paper to represent the connections he and his friends have in pre-school class. It was a true inspiring moment for me that shows how networks are natural representations of real physical world and how easily a child can understand it. He contributed much on my motivation to pursue research. I learned a lot during the journey; hard work, concentration, continuous learning, failure, revival and persistence. I am happy to have this, which changed the way I live, work and see things in life.

ABSTRACT

COMMUNITY DETECTION IN COMPLEX NETWORKS USING LOCAL METHODS AND INFORMATION FLOW

Many real world systems can be represented using networks or graphs where elements of system are denoted by nodes and their relations by edges. Complex networks are special kinds of these networks with their emergent features created by interactions among nodes. One such emergent feature is the community structure. A community is a group of nodes where nodes within same community have more connections (i.e., edges) with each other than with the nodes in the rest of the network. Identifying such communities is the task of community detection that can be used to identify nodes with similar functions or features, densely connected regions in networks, information flow patterns and spreading of a disease or information in a network.

In this thesis, we work on community detection on complex networks using local approach and information diffusion. We investigate current algorithms and try to understand the limitations of them. We especially focus on high time-complexity of algorithms because of using global approach, i.e., try to optimize a global metric or perform computations regarding the whole network repeatedly. We propose new algorithms using local approach (i.e., similarity based on common friends) and information diffusion (i.e., gossip spreading). Local approach uses locally available or computable information around a node to identify its community. With this, community detection task can be seen as a set of distributed and parallel tasks running simultaneously on different parts of the network. We also propose a variant of label propagation algorithm which decreases its overall execution time by eliminating unnecessary steps. During these studies, we develop a community detection framework which simplifies the task of defining, testing and comparing a new community detection algorithm.

ÖZET

YEREL YÖNTEMLER VE BİLGİ YAYILIMI KULLANARAK KARMAŞIK AĞLARDA KÜMELENME TESPİTİ

Gerçek hayattaki birçok sistem ağ gösterimi kullanılarak temsil edilebilir. Ağ gösteriminde, sistemi oluşturan bileşenler düğümlerle; bu bileşenlerin arasındaki ilişkiler ise bağlantılarla gösterilir. Özel bir ağ tipi olan karmaşık ağlar ise sistemin bileşenlerinin birbiriyle etkileşimi sonucu ortaya çıkan özelliklere sahiptir. Ağlardaki küme yapısı bu özelliklerden biridir. Bir ağ içindeki kümelenme, birbiriyle daha sıkı bağlarla bağlanmış düğümlerden oluşur; bu düğümlerin ağ içindeki diğer düğümlerle ise daha az bağlantısı bulunmaktadır. Bir ağ içinde bu tip kümelenmeleri bulmak, ağ içinde benzer özellik veya fonksiyona sahip alanların tespiti, bilginin dağılımı veya hastalık yayılımı gibi birçok alanda önemli kullanım alanına sahiptir. Bu tezde, karmaşık ağlardaki kümeleri yerel yaklaşımlı algoritmalarla (ör: ortak arkadaşlar üzerinden benzerlik) ve bilgi yayılımı yöntemleriyle (ör: dedikodu yayılımı) tespit etme üzerinde çalışmalar yaptık. Mevcutta kullanılan algoritmaları inceleyerek bunların yetersiz kaldığı alanları tespit ettik. Tüm ağ üzerinde bir değeri iyileştirmeye çalışan küresel yaklaşımlı algoritmaların uzun çalışma sürelerine sahip olduklarını gördük. Yerel yaklaşım kullanarak, dügümlerin benzerlikleri ve dedikodu yayılımını baz alan algoritmalar geliştirdik. Yerel yaklaşımlı algoritmaların çalışması icin, bir düğüm etrafındaki kısıtlı bilgi yeterli olmaktadır. Bu sayede, küme işlemi paralel ve dağıtık şekilde ağın farklı yerlerinde eşzamanlı yapılabilmektedir. Bununla birlikte, bilinen bir küme bulma algoritmasını baz alarak (etiket yayılım algoritması), algoritmada gereksiz yapılan adımları ortadan kaldıran bir yaklaşımla, çalışma zamanını kısaltan yeni bir algoritma geliştirdik. Tez çalışmaları sırasında tanımladığımız bu algoritmaların tanımlanması sürecinde yeni bir geliştirme altyapısı da oluşturduk.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

A_i	Local neighborhood of node i
C	Set of communities in a network
C_{min}	Minimum size of a community
C_{max}	Maximum size of a community
E	Set of edges in a graph
E^p	Set of edges in a preference network
e_{ij}	Edge between node i and j
f_c	Fitness value of chromosome c in genetic algorithm
G	A graph consisting of vertices and edges
G^p	Preference network obtained from G
$I(X;Y)$	Mutual information between X and Y
$J(i,j)$	Jaccard similarity of nodes i, j
k_0	Critical degree for gossip spreading in unweighted networks
k_0^w	Critical degree for gossip spreading in weighted networks
$\langle k \rangle$	Average degree of nodes in a network
k_i	Degree of node i in a network
k_{max}	Maximum degree of a node in a network
k_{out}	Number of outgoing edges (degree) of a node
L_i	Community membership label of node i
$\mathbb{L}$	Set of community labels
M	Number of edges in network
m_{ij}	Set of friends of node i who receives the gossip originated by j in a weighted network
N	Number of nodes in network
$p(i)$	Preference of node i
S	Set of boundary nodes in a graph
$s_i(j)$	Preference score of node i for neighbor j
V	Set of vertices in a graph
v_i	Node i in a graph

$w_{i,j}$	Edge weight between nodes i, j
$\overline{w_i}$	Average edge weight of node i
$\triangle_i$	Number of triangles around node i
$\wedge_i$	Number of triplets where i is in center
β_{ij}	Gossip spread factor of node i originated by j in a weighted network
β_i	Average gossip spread rate of node i in a weighted network
β_{k_α}	Average gossip spread factor of nodes having α degree in a weighted network
$\cap_{ij}$	Number of common neighbors of i, j
$\Gamma(i)$	1-neighborhood of node i
$\sigma_{i,j}$	Gossip spread factor of i by originator j
σ_i	Average gossip spread factor of i
σ	Average gossip spread factor of an unweighted network
σ_{k_α}	Average gossip spread factor of nodes having α degree in an unweighted network
μ	Mixing parameter of generated LFR networks

LIST OF ACRONYMS/ABBREVIATIONS

BA	Barabasi-Albert
CC	Clustering coefficient
ER	Erdős-Renyi
G-CN	Community detection using boundary nodes - using common neighbors with group approach
GN	Girvan-Newman
GT	Ground-truth
INF	Infomap algorithm
LFR	Lancichinetti-Fortunato-Radicchi
LPA	Label Propagation Algorithm
LPAc	Community detection algorithm of Xie and Syzmanski
LVN	Louvain community detection algorithm
MCMC	Monte-Carlo Markov chain
NM	Newman's algorithm
NMI	Normalized mutual information
PCN	Community detection using preference networks - with number of common neighbors score
PSC	Community detection using preference networks - with gossip spreading capability score
SBM	Stochastic block model
SNAP	Stanford network analysis project
WS	Watts-Strogatz

1. INTRODUCTION

1.1. Complex Systems and Networks

Euler's solution to the puzzle of Konigsberg's bridges in 1736 is one of the earliest use cases of graph theory for real-world problems [1]. Many researchers have been working on graphs to understand the theoretical and mathematical foundations of different systems represented by graphs and also graph theory by itself. In the last century, graphs are used extensively to represent a wide variety of real-world systems in various domains, i.e., social, biological, technological, economic or ecological systems. These systems are consisting of elements and relations between these elements. In graph or network representation, the elements of a system are denoted by nodes and their relations with each other by edges [2]. We can analyze different aspects of a system using different methods and tools on graphs. Mobile communication networks, scientific collaboration networks, patent networks, protein-protein interaction networks, and brain networks are examples of the network representation of corresponding systems [3–7]. *Complex networks* are the graph representations of the corresponding real-world *complex systems*; i.e., the brain network of the human body, friendship networks of people, airline traffic connection network among cities and protein-protein interaction networks. The special thing about complex networks is that the elements of the system interact with each other locally in a decentralized way and these interactions and positions of elements in the network lead to emergent behaviors, features and global patterns. The system as a whole exhibits an emergent behavior without explicit intention of individual elements (i.e., nodes).

We can create a network by creating nodes and wiring edges among them randomly in a simple way. However, this will not create a complex network. A complex network exhibits a structure or non-trivial property only when there exists some fundamental design principal of its formation. In order to understand a complex network better, one may compare it with its random null model. A null model proposed by Newman and Girvan [8] consists of a randomized version of the original graph, where

edges are rewired at random, under the constraint that the expected degree of each node matches the degree of the node in the original graph. A null model will not have most of the non-random properties of the original complex network.

1.2. Triangles, Information Diffusion and Gossip Spreading

Complex networks have network motifs as its building blocks and simple redundant connections among the nodes. The network transitivity of the nodes in complex networks is a clear deviation from the behavior of a random graph [9]. Transitivity can be defined as the increased probability of having a connection between two nodes when they are both connected to common node, i.e., when there is an edge between nodes i,j (i.e., e_{ij}) and nodes i,k (i.e., e_{ik}); then there is a high probability that j and k will create an edge (i.e., e_{jk}) and forms a triangle. Triangles or triadic closures are very common in many complex networks, especially in social networks; there exist many triangles between closely related groups of nodes while very few or no triangles exist between nodes of different groups [10]. Triangles are also important for social cohesion and information diffusion, i.e., when two friends of a person also know each other, then we can say there is a strong relationship between these three people [11].

Information diffusion is also easy and fast when there are many triangles in a network. Gossip is a special type of information diffusion. It is one of the oldest and most common means of information sharing among people. Gossip has been investigated by various disciplines including behavioral science, anthropology, psychology, sociology and complex networks [12–30]. Researchers try to figure out the underlying cause of gossip and its effect on society. There are many different definitions of gossip in society [12]. Unlike rumor [31], gossip is more personal and is assumed to be spread among people who know the *victim*, the person who is the subject of the gossip, in person [12]. Gossip can be spread if the victim, spreader and the receiver all connected to each other, i.e., they form a triangle. There are positive and negative gossips [13–15], which may have both positive and negative effects on individuals and the community [13, 21–23]. Informality and privacy are important conditions for the spread of the gossip [20]. Gossip is mostly perceived negatively by the victim [12, 13].

The victim tries to block the flow of the gossip about him [17–20]. Not only the victim but also his close friends try to stop gossip to protect the interests of both their friend and their group [16, 23, 24]. A gossipy conversation stops when a victim (or a relative or close associate of the victim) enters within the earshot.

Gossip can be used as a weapon to enforce social norms, but this will bring social costs to gossipers as well [22]. It tends to weaken already weak relations [13]. The victim may be hurt by seeing how other people deal with his personal life and how they manipulate information about private matters [23]. This may lead to retaliation; either by counter gossiping or breaking the relationship with the gossiper. People, who gossip about a person, will both have a bad reputation and harm the relationship with the victim [12, 22, 26]. It is less costly when spread by socially distant people [23] than closer people. As close friends would experience more harm in their relationship in a gossip situation, they would rather not gossip about their closer friends [13, 16, 23–25].

Gossip spreading is deeply analyzed in social sciences; however other research fields such as Computer Science have also some interest. In ad hoc computer networks, a routing algorithm, called gossip protocol, is inspired by gossip propagation in social systems [29]. There have been recent studies about gossip spreading in complex networks [27, 28]. Gossip spreading model, proposed by Lind *et al.* [27], is based on information spreading among the 1-neighbors of the victim. The model is based on the assumption that gossip is personal and people tend to spread gossip about people (i.e., victims) they know to their acquaintances who also know them (i.e., victims) in person. A friend who receives the gossip will become spreader and it will further gossip to its common friends. So, spreading occurs on triangles, and also on *triangle cascades* where we define it as a group of adjacent triangles on a vertex (i.e., victim). When the model is applied to social networks, it is observed that there exists a degree k_0 such that gossip spreading rate becomes minimum if the victim is of degree k_0 [27]. Similar results are obtained for networks generated by the Barabasi-Albert model.

We introduce a decision mechanism on top of the original gossip algorithm of Lind *et al.* and analyze the gossip spreading behavior on weighted complex networks [30]. Details of our work are in Chapter 3. We use gossip spreading and related metrics to discriminate social networks from other types of networks. A surprising finding is that, by just looking at gossip spreading behavior, we can group similar types of networks together, i.e., gossip spreading is more in social networks as they have many triangles and triangle cascades. We try to define a new metric to classify weighted complex networks using our model. The new metric is based on gossip spreading activity in the network, which is correlated with both topology and relative edge weights in the network. The model gives more insight into the weight distribution and correlation of topology with edge weights in a network. Gossip propagation is found to be a good indicator to distinguish co-occurrence and social pattern networks.

1.3. Community Structure and Community Detection in Networks

Gossip spreading or information diffusion is easier when there is clustering in a network, i.e., communities. Communities are the key emergent structures in complex networks. A *community* can be defined as a group of nodes in a network such that nodes in the same group have more connections with each other than with the nodes in the rest of the network [32]. It can be seen as a group of nodes sharing common properties or playing similar roles in a network. A community is also called a *cluster* or a *module*. *Community detection* is the task of identifying such groups in networks. It is one of the key areas in complex networks that has attracted great attention in the last decade and has many practical uses in various domains, i.e., social network analysis, recommendation systems, anomaly and fraud detection, brain function analysis, drug discovery, etc. There have been many community detection algorithms proposed so far [10, 32–45]. There is a comprehensive survey by Fortunato on community detection [46] and we will go into the details of different methods and algorithms in Chapter 2.

1.4. Local Approach for Community Detection

Although most of the community detection algorithms perform well on small networks consisting of hundreds or thousands of nodes; there are only a few of them that can run on very large networks of millions or billions of nodes, due to their time-complexity. With the availability of large network datasets, this becomes a critical challenge. If a community detection algorithm has to deal with the whole network during its execution steps or needs to optimize a global value (i.e., network modularity), it becomes computationally expensive to run this algorithm on large networks. Besides their large sizes, real-life networks are dynamic and evolve over time, i.e., structure and size can change while a community detection algorithm is still running on such a large network. Additionally, processing the whole network data may require storing and accessing it many times, which is expensive in terms of data storage, too. On such large networks, local community detection algorithms can perform well, where most of these algorithms have linear time-complexity with the network size [34–40]. Local community detection algorithms generally break down the community detection task of the whole network into subtasks, where each subtask handles a node or a small group of nodes. Subtasks identify the community of each node by using only the local information available around that node, i.e., its immediate 1-neighborhood. Completed subtasks are then merged together to get the community structure of the whole network. With the local approach, each subtask uses limited information and performs limited computation; this leads to overall low time-complexity of the algorithm and low execution times on very large networks. Note that, when network size gets larger, computation time for each subtask does not change; one should create more subtasks on a larger network. This scalability is possible by the distributed and parallel nature of the local approach where communities computed locally by each subtask. Besides their practicality, local algorithms may be only viable options on very large networks.

Local community detection is powerful for its speed and scalability. It also preserves the granularity of communities by avoiding the merge of small communities onto larger ones; locally available information is more important for the community detection subtasks. This will overcome the resolution limit problem [47] where smaller

communities are merged to larger ones to attain a global metric, i.e., most of the modularity optimization techniques have this problem. Being able to detect more granular structures and using local information, local algorithms can also identify subtle communities in a network; especially when nodes are connected loosely with each other. Many of the algorithms, that use global approach, tend to find a single large community on such networks.

1.5. Community Detection Using Preference Networks: A Meta-network Approach

By the definition of community in a network, nodes inside same the community have dense connections with each other than nodes in different communities. These connections are created as a result of underlying relations between these nodes, i.e., similarity of nodes, communications between them or their reachability to some further nodes, etc. So, the topology of a network reflects more information than simple connections between nodes. We can assume that connections in a complex network are formed either by strong connectivity (i.e., social cohesion) or to enable easy and redundant information diffusion. As discussed earlier, complex networks have non-random structures, building blocks or emergent structures compared to their null model having similar degree distribution. A simple building block of such structures is a triangle. Triangles are important especially for establishing strong relations (i.e., social cohesion) and redundant pathways for communication (i.e., information diffusion). So when we see triangles in a part of a network, we can assume that there is a clustering of nodes in there. Triangles are good indicators and building blocks of communities in complex networks. The number of triangles shared by two nodes shows how many common friends they have, i.e., triadic closures formed by two nodes and their common friends. When two nodes share a large number of triangles then it is an indicator of a similarity between them. Number of shared triangles, which in fact is the number of common neighbors of two nodes, can be used as a local information for community detection. Triangles also form triangle cascades, that enables indirect reachability to between neighbors of a common node who are not neighbors of each other. It may be also

useful to use number of triangle cascades as a similarity measure or as an indicator of enabler for information diffusion around a node. Using both triangles and triangle cascades as a means of clustering, we propose a novel community detection algorithm using preference networks [41]. Given a network, in order to identify communities, we ask each node to select their preferred node in their 1-neighborhood to be in same community with. Then we build a *preference network* with the same set of nodes of the original network and create directed edges using the answers of each node; such that there is a directed edge in preference network from answerer node to its preferred node. Preference network approach is a good example of emergence of communities from simple local decisions of each node. Each node makes a preference based on local information and the aggregated preferences will create connected components in the preference network, which we interpret as the communities of the original network.

In our algorithm, each node decides its preferred node based on two alternative measures it can obtain locally. Firstly, the number of triangles shared by two neighbors, that is actually the number of common neighbors they have, can be used as a similarity metric for each connected node pair. If two nodes have a large number of common neighbors, then we assume that they are highly similar and should be in the same community. So, if a node can make only one choice from its 1-neighborhood, which is the case in our algorithm, then it will prefer the neighbor having the largest number of common neighbors with it; i.e., to maximize the similarity within the community.

Second measure used to decide on the preferred node is based on both triangles and also triangle cascades shared between neighbors. This measure can be obtained using gossip spreading around the node who is about to make the preferred node decision. Gossip spreading rate, which shows the ratio of gossip spread among the friends of a victim node, is higher around a victim node that has many triangles in its 1-neighborhood. With the existence of triangle cascades around that node, this rate increases further, i.e., triangle cascades establish gossip pathways for the neighbors that are not connected directly but via intermediate triangles. In order to evaluate gossip spread rate around a node, we use gossip algorithm of Lind *et al.* [28] to spread gossip around that node and gossip spread rate. However, the selection of neighbor

for initiating the gossip, i.e., *originator node*, can change the spreading rate; i.e., each neighbor may have different spreading path, different position in 1-neighborhood of victim and different common neighbors with victim. For this reason, different originator nodes may result in different gossip spreading rate. So, each neighbor of a victim node is selected as the originator and we evaluate gossip spreading rate of that selection and set this value as the *spreading capability* of the corresponding neighbor in originator role. A high spreading capability of a neighbor is an indicator of a large number of common friends between the node and that neighbor, as well as other indirect friends reachable through triangle cascades. As these values are calculated for each neighbor of a node, the node makes the decision on its preferred node by choosing the neighbor having largest spreading capability about it. Details of the algorithm are in Chapter 4.

1.6. Community Detection Using Boundary Nodes

We investigate other local algorithms that have low time-complexity. One of these algorithms is the label propagation algorithm, denoted by LPA, that is proposed by Raghavan *et al.* [35]. It is a linear-time algorithm with iterations of label propagation among neighbors. In the initial state of the network, nodes have their unique community labels showing their community memberships. Then label propagation iteration starts, where at each step a node is selected at random and update its community label with the most common (i.e., popular) community label in its 1-neighborhood. Label propagation process is asynchronous, i.e., when a node is to decide its label in 1-neighborhood, those neighbors do not change their labels simultaneously. The algorithm terminates after a fixed number of iterations or when there are no possible label updates for any node in the network. In LPA, after a few iterations, as nodes update their labels many of the neighboring nodes start to have the same labels and this leads to clustering of nodes. Especially when there is a community structure in the network, these clusters enlarge and most of the nodes within these clusters are surrounded by nodes having the same labels. In such a situation, there is no chance of label change for these nodes if one intends to perform label propagation for them, i.e., the node and all of its neighbors have the same label. However, in LPA, the algorithm tries to perform

label change or update for all the nodes, including the ones inside clustered regions. So, there is a potential performance improvement in the algorithm by reducing these unnecessary operations.

We propose a new local community detection algorithm, that finds communities by identifying borderlines between them using boundary nodes [42]. A node is defined as an *interior node* if all of its neighbors belong to the same community with the node. A node that is not an interior node is called a *boundary node*, i.e., having one or more neighbors from a different community. Initially, all the nodes in the network have unique labels and form communities of size one, i.e., they are the only members. There may be methods to decrease this number i.e., initially putting similar neighbors into the same community, in order to eliminate unnecessary operations on merging of microclusters. The network will initially have many boundary nodes. Community detection process naturally decreases their numbers by identifying communities of them. In the final situation, only the actual boundary nodes remain, and they constitute the borderlines between communities. Our algorithm eliminates unnecessary label propagations in LPA and decreases the overall execution time. As the second enhancement over LPA, it introduces a more comprehensive decision mechanism on which label to take during label propagation, i.e., each node decides among the options in its 1-neighborhood according to the largest "benefit score" exhibited by neighbors to attract the node to their communities. The measures used to calculate "benefit scores" and decisions based on these scores should lead to good community structure. We try different measures as benefit and approaches to calculate the "benefit scores" based on these benefits. We compare the results and conclude that using number of common neighbors in 1-neighborhood as benefits and making label update decisions accordingly yield the best results for community detection. Similar to other local algorithms, our algorithm preserves small communities as well as big ones and can outperform other algorithms in terms of quality of the identified communities. Especially when the community structure is subtle, it is better than the original LPA and other algorithms, where LPA generally puts all the nodes in a single community. Our algorithm has a distributed and parallel nature and can be used on large networks. Details of the algorithm are in Chapter 5.

1.7. A Framework and Toolset for Community Detection

During the process of defining new community detection algorithms, we have to deal with many tasks, that take a lot of our time and distract our attention from the main task of defining new community detection algorithm, i.e., trying to find network datasets, implementing or integrating the known algorithms, comparison of different algorithms, generating network datasets, unifying input formats for different algorithms, consistent outputs from different algorithms and comparison of partitions, etc. Most of these tasks are repeated many times and they are error-prone because of the manual processing. When a researcher is working on defining a new community detection algorithm, with each small change in the code or a change in a parameter in the algorithm, he needs to perform many of the repetitive tasks over and over again.

In order to speed up the development and testing process of defining a new community algorithm and make these tasks automated and error-free, we build a software framework, where a researcher can use many existing network datasets, generate new datasets, compare his newly developed algorithm with some of the known algorithms and compare partitions using known metrics. Our work provides a structured, end-to-end and robust framework, with necessary toolset and software, for everyone who needs to focus on his own algorithm, but also needs other facilities of network analysis and community detection. We provide modular tools so that different network formats can be ingested for the use of different algorithms. A new algorithm can be incorporated into the framework easily. Different metrics and methods are provided for network analysis as well as the comparison of community detection algorithms within the framework. This framework saves time during the development and testing of our new algorithms and we provide it freely to other interested researchers working in this area. Details of the framework are in Chapter 6.

2. COMMUNITY DETECTION IN COMPLEX NETWORKS

2.1. Introduction

Community detection is one of the key areas in complex network research to understand the structure of complex networks. Researchers extensively work in this field and many algorithms have been proposed so far. There is a comprehensive literature survey by Fortunato [46] on community detection in complex networks. Most of the clustering problems are NP-hard [46], as there are many possible combinations for the defining a partition that represents the community structure of a network. It is primarily an optimization task in a large search space, where the objective is to find the "best" partition of the community structure. In general, the number of communities or the boundaries among communities is unknown in advance.

A community can be seen as a dense group of nodes in terms of connectivity. One should expect to see communities in a network if its connections are not wired with a random process. However, there are other views on the definition of a community, i.e., nodes having similar functions, nodes with more similarities or blocks of nodes with similar connection patterns, etc. Different aspects and purposes of community detection are analyzed in a recent work by Schaub *et al.* [48]. They discuss that understanding the motivation of community detection for a specific problem is important to select the most suitable algorithm or approach since there are many facets of community detection. It is also important to understand the motivations behind the community detection task when a researcher proposes a new community detection algorithm, i.e., a hierarchical system like human body may be better analyzed using a hierarchical community detection algorithm. The problem domain, size and the structure of the network are also important in community detection. In large dynamic peer-to-peer networks, dynamic routing is important and there is no centrally maintained routing table. In such a case, the grouping of nodes into clusters locally around each computer

will do the job and enables efficient routing, while a global approach will not be a good fit for this purpose, i.e., processing the entire network to produce results will be time-consuming and computationally expensive. Community detection is important not only for identifying the groups, but also finding the boundaries between them; as these boundaries play an important role in exchanging ideas in a group, diffusion of information in a social network or spreading of a disease in the human body. It is also critical to protect a module, community or cluster from a damage from outsiders using this boundary information, i.e., nodes on these boundaries constitute the frontlines. In social networks, an inter-modular node, who is sitting in a critical position in terms of connecting many nodes from different communities, is defined as a "stranger" by Simmel, a century ago [49]. A stranger is sitting between different groups and it is critical in communication between them; it does not belong to any group. The stranger can benefit from its *in-between* position by controlling the communication passing on it. Burt [50] defines "structural holes" as the non-redundantly positioned nodes between different groups, who enjoy being in a central position in the communication. He argues that innovators and successful managers occupy the structural holes in their networks. Additionally, he states that a strong relationship indicates the absence of structural holes, that is consistent with the assumption that social cohesion needs redundant connections, i.e., there exists many triangles and triadic closures in communities. Using similar approach, Csermely [51] works on identifying active protein centers using a network-based approach and argues that behavior of creative persons in networks corresponds to the behavior of amino acids of active centers, which are called "creative elements".

2.2. A Brief History

Community detection in complex networks is popular in recent years, however researchers started working on it several decades ago. One of the early efforts of community detection was carried out by Rice [52] in 1927, by looking for *blocs* or clusters of people in small legislative bodies based on their voting patterns. Later on, in 1941, Davis *et al.* [53] carried a study, named as *Deep South*, that investigates the social

activities of people in a small city and its surrounding of county of Mississippi in order to understand the corresponding communities. They introduced the concept of *caste*, where a person is assigned a caste at birth, maintains that status throughout his life and passes it to his children. They showed that, caste is a discriminative feature to identify communities, but they need further subdivisions inside communities using social classes. Then, in 1947, Homans [54] showed that social groups should be identified by rearranging the rows and columns of matrix describing the social ties, i.e., adjacency matrices of nodes in a network, until the matrix is in an approximate block-diagonal form. This procedure can be seen as the first attempt of *block-model* approaches for community detection. In 1955, Weiss and Jacobson [55] looked for the communities of individuals in a government agency, by analyzing the data gathered with private interviews of 196 members of that agency. They analyzed the matrix of working relations, built by the answers of interviews; i.e., frequency of their contacts with each other, reason for contact, subject matter discussed and the relative importance of contact. Workgroups were identified by removing the members working with different groups in the network data, i.e., liaison persons, who play central role in connecting different workgroups. The idea of separating networks by cutting inter-group nodes is the basis of many recent community detection algorithms. This approach also uses the notion of betweenness centrality for cut decisions. The karate club study by anthropologist Zachary [56] is one of the most popular community detection analysis, which identifies the communities in a network of 34 nodes representing the members of a karate club. The members of the club is divided into two over a dispute and network dataset contains the ground-truth communities, which makes the network a popular benchmark dataset for community detection algorithms. There are other studies on various networks. Bech and Atalay [57] worked on network analysis of loans among financial institutions to understand how system health is affected by the communities and their members. Social behavior of bottlenose dolphins of the New Zealand is studied by Lusseau *et al.* [58] and the network dataset of 62 dolphins is famously known as *dolphin network*. In their study, authors observe stable small groups of dolphins, i.e., communities, which contains members from different sexes and members in same community are seen together more often than expected by random chance.

2.3. Use Cases of Community Detection

With the advent of the Internet and high adoption of mobile devices, there are many social and technological networks that are being investigated with community detection approach. In World Wide Web context, a practical use case for community detection is to identify users with similar interests and are geographically near to each other so that they get better web experience with a dedicated proxy server. Identifying cores or border nodes of communities is critical for routing protocols in ad-hoc networks or peer-to-peer networks. With the explosively increasing amount of content generated by connected users or devices, there is a need to understand such a huge amount of data using different techniques. Especially, on the networks of blogs, user reviews, recommendations and multimedia sharing sites, community detection can be used to enable new insights and actions. Identification of influential nodes can give an idea on how to diffuse a commercial or a sample product with a limited budget, i.e., selecting only influential ones. Community detection can be used as a basis to increase the spread of an idea to as many nodes and communities as possible. Conversely, it can be used for containment of a disease within a restricted group or area. Identifying groups or communities can also lead to more insights revealed by the similarity or functioning of nodes in the same community. Many social platforms try to find the best method to suggest new connections to its users, i.e., new friends on a social network, new connections on a job network, etc., in order to increase the strength of the network on their platform with more connections among people. With the help of these additional connections, people may possibly spend more time on these platforms. Using community detection, one can identify the *missing links* between nodes in a community and suggest it to the user [59, 60]. Link prediction of link suggestion is somehow similar to product recommender systems of various online platforms. Using the same principle of friend or link suggestions in social networks, community detection can be used to group similar products or users to make recommendations about new products to existing users [61, 62], i.e., *collaborative filtering*. Sahebi and Cohen [63] proposed a community detection based approach to overcome the *cold start* problem of a recommender system by finding latent communities of users. Recommender systems can also target the groups or communities with similar tastes [64]. There are many other

recommender systems based on community detection in various application areas [65–67].

Community detection has practical use cases in identifying similar functioning elements in a system. Given the community structure of a network and a known member from a community, identification of other existing or latent members of the community can be useful in drug discovery or protein-protein interactions. Udrescu *et al.* [68] used network analysis and community detection on drug-drug interaction networks to group similar drugs. Using only the in-silico analysis, they successfully identified groups of interacting drugs and the related types of diseases. Their results are useful in better understanding the drug-drug interactions and drug repurposing, i.e., using existing drugs for new purposes. Drugs that topologically lie at the border between two communities may have pharmacological properties of both neighboring communities, so one can further analyze the usage of that drug with a new purpose. Predicting possible drug-drug interactions is also crucial, as it can happen unexpectedly in the human body when more than one drugs are co-prescribed, causing serious side effects. Zhang *et al.* [69] used community detection with label propagation (LPA) in order to identify possible drug-drug interactions. Nacher and Schwartz [70] analyzed the modular structures in the network of the protein complex and their drug interactions, that potentially uncovers the relationship between protein complexes in human body and diseases. Proteins interact with each other and define protein complexes; these complexes have a rich variety of functions in cells and play a key role in many human disorders. Their analysis unveils new associations between diseases and protein complexes and potential discovery of new drugs that need to combat complex diseases.

Another practical use of community detection in health domain is to predict mutated or risky genes, i.e., latent damaged genes appear in the same community as some other known risky genes. Genes causing the same or similar diseases often lie close to one another in a protein-protein interaction network [71]. Shen *et al.* [72] use community detection approach to discover similar proteins in protein complexes. Cantini *et al.* [73] used multiple networks, i.e., protein-protein interaction networks, gene co-expression, microRNA co-targeting, etc. and analyzed their corresponding

communities to uncover new cancer driver genes.

Identifying members of a community gives an idea about the similarity between these members. As it is useful to use it in protein or drug discovery in biological networks, same principle can be applied to financial risks of companies or banks by looking at their community relationships with other risky companies or individuals; where the network is consisting of corporations and individuals and their relations, i.e., corporations having shares of another corporation, individuals taking roles in different corporations, etc. Bargigli and Gallegati [74] identified the communities in financial networks to better understand the potential source of a systemic risk in the financial sector. Guerra *et al.* [75] analyze the systemic risk with network analysis of multiple factors and clusters of banks and different entities in a financial network. Their method helps to identify "important" banks in the system in terms of the risk they can cause and gives the insight to avoid a systemic risk in the financial system.

Fraudulent activities in several domains can also be identified using community detection on graphs. A fraud is actually an *anomaly*, that is an unexpected event or behavior by an element or a group of elements in the system. Anomaly detection techniques can be used to detect anomalies or frauds in complex environments. Compared to anomaly detection methods focusing on the attributes of individuals, studying the relational aspects of individuals with the help of networks and communities, provides a more comprehensive perspective for anomaly detection. One can use community detection as an outlier detection method, i.e., nodes outside communities can be seen as anomalous; or it can be used to identify normal or anomalous nodes or instances, according to the community they belong, i.e., first identify anomalous communities and then mark a node or instance as anomalous if they belong to these communities. Liu *et al.* [76] use network analysis to detect fraud activities on healthcare data based on communities. They identify anomalous communities based on community size, features of nodes inside those communities and interaction between communities. There is an extensive survey by Akoglu *et al.* [77] on the use of networks and communities for anomaly detection.

2.4. Approaches in Community Detection

We investigate the examples from various domains to understand how community detection can be used to better understand a system and its structure with a comprehensive view. Many community detection algorithms are proposed to enable such analysis. We now give an overview of different approaches and methods for community detection algorithms. There are various community detection algorithms using different approaches. Some of them are divisive/cut-based clustering, modularity optimization, statistical inference, label propagation, spectral clustering, evolutionary methods, information theoretic approaches, topology related methods, and artificial intelligence.

2.4.1. Community Detection in Static and Dynamic Networks

In most of the early examples of community detection algorithms, networks are seen as static structures; i.e., nodes and edges do not evolve over time. This simplistic view is the basis of many community detection algorithms which have successful results. Some examples are [10, 32–37, 43–45]. As long as there are no changes in the network structure, these algorithms on static networks can be used effectively. However, most of the social and real-life networks are dynamic, i.e., nodes and edges inserted, deleted over time. With the high adoption of mobile devices, computers and many other connected devices, there are many new dynamic social and technological networks, that are very large and change rapidly. Many researchers investigate the dynamic changes in networks and how it affects the community structure [78–85].

Communities in a dynamic or evolving network can be analyzed by slicing the network into many snapshots at different timestamps, where each slice is a static network representing the dynamic network as of the snapshot time. Algorithms using this approach [78–82], first analyze the slices of different snapshots and then monitor the evolution of each community. These algorithms can be very time-consuming on the networks changing rapidly and when time slices are very small. It is also possible that they are not able to capture the evolution of community structure over time, as it is

hard to match the same community from two different snapshots of the network. A better approach should be to adaptively update community structure, only on the parts of the network where modifications occur. Nguyen *et al.* [83] use a local modularity-based approach that is working on the parts of the network where changes happen. Cazabet *et al.* [84] propose the iLCD algorithm, that uses the succession of structural changes in a dynamic network. Han *et al.* [85] use an adaptive label propagation algorithm on small portions of the network, where changes take place. Their approach is local and needs less computation time.

2.4.2. Overlapping Communities

Community detection algorithms have differences on how they handle the edge weights or direction of edges of the network they are working on. At the minimum, most of the algorithms work on unweighted and undirected networks. However, some algorithms may need edge weights or directed edges to uncover the communities. The uniqueness of community membership is also treated differently by some algorithms. A community detection algorithm is defined as *overlapping* when it allows a node to be members of multiple communities and *non-overlapping* otherwise. Many real-life social networks are overlapping; i.e., a personal contact can be both a friend from school and a colleague in a workplace. On such networks, overlapping community detection algorithms [86–89] can produce better results.

2.5. Community Detection Techniques

Community detection is not a well-defined problem and there is not a universally accepted definition of a community. Therefore, different algorithms use different heuristics and approaches. It is possible to structurally identify the clusters or communities if the network is sparse. However, if the number of edges is larger than the number of nodes in a network, then data clustering [90] techniques may help. In data clustering, communities are seen as sets of nodes, that are close to each other according to some similarity metric. Some classical techniques in data clustering are hierarchical, partitional and spectral clustering. Other data clustering techniques include techniques

like self-organizing maps and singular value decomposition and principal component analysis.

2.5.1. Traditional Methods

The problem of graph partitioning consists of dividing the nodes into g groups of predefined size, such that the number of edges between groups is minimum. Though it needs prior knowledge of the number of communities or clusters, it has practical use cases in areas like parallel computing, circuit layout design, etc. Kernighan-Lin algorithm [91] is one of the earliest and most used algorithms for graph partitioning, that is devised for partitioning electronic circuits on boards with the least possible connection between different boards. Another popular technique is the spectral bi-section method [92], that is based on the properties of the spectrum of the Laplacian matrix. Max-flow min-cut theorem, proposed by Ford and Fulkerson [93], investigates the minimum set of edges, that will cut the maximum flow by their removals. In this context, edges can be seen as carriers of a flow; i.e., water, electricity, etc., and they have a capacity. There are several efficient routines to compute maximum flows in networks [94, 95]. Algorithms for graph partitioning is not a good fit for community detection, because it is necessary to provide as input the size or number of groups. It is practically not possible for most of the cases in community detection.

2.5.2. Divisive Approach

One of the early examples of community detection algorithms, Girvan-Newman (GN) algorithm [32], is based on a divisive approach. It is a hierarchical divisive algorithm, that tries to maximize the segregation of network with the removal of edges. The algorithm iteratively computes shortest paths between each pair of nodes in the network and identifies how many shortest paths pass on each edge in the network, that defines the *edge betweenness* of the edge. Iteratively, the GN algorithm cuts the edge with largest edge betweenness; i.e., this cut will disconnect as many nodes as possible in the network. After each cut, algorithm re-evaluates all the shortest paths in the new configuration and identifies the next edge to cut in the same way, until

it cuts all the edges. In the end, the algorithm builds a hierarchical structure, a *dendrogram*, regarding the order of edge cuts. Communities can be obtained by using the dendrogram, by cutting it at a proper level. It is intuitive that intercommunity edges have a large value of the edge betweenness because of the fact that they are part of many shortest paths, that connect vertices from different communities. GN algorithm can successfully identify hierarchical communities in networks, however, with its $O(n^3)$ time-complexity, GN algorithm is not a good fit for large networks, where the number of nodes, n, is very large. Rattigan *et al.* [96] proposed a faster version of GN algorithm with a quick approximation of the edge betweenness, where the network is divided into regions and each node's distance to these regions are computed. Holme *et al.* [97] use a different divisive approach for community detection; they remove nodes rather than edges. A centrality measure for the nodes proportional to their site betweenness and inverse proportional to their indegree is used to identify boundary nodes and they are removed iteratively. Their algorithm is applied to biochemical networks to get a hierarchical view of the underlying system.

Another divisive approach [10], that is based on finding intercommunity edges, uses the triangles of triadic closures in networks. Communities are characterized by dense connections between its members, so one should expect many redundant connections between the pair of nodes, i.e., triangles. Contrarily, a node sitting in between different communities will not have many triangles compared to a node inside a community. Radicchi *et al.* [10] proposed a new measure, edge clustering coefficient, such that low values of the measure indicates intercommunity edges.

2.5.3. Modularity Optimization

Communities or modules are hard to define and quantify. Network modularity, proposed by Newman and Girvan [8], is one measure to understand the strength of the division of a network into modules. Modularity is the fraction of edges, that fall within the given communities, minus the expected fraction if edges were distributed at random. It reflects how good a community partition represents actual community structure compared to the null model. The null model of the network consists of

a randomized version of the original network, where edges are rewired at random under the constraint that the expected degree of each vertex matches the degree of the vertex in the original graph. The value of modularity lies between -1.0 and 1.0, a positive modularity is expected when there is clustering in a network. There are many algorithms based on modularity optimization; i.e., they try to find the optimum partition that provides the highest modularity. Finding the desired partition is an optimization process, where many different techniques can be used. For modularity optimization, one of the well-known approaches is the hierarchical or agglomerative approach.

2.5.4. Agglomerative Approach

Newman [33] proposed a community detection algorithm that tries to find communities by optimizing the network modularity and it builds a hierarchical community structure (i.e., dendrogram) in bottom-up fashion using an agglomerative approach. Starting with an initial state, where each node is in its community, algorithm repeatedly joins communities together in pairs by choosing the "best" join, that yields highest gain in network modularity. When all joins are completed and dendrogram is built, communities can be obtained by cutting it at an ideal point where modularity is maximum. Although this algorithm has better time-complexity compared to GN algorithm, its time-complexity is still high due to its global community detection approach; it has very long execution times on large networks. Clauset *et al.* [98] proposed an improvement over the algorithm in ref [33] by eliminating a large number of unnecessary operations on sparse matrices with the introduction of max-heaps as the data structure in the algorithm. This greedy optimization is one of the few algorithms that can be used for modularity optimization on large networks; we call this version of the algorithm as Newman's algorithm (NM). Modularity optimization tends to form large communities at the expense of small ones. The greedy optimization of modularity is the cause of the problem known as *resolution limit* [47]. Danon *et al.* [99] suggested normalizing modularity gain produced by the join of two communities, by the fraction of edges incident to one of the two communities with the purpose of favoring small

communities. Their approach yields better results on networks formed by communities of very different sizes. Modularity maximization method of Clauset *et al.* may create unbalanced dendrograms. Wakita and Tsurumi [100] proposed a method to balance the gain of modularity during the merge of communities with the consolidation ratio, that favors equal size communities. Their method both improved the speed of method proposed by Clauset *et al.* and the accuracy of the communities found on large social networks. The hierarchical agglomerative approach can be improved if the procedure starts from some reasonable intermediate configuration, rather than individual nodes. Proposed improvements [101, 102] suggest configurations based on initially grouping nodes into subgraphs using similarities of nodes and topology.

Blondel *et al.* [36] proposed a different greedy approach, known as *Louvain* (Lvn) algorithm, that identifies the communities in a hierarchical manner by iteratively replacing the nodes in the original network by supernodes created by merging nodes into supernodes with highest modularity gain. Although it has speed improvement, it still has a resolution limit problem. In general, methods based on greedy optimization of modularity have low accuracy in identifying the correct community structure. Especially in networks consisting of communities of different sizes, these methods have less accurate results.

2.5.5. Partitional Clustering

Partitional clustering algorithms try to find the preassigned k number of clusters, such that dissimilar nodes are separated as much as possible according to the given distance metric. The most popular partitioning algorithm is the *k-means* clustering [103], which tries to find k *centroids*, such that they are as far as possible from each other and nodes are assigned to the nearest centroid. There are many other routines using different approaches like *k-nn*, *k-median*, *k-center*, etc.

2.5.6. Spectral Clustering

In spectral clustering, the data points of a network, i.e., nodes, are represented in a similarity matrix, that is used to embed the nodes to a low-dimensional space (spectral embedding) using eigenvectors of the similarity matrix. Graph Laplacian matrix, L, is used as the similarity matrix of nodes, that can be obtained by $L = D - A$, where D is the degree matrix of nodes and A is the adjacency matrix. First k eigenvectors of Laplacian matrix L, corresponding to k smallest eigenvalues, will give the k possible centroids for clusters, i.e., spectral embedding. With a clustering algorithm like k-means, this embedding is used to partition the network. There are various spectral clustering algorithms; recursive bi-partitioning of Hagen and Kahng [104], clustering with multiple eigenvectors of Shi and Malik [105], etc.

2.5.7. Evolutionary Algorithms

One possible community detection approach can be to propose many solution alternatives, i.e., many different partitions of the network as community representations. Then, one can evaluate a fitness value of each of the proposed partition, that measures how well the partition represents the actual community structure of the given network. This process can be implemented using an evolutionary algorithm, namely genetic algorithm. It was proposed in our early work during Master Thesis, that is, community detection in complex networks using genetic algorithms [45]. In this approach, we propose many solution candidates, i.e., different community partitions, evaluate their fitness for the solution and try to create better solutions by reproduction of them in next generations. Let us assume the solutions candidates as individuals in a population. Our purpose is to create better and better individuals for each new generation and stop reproduction when we have a sufficiently good solution candidate in the population. This is similar to reproduction in nature and we use chromosome representation for expression each solution candidate, i.e., each proposed partition of the network is encoded in a chromosome. Initially, proposed solutions can be random partitions or partitions proposed by using some heuristic or taken from another algorithm's intermediate results, so we initialize chromosomes accordingly. General practice is to initialize

them randomly. We need a fitness function to evaluate how good a chromosome (i.e., its proposed solution encoded within), can solve the community detection problem. We decide to use network modularity as fitness function, i.e., given a chromosome c, we calculate the network modularity of the partition it encodes and set this value, f_c, as its fitness value. The members of the population (i.e., chromosomes) are sorted by their fitness values, f_i, and they mate with each other accordingly, i.e., chromosomes with higher fitness values will mate each other. With this mating, the algorithm performs reproduction by creating new chromosomes (i.e., children) from existing ones (i.e., parents) with the aim of improving the quality (i.e., fitness value) of the population for the next generation. To achieve this, genetic functions like cross-over and mutation are used during reproduction, that are mechanisms of biology creating diversity in nature. The reproduction process should be repeated many times to have better quality chromosomes in the population for the next generations and this will improve the proposed solutions. We use different techniques of genetic operations as detailed in [45] and have successful results in identifying community structure with this evolutionary approach of artificial intelligence.

2.5.8. Information Theoretic Approach

For community detection task, researchers use different approaches; one such approach is using information theory for community detection. Infomap algorithm (Inf), which is proposed by Rosvall and Bergstrom [34], identifies the modules of the network by finding the optimal compression of its topology using the regularities in its structure. The algorithm tries to find the best community partition among many possible ones, based on the information theory. Given a network X, the algorithm tries to find a simpler description Y, that maximizes the mutual information, $I(X;Y)$, between the description and the network. Since it tries to find the communities, it will enumerate communities of X in Y using encoders. Idea is to represent the largest group of nodes with the minimum number of bits, where each group has its own bit string. It uses random walks to discover the optimum coding and using this coding; it performs lossless compression for describing the communities in the network, where a decoder can

reveal the actual network. It is a very efficient algorithm with good results in community detection. Especially, on the networks consisting of communities with different sizes, it performs better than modularity based approaches, i.e., in modularity based algorithms, the size of the module depends on the network size.

2.5.9. Block Models

Block modeling is the decomposition of a graph into classes of vertices with common properties. Nodes are generally grouped in classes of topological equivalence, i.e., structural or regular equivalence. *Structural equivalence* forms classes of nodes such that nodes within a class have same neighbors. In a more relaxed way, *regular equivalence* defines the classes such that nodes within a class have similar connection patterns to nodes of other classes. Structural equivalence can be used to group nodes in a probabilistic model such that linking probabilities of a node to all other nodes are same for nodes in the same class, i.e., stochastically equivalent. A stochastic block-model (SBM) [106] is a generative model for random graphs, that aims to produce the communities for the given particular edge densities. In SBM, nodes are partitioned into $K < n$ disjoint communities according to some latent random mechanism. The edges occur independently with probabilities, depending only on the community memberships of the nodes, such that, nodes within same community or *block* will have a high density of connections compared to other nodes in other blocks. The goal of community detection using stochastic block model is to recover latent community structure of a given network. Once the model variables are known, it is easy to build the network with blocks. However, in community detection, we have the network dataset and need to infer the parameters that generated it; that is basically a parameter optimization. Algorithms based on SBM uses different machine learning techniques for such an optimization, i.e., Monte-Carlo Markov Chain (MCMC), belief propagation, cavity method, expectation maximization, etc. Spectral clustering is also used to identify communities in SBM [107]. There are several community detection algorithms that use SBM [108–110].

2.5.10. Local Approach

Community formation is something local by nature. But algorithms to detect communities may use global information rather than local information. For example, GN algorithm [32] iterates over the network. In each iteration, it calculates the number of shortest paths passing through each edge and removes the edge with the largest number. Such calculations require information of the entire network. This global approach, which is fine for networks of small sizes, is not feasible for very large networks.

With the introduction of many large and dynamic networks in real life in recent years, time-complexity of community detection algorithms becomes more critical. In order to work on such large and dynamic networks, we need distributed and scalable community detection algorithms. Community detection algorithms with local approach can be good candidates to satisfy the needs. These algorithms generally divide the community detection task into subtasks and perform each subtask on different parts of the network in a distributed and parallel fashion. In recent years several local community detection algorithms have been proposed [34, 35, 37–43].

To be able to run in parallel on different parts of the network, subtasks should not enqueue on a global resource, i.e., a global metric or information, to carry their task. For this reason, each subtask should use only locally available information around a node or a group of nodes to identify their communities. Information used by local community detection algorithms is generally easy to obtain or compute, i.e., 1-neighborhood of a node. Some algorithms merge nodes into communities based on the optimization of a local metric [40]. This is the real advantage of local algorithms, that makes them distributed and scalable.

Raghavan *et al.* [35] proposed label propagation algorithm, denoted by *LPA*, which updates community label of a node with the most common label in its 1-neighborhood, i.e., majority rule of labels. Labels of all nodes in the network are updated asynchronously and the algorithm terminates when there are no possible label updates in the network. It is a linear-time algorithm, that can identify communities

in a fast way. However, it tends to find a single large community, especially when community structure is subtle.

Xie and Szymanski [43] proposed an enhancement over LPA, which we denote by *LPAc*, using neighborhood-strength driven approach. LPAc improves the quality of identified communities by incorporating the number of common neighbors to the majority rule of labels in LPA, i.e., a node decides which label to take based on scores of communities in its 1-neighborhood. The score of a community is calculated by its population size in node's 1-neighborhood as well as the number of common friends its members have with the node multiplied with a constant, $c < 1$. Another major difference is that the algorithm does not perform label propagation for all the nodes but only for active boundary nodes. A node who has one or more neighbors from different communities is called a *boundary node*, otherwise, it is an *interior node*. The algorithm defines a *passive* node as a node that would not change its label if there is an attempt to update it, and a node that is not passive is called *active*. It keeps a list of these nodes and performs label propagation only on active boundary nodes. Algorithm iteratively selects a node i from the active boundary list and updates its label, $L(i)$. After the label update, status of node i is checked and if it becomes a passive or interior node, it is removed from active boundary list. After label update, neighbors of i are checked for a change of status, i.e., if they become active boundary nodes, they are inserted into the list, if they change from active to passive, they are removed from the list. The algorithm iteratively identifies the labels of nodes in active boundary list and maintains the list with removals and insertions of nodes during label updates. Algorithm completes when the active boundary list is empty. The algorithm has improvements over LPA, in terms of the quality of identified communities. However, it has longer execution times; in some cases, it has also a convergence problem, i.e., some nodes are removed from and reinserted to active boundary list and algorithm could not finish due to the non-empty list. LPAc still has the issue of finding a single community, which is a drawback of LPA, too.

We also propose an enhancement over LPA, namely, community detection using boundary nodes in complex networks [42]. Our approach also performs label propagation only on boundary nodes but with a major difference. When we pick a boundary node from boundary list and perform label propagation for it, we immediately remove it, regardless of its status after label propagation. This eliminates the convergence problem and saves many control routines during label propagation. Overall time-complexity is better in our algorithm. We also incorporate a new decision mechanism based on the number of common neighbors, that leads to improved quality of identified communities. Details of our proposed algorithm are in Chapter 5.

In recent years, many other local algorithms are proposed using different approaches. As mentioned earlier, real-world networks become very large and dynamic. On such networks, global approaches cannot perform due to their high time-complexity. Most of these algorithms are also incapable of analyzing the dynamic structure of these networks. Even if we overcome the long running time of a global community detection algorithm, computation of a global metric is difficult as the network is dynamic and evolves over time. In such a situation, a distributed and parallel community detection approach can be the solution, i.e., community detection is distributed as parallel subtasks on different parts of the network that can be run on dynamic, large networks. Such an algorithm can use local information available around a node, i.e., algorithm can handle what it already has (i.e., a portion of the network at a certain time) and can continue with what will come later; it does not need the snapshot of the whole network at once.

We propose a local algorithm, namely, community detection using preference networks [41]. In our algorithm, each node decides its community according to the neighbor with the highest number of common friends or highest gossip spreading capability. In the algorithm, community memberships are not represented by labels; instead, a preference network built by the preference of each node for being in the same community with. Then the connected components in the preference network are treated as communities of the original network. Details of the algorithm are provided in Chapter 4.

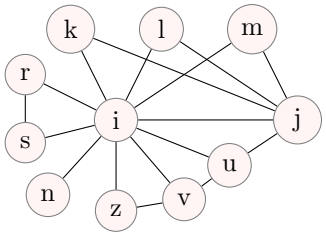


Figure 2.1. Clustering coefficient, common neighbors and triangle cascades.

2.6. Triangles, Communities and Gossip Spreading

In his book, Simmel [11] argued that a strong social tie could not exist without being part of a triangle in a relation, i.e., relation among three people where all know each other. People who have common friends are more likely to create friendships; they form triangles. There is a correlation between triangles and communities in social networks; there exist many triangles within communities while very few or no triangle exists between nodes of different communities [10]. Triangle is the smallest cycle of size three. There are studies that investigate cycles of size four or more [111] but we focus on triangles in our studies.

Clustering coefficient (CC), is equal to the probability that two nodes that are both neighbors of the same third node will be neighbors of one another [112]. This metric shows the number of existing triangles around a node compared to the all possible triangles. A high clustering coefficient will mean many triangles and clustering around a node;

$$CC(i) = \frac{\triangle_i}{\wedge_i}$$

where $\triangle_i$ is the number of triangles around node i and $\wedge_i$ is the number of triplets where i is in the center. A triplet is formed by three nodes and two edges [112] . For example, in Figure 2.1, i, r, s form a triangle where s, i, n make a triplet.

Radicchi *et al.* [10] proposed a community detection algorithm based on triangles and clustering coefficient. In recent years, local community detection algorithms are more focused on the similarity of nodes; especially on the local level, i.e., 1-neighborhood of nodes. A node similarity-based approach is applied to several known community detection algorithms by Xiang *et al.* [113], where they presented the achieved improvements on those algorithms by using various node similarity metrics. Some examples of those similarity metrics are the number of common neighbors and Jaccard similarity [114].

The *number of common neighbors* of nodes i and j is given as

$$\cap_{ij} = |\Gamma(i) \cap \Gamma(j)| \tag{2.1}$$

where $\Gamma(i)$ is the *1-neighborhood* of i, i.e., the set of nodes whose distances to i are one. The number of common neighbors shows the number of triangles formed on two nodes, i.e., in Figure 2.1, four triangles are formed on i and j by their common neighbors k, ℓ, m, and u.

Jaccard similarity is the fraction of common neighbors of i and j to the union of their 1-neighborhoods given as

$$J(i,j) = \frac{|\Gamma(i) \cap \Gamma(j)|}{|\Gamma(i) \cup \Gamma(j)|}. \tag{2.2}$$

All of these metrics are related to friendship transitivity and triangles.

A new metric, spread capability, is proposed as a similarity metric in our work [41]. This metric is calculated by using the gossip algorithm of Lind *et al.* [28]. A gossip about a *victim* node i is initiated by one of its neighbors, node j (*originator*), and j spreads the gossip to common friends with i, i.e., gossip about i is meaningful to friends of i only. Nodes hearing the gossip from j behave the same way and propagate it further in the 1-neighborhood of i until no further spread is possible. To measure how effectively the gossip is spread, they calculate spread factor of the victim i by

originator j as

$$\sigma_{i,j} = \frac{|\Gamma_j(i)|}{|\Gamma(i)|} \tag{2.3}$$

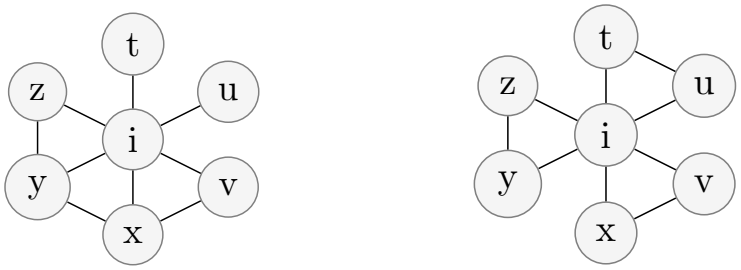
where $\Gamma_j(i)$ is the set of neighbors of i who heard the gossip originated by j. Lind *et al.* calculated the spread factors of each originator $j \in \Gamma(i)$ and averaged them to get *spread factor* of i, i.e.,

$$\sigma_i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} \sigma_{i,j}. \tag{2.4}$$

We use the spread factor in a different way in our algorithm; instead of average gossip spread factor of a node, i.e., σ_i, we focus on $\sigma_{i,j}$ values, which show the contribution of each originator j to that average. We call $\sigma_{i,j}$ as *spread capability* of j around i. Spread capability is directly related with the connectivity of j and its position in the neighborhood of i. So, each $j \in \Gamma(i)$ can have a different spread capability around i and they can be used as a similarity measure between i and j from the perspective of node i. Note that $\sigma_{i,j} \neq \sigma_{j,i}$.

Spread capability metric has similarity with the number of common neighbors and clustering coefficient, but has additional information, see Figure 2.2. It contains the number of common neighbors (triangles) between i and j; moreover, it has the number of other triangles around i with its neighbors along the spreading pathway of gossip originated by j. We call such adjacent group of triangles as a *triangle cascade*, where all triangles are cornered at the same node (i.e., i) and are adjacent to each other through common edges.

In Figure 2.1, $j, u, v, z \in \Gamma(i)$ form a triangle cascade cornered at i. On this triangle cascade, gossip about i originated by j is spread to k, l, m, u directly. By using the cascade, gossip is propagated to v by u and then to z by v. Although $v, z \notin \Gamma(j)$, j still has a role in spreading gossip to v and z by means of triangle cascades. Hence



(a) (b)

Figure 2.2. Difference between spread factor and clustering coefficient. Networks (a) and (b) have 7 nodes and 9 edges. Degree of i is 6 and clustering coefficient of i is 0.2 in both cases. But spread factors are 0.5 in (a) and 0.33 in (b).

we have $\Gamma_j(i) = \{k, \ell, m, u, v, z\}$. Note that for all $a \in \Gamma_j(i)$, we have $\Gamma_a(i) = \Gamma_j(i)$. This property becomes very useful to reduce the computation of gossip spread factor on a cascade. Once $\sigma_{i,j}$ is calculated, then $\sigma_{i,j} = \sigma_{i,a}$ for all $a \in \Gamma_j(i)$. See further discussion in 4.4.

2.6.1. Method for Comparing Two Partitions

The success of a community detection algorithm lies in finding communities of ground-truth. A non-overlapping community detection algorithm outputs a partition such that every node belongs to exactly one community. Being in the same community is an equivalence relation. Hence, the community structure of a network is a partition of a set of nodes. Suppose we have the partition of the ground-truth. When our community detection algorithm produces its partition, we need to compare these two partitions to understand how similar they are. For comparison of two partitions, *Normalized Mutual Information* (NMI) can be used [115]. NMI is a metric to understand how far (close) two partitions are; if NMI of two partitions is close to 1.0, then they are very similar, i.e., number of communities and the members of communities in two partitions are similar; and when it is close to zero, two partitions are different from each other.

2.6.2. Method for Testing of Algorithms

A newly proposed community detection algorithm needs to be run on network datasets to understand its performance in finding correct community structures and also its execution times. The first option can be to use computer-generated networks. The LFR benchmark networks [116], with the planted community structure, can be used for testing of community detection algorithms. They are generated with a parameter vector of $[N, \langle k \rangle, k_{max}, C_{min}, C_{max}, \mu]$, where N is the number of nodes and μ is the mixing parameter controlling the rate of intra-community edges to all edges of nodes in the generated network. Community structure of an LFR network is related to the mixing parameter it is generated with. As μ increases the community structure becomes more blurry and difficult to detect. One should investigate the response of community detection algorithms to datasets generated with various mixing values. Being non-deterministic, LFR can generate different networks for the same parameter vector. In order to avoid potential bias of an algorithm to a single network, one can generate a set of LFR networks instead of a single network, i.e., 100 networks for each parameter set, and run the algorithm on all of these networks, and results of the algorithm are averaged for each parameter set.

As the second group of testing, one can use real-life networks with ground-truth community structure, i.e., DBLP dataset, Amazon co-purchase network, YouTube network and European-email network datasets provided by SNAP [117] and Zachary karate club network [56]. There are more real-life networks publicly available. The main issue with real-life networks is that provided ground-truth communities may not reflect the actual community structure of the network. It is due to the fact that, ground-truths created for these networks are generally obtained using some metadata; however, metadata may not reflect the ground-truth or can show different aspects of the network as discussed in the work of Peel *et al.* [118]. When an algorithm fails to find communities in a real-life network, this situation should be handled carefully; as it may not be the failure of the community detection algorithm.

A benchmark can also be done between the newly proposed algorithm and some of other known community detection algorithms; i.e., Newman's algorithm (NM) [98], Infomap (Inf) [34], Louvain (Lvn) [36] and Label Propagation (LPA) [35] on both generated and real-life network datasets. Partitions identified by each algorithm can be compared with the ground-truth using NMI. Execution times of all the algorithms should also be reported.

When we do not have the ground-truth community structure for real-life networks that reflects the actual communities, we can not benchmark the results of a newly proposed community detection algorithm on these networks. One can try a comparative analysis by running a set of algorithms on a network dataset and make a pairwise comparison between identified partitions of these algorithms. This is not a good way of quality testing for an algorithm because there is not a universally "best" community detection algorithm that can be used as gold standard. Such a comparative analysis can only see how close or far each algorithm to any other algorithm in terms of the number of identified communities or NMI values.

3. GOSSIP ON WEIGHTED NETWORKS

3.1. Introduction

We analyze gossip spreading on weighted networks and try to define a new metric to classify weighted complex networks using our newly proposed gossip spreading model [30]. The model proposed here is based on the gossip spreading model introduced by Lind *et al.* [28]. Our model gives more insight into the weight distribution and correlation of topology with edge weights in a network. The proposed metric is based on gossip spreading activity in the network, that is correlated with both topology and relative edge weights in the network. The metric enables us to distinguish networks from each other, i.e., social networks have different values compared to co-occurrence networks. It also measures how suitable a weighted network is for gossip spreading.

3.2. Gossip Spreading

3.2.1. Definitions

Let $G(V, E)$ be a network where V and E are the sets of *nodes* and *edges*, respectively. $N = |V|$ and $M = |E|$. The *1-neighborhood* of node i, denoted by $\Gamma(i)$, is the set of nodes directly connected to i by an edge. The *degree* of i is $k_i = |\Gamma(i)|$.

A *victim* i is a node who is the subject of a gossip and will suffer from the spread of the gossip. The node which originates the gossip is called the *originator* j, where $j \in \Gamma(i)$. A *spreader* s is a node who hears the gossip and furthers it. A *target* t is a node that is connected to both the victim i and the spreader s. Then, gossip about a victim is spread in the network from spreader to target which in turn becomes a spreader.

3.2.2. Observations

Note that victim-spreader-target form a triangle, i.e., triangle formed by nodes i, s, t. Triangles in network topology have a huge effect on how far a gossip would spread in a network. Consider the following extreme cases: If the network is a complete graph, all the nodes in $\Gamma(i)$ "know" each other and i. Hence any two nodes in 1-neighborhood of i and i itself form a triangle. Therefore all the nodes in $V \setminus \{i\}$ would get the gossip, independent of who the originator node is. On the other hand, in a star connected network, where there is no triangle, there will be no gossip spreading about any node.

Given a network, gossip propagation depends on both victim and originator. Let node v be the victim in the sample network given in Figure 3.1. When f is the originator, there won't be any gossip about v. However when g is the originator, h can get the gossip but due to its limited neighborhood, it cannot spread gossip to any other node. In the v-g-h triangle, as h has similar connectivity with g (i.e., connected to v and h only), similar gossip behavior will take place if h is the originator. Finally, let's consider one of the remaining nodes as originator: $\{a, b, c, d, e\}$. If any node in this group is selected as the originator, all the nodes in the group will get gossip eventually. Suppose b is the originator. a, c, d get it immediately since each forms a triangle with the common edge (v, b), i.e., $a, c, d \in \Gamma(v) \cap \Gamma(b)$. Although a and c cannot propagate it any further, d can. Once d gets it, it propagates to e. Here a *gossip cascade* occurs since the common edge changes from (v, b) to (v, d). As a summary, (i) a triangle is required for a single gossip, (ii) a sequence of triangles with a pair-wise common edge is necessary for gossip cascades.

3.2.3. σ-metrics

Note that given a network, the size and the duration of the gossip propagation depend on who the victim is and who originates the gossip. When the propagation ends, two metrics are investigated [27]. Let $\Gamma_j(i) \subseteq \Gamma(i)$ be the set of friends of victim i those received the gossip originated by j. *Spreading time* τ_{ij} is the largest distance between the originator and the nodes in $\Gamma_j(i)$. As already defined in Eq. 2.3, *spread-*

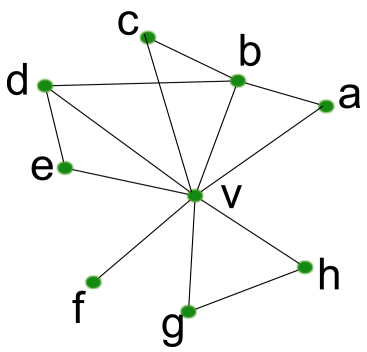


Figure 3.1. A sample network for gossip propagation. For unweighted case $\sigma_{va} = \sigma_{vb} = \sigma_{ve} = 5/8$, $\sigma_{vf} = 1/8$, $\sigma_{vg} = 2/8$ and $\sigma_v = 30/64$. For weighted case $\beta_{vf} = \beta_{vb} = 1/8$, $\beta_{va} = \beta_{vg} = 2/8$, $\beta_{vd} = \beta_{ve} = 3/8$ and $\beta_v = 16/64$ when all the weights $w_{ij} = 1$ except $w_{vb} = 2$.

factor σ_{ij} is the fraction the friends of i who received the gossip originated by j, that is

$$\sigma_{i,j} = \frac{|\Gamma_j(i)|}{|\Gamma(i)|} \tag{3.1}$$

where $\Gamma_j(i)$ is the set of neighbors of i who heard the gossip originated by j. Then, one can define the following average spread factor of a node i (as defined in Eq. 2.4)

$$\sigma_i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} \sigma_{i,j} \tag{3.2}$$

and average spread factor for the whole network as [27]

$$\sigma = \frac{1}{N} \sum_{i \in V} \sigma_i. \tag{3.3}$$

Spread factors are also averaged by the degree of victim nodes as

$$\sigma_{k_\alpha} = \frac{1}{|V_{k_\alpha}|} \sum_{i \in V_{k_\alpha}} \sigma_i \tag{3.4}$$

where V_{k_α} is the set of vertices with degree k_α. Note that $\sigma_{ij}, \sigma_i, \sigma \in [0,1]$ since $0 \leq |\Gamma_j(i)| \leq |\Gamma(i)|$. This can be used to check whether there exists a degree, i.e., k_0, in the network where gossip spread factor hits minimum.

3.3. The Model

The base model of Lind *et al.* [27] is defined on unweighted networks. Once a node gets a gossip, it must propagate it immediately. In the base model, a node does not have any decision on propagation, therefore gossip can spread as far as the network topology permits. So, the connectivity and topology determine the spreading. Lind *et al.* [28] extend their model with a probabilistic gossip spreading.

In our work, we assume that the receiver of gossip neither directly propagates it nor has a probabilistic behavior for propagation [30]. Spread or stop decision is based on how closely related the spreader and the victim is [13,16,24]. If the victim is a close friend of the spreader, he prefers not to propagate the gossip. With this motivation, we propose a gossip spreading model that extends the original model in ref [27] in two ways: (i) The network is a weighted network where edge weight w_{ij} represents the strength of the "friendship" between nodes i and j. Higher edge weight indicates a closer friendship. (ii) Based on the strength or closeness of "friendship" with the victim, spreader decides whether to stop or to propagate the gossip.

The spreading is based on triangle cascades as in [27] running on the corresponding unweighted network. The difference in our model is that spreader may choose to stop propagation if he perceives the victim as a close friend. We define the close friendship as being closer than the average $\overline{w_s}$, that is, node i is a *close friend* of node s if

$$w_{si} > \overline{w_s} = \frac{1}{|\Gamma(s)|} \sum_{\ell \in \Gamma(s)} w_{s\ell}. \tag{3.5}$$

Note that decision of node s to propagate a gossip about i depends not only on their friendship w_{si}, but also other edge weights in 1-neighborhood of s. Node s will gossip about i if s has closer friends than i, i.e., average edge weights of s with its friends in its 1-neighborhood exceeds its edge weight with i.

3.3.1. β-metrics

In order to quantify the spread of gossip, we extend σ-metrics to corresponding β-metrics as follows

$$\beta_{ij} = \frac{|m_{ij}|}{|\Gamma(i)|} \tag{3.6}$$

$$\beta_i = \frac{1}{|\Gamma(i)|} \sum_{j \in \Gamma(i)} \beta_{ij} \tag{3.7}$$

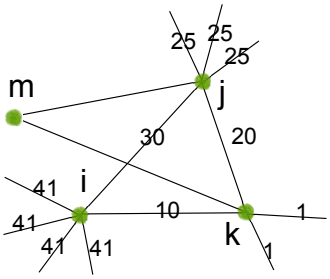
$$\beta = \frac{1}{N} \sum_{i \in V} \beta_i \tag{3.8}$$

where m_{ij} is the set of friends of i who receives the gossip originated by j on the weighted network.

Consider a weighted network and its corresponding unweighted network. Note that some nodes in $\Gamma(i)$ in the weighted network prefer not to propagate the gossip while the corresponding nodes in the unweighted network always propagate. Hence, we have $0 \leq m_{ij} \leq |\Gamma i|$ for all victim-originator pair. Therefore $\beta_{ij}, \beta_i, \beta \in [0, 1]$.

For V_{k_α} being the set of vertices with degree k_α

$$\beta_{k_\alpha} = \frac{1}{|V_{k_\alpha}|} \sum_{i \in V_{k_\alpha}} \beta_i. \tag{3.9}$$



s	t	v	$\overline{w_s}$	w_{sv}	effective (s,t)
k	i	j	8	20	0
k	j	i	8	10	0
j	i	k	25	20	1
j	k	i	25	30	0
i	j	k	34	10	1
i	k	j	34	30	1

Figure 3.2. Gossip is not symmetric. Using the table that lists effective spreading behavior, we can see that node j spreads gossip about i to k through the edge, while k does not spread gossip about i to j on the same edge.

3.3.2. Effective Network

Gossip propagation uses different paths depending on the victim-spreader pair. Note that the "close friendship" relation defined on $V \times V$ is not symmetric. Suppose node i usually makes strong ties with its friends, and node k tends to make weak ties, i.e., a high $\overline{w_i}$ and a low $\overline{w_k}$. Let $\overline{w_k} < w_{ik} < \overline{w_i}$. Then node i spreads gossip about node k while k does not about i.

The contribution of an edge to gossip propagation is not trivial. A segment of a larger network is given in Figure 3.2. Consider nodes i, j and k and their neighborhoods. The average friendship of i, j and k are $\overline{w_i} = 204/6 = 34$, $\overline{w_j} = 125/5 = 25$ and $\overline{w_k} = 32/4 = 8$, respectively. All spreader-target-victim cases are listed in Figure 3.2. Note that whether or not gossip reaches to target from the spreader, depends on the configuration of spreader-target-victim. For example let i be the victim and consider edge $e_{j,k}$. If the spreader is j, the edge is used (i.e., j tells gossip about i to k through this edge), but if the spreader is k, the edge is not used (i.e., k does not tell gossip about i to j). This behavior is given in the effective (s,t) column of table in Figure 3.2. A "0" (zero) means that the edge is not used.

Furthermore, suppose there was a node m which is connected to both j and k. Then, depending on the victim being i or m, the edge $e_{j,k}$ becomes effective or not. So for each victim-spreader pair, the underlying network behaves differently. The flow of gossip uses different paths as if there were different networks. Gossip spread has to be evaluated for each spreader-target-victim configuration separately. Note that the dynamics cannot be translated by means of preprocessing to dynamics of gossip on unweighted networks. Note also that the proposed model is actually defined on both directed and undirected weighted networks.

3.4. Data Sets

The proposed model can be applied to both directed and undirected weighted networks, but we focus on undirected weighted networks and leave the directed case for another study. Since gossip spreading is the focus of this work, we select the datasets related to human interactions. Datasets are from two basic domains, i.e., co-occurrence networks and social pattern networks. Generated network datasets and other real-life network datasets which are not based on human interactions are also used for comparison.

3.4.1. Co-occurrence Networks

Co-occurrence networks are based on bipartite graphs $G(A \cup B, E)$, where the set of nodes are in dichotomy of A and B. The nodes of the corresponding co-occurrence graph are vertices in A. Nodes $v_i, v_j \in A$ are connected whenever there is $b \in B$ such that v_i and v_j are connected to b in the bipartite graph G.

Reuters-21578 corpus is well-known in Computer Science literature [119]. It is composed of 21,578 Reuters news articles in 1987. In Reuters co-occurrence network, denoted by C-REU, nodes are the persons that appear in the news articles [120, 121]. Two persons are connected if they appear in the same article. Edge weight between i and j, i.e., w_{ij}, is defined as the number of times two persons appeared in the same article together. Note that each article contributes n to the sum of weights, i.e.,

$\frac{1}{2}\sum_{i,j} w_{ij}$, where n is the number of persons occur in the article.

In co-authorship networks, authors are represented by nodes. Two authors are connected if they have a common paper. Every paper with n author contributes $1/(n-1)$ to the weight associated with an edge between two of its authors. Note that each paper has a total contribution of $n/(n-1)$ to the sum of weights. We investigate the co-authorship networks of High-Energy Physics Theory, Condensed Matter collaborations 2005, Astrophysics [122] and co-authorship in Network Science [44], denoted by C-PHE, C-PCM, C-PA, C-NS, respectively.

Our final co-occurrence data set is from quite a different domain [123]. The network is based on Victor Hugo's novel Les Miserables. The nodes represent the key characters and two nodes are connected if they co-appear on the same stage. The weights on edges represent frequencies of their co-appearance.

3.4.2. Social Pattern Networks

The SocioPatterns project [124] collects data on socially interacting people in different settings. Nodes are the individuals and two nodes are connected by a weighted edge if they happen to be in "closed-range face-to-face proximity". Each edge has a weight that gives the duration of contact as the number of 20-seconds intervals.

The datasets that are used in this work are collected in various environments; namely, a long-running museum exhibition, a scientific conference and a school. There are many days of recordings in the museum set; we use data recorded on dates April 28, May 03, Jun 04 and July 07, 2009, denoted by S-M0428, S-M0503, S-M0604, and S-M0707, respectively [125]. The conference data is represented by S-CON [125]. For the school dataset, we use two different days, i.e., first day denoted by S-CH01 and second day by S-CH02 [126].

3.4.3. Miscellaneous Networks

In order to compare co-occurrence and social pattern networks, we use real networks originated from dynamics other than human interactions. We investigate three weighted networks from very different domains; namely, linguistics, neuroscience and air transportation. Note that they are comparable in size to the human interaction networks that are investigated.

We use the dataset known as the Edinburgh Associative Thesaurus, denoted by M-EAT, which is an example of an association network [127]. In this network, the nodes are the words. A subject is given a word and asked to provide the first word that comes to her mind. These two words are connected by an edge. In a multi-subject test, the frequency of association between two words is the weight of the edge connecting the two.

US airport network, denoted by M-USAIR, is the network of the 500 busiest commercial airports in the US. Two airports are connected if a flight was scheduled between them in 2002 [128]. Edge weights show the total number of available seats between two airports in the corresponding year.

The neural network, denoted by M-NCE, is the nervous system of C. Elegans that has 302 neurons and edges connect the neurons with each other [129, 130]. Two neurons are connected if at least one synapse or gap junction exists between them. Edge weights are set by the number of synapses and gap junctions.

3.4.4. Generated Networks

Finally, synthetic networks generated by means of well-known models of Erdős-Renyi (ER), Barabasi-Albert (BA) and Watts-Strogatz (WS) are used [130–132]. Since the models generate unweighted networks, edge weights are assigned using a distribution. The details are discussed in Section 3.5.5.

Table 3.1. Network gossip coefficients on various datasets (N: number of nodes, M: number of edges, k_0: critical degree, k_0^w : critical degree in weighted networks, CC: clustering coefficient, σ: spread factor in unweighted networks, β: spread factor in weighted networks).

Type	Data Set	N	M	k_0	k_0^w	$\frac{k_0^w}{k_0}$	CC	σ	β	$\frac{\sigma}{\text{CC}}$	$\frac{\beta}{\text{CC}}$	$\frac{\beta}{\sigma}$	$\frac{\beta}{\sigma\ \text{CC}}$	Ref
Co-occurrence	C-NS	1,589	2,742	12	34	2.83	0.64	0.68	0.35	1.06	0.55	0.51	0.80	[44]
Co-occurrence	C-PHE	8,361	15,751	15	34	2.27	0.44	0.55	0.37	1.25	0.84	0.67	1.52	[122]
Co-occurrence	C-PA	16,706	121,251	11	37	3.36	0.64	0.79	0.48	1.23	0.75	0.61	0.95	[122]
Co-occurrence	C-PCM	40,421	175,691	27	69	2.56	0.64	0.78	0.49	1.22	0.77	0.63	0.98	[122]
Co-occurrence	C-LM	77	254	4	15	3.75	0.57	0.72	0.48	1.26	0.84	0.67	1.18	[123]
Co-occurrence	C-REU	5,249	7,528	21	34	1.62	0.44	0.47	0.24	1.07	0.55	0.51	1.16	[120]
Social Pattern	S-CON	113	2,196	NA	30	NA	0.53	0.99	0.82	1.87	1.55	0.83	1.57	[125]
Social Pattern	S-M0604	133	580	7	6	0.86	0.50	0.72	0.51	1.44	1.02	0.71	1.42	[125]
Social Pattern	S-M0428	206	714	4	6	1.50	0.41	0.71	0.49	1.73	1.20	0.69	1.68	[125]
Social Pattern	S-M0503	309	1,924	3	7	2.33	0.36	0.86	0.61	2.39	1.69	0.71	1.97	[125]
Social Pattern	S-M0707	422	2,841	5	5	1.00	0.45	0.82	0.52	1.82	1.16	0.63	1.40	[125]
Social Pattern	S-CH01	236	5,899	32	36	1.13	0.50	1.00	0.77	2.00	1.54	0.77	1.54	[126]
Social Pattern	S-CH02	238	5,539	21	36	1.71	0.56	1.00	0.74	1.79	1.32	0.74	1.32	[126]
Miscellaneous	M-EAT	23,219	304,934	14	17	1.21	0.10	0.48	0.37	4.80	3.70	0.77	7.70	[127]
Miscellaneous	M-NCE	297	2,148	5	32	6.40	0.29	0.81	0.50	2.79	1.72	0.62	2.14	[130]
Miscellaneous	M-USAIR	500	2980	34	46	1.35	0.62	0.79	0.66	1.27	1.06	0.84	1.35	[128]
Generated	G-ER	1,000	10,492	30	34	1.13	0.02	0.08	0.07	4.00	3.50	0.88	44.00	[131]
Generated	G-BA	1,000	10,380	37	46	1.24	0.13	0.42	0.26	3.23	2.00	0.62	4.77	[132]
Generated	G-WS	1,000	10,000	27	24	0.89	0.54	0.85	0.50	1.57	0.93	0.59	1.09	[130]

3.5. Discussion

We use weighted networks for our model, for comparative analysis, the underlying unweighted network is analyzed by the model of Lind *et al.* [27]. There are similarities as well as differences between our weighted model and model of Lind *et al.* as summarized in Table 3.1. Data is organized in four groups in Table 3.1. The first group contains six data sets of co-occurrence networks. The second contains seven data sets of social pattern networks. The other two groups are miscellaneous real networks and generated networks.

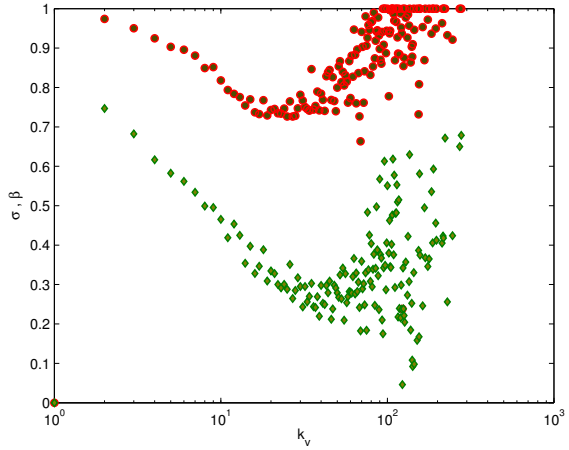
In Table 3.1, the first two columns are the data sets. N and M columns are the number of nodes and edges, respectively. CC is the clustering coefficient [130]. σ and β are defined in Eq. 3.3 and Eq. 3.8, respectively. k_0 and k_0^w are the degrees where the gossip spread becomes a minimum.

3.5.1. Degree with Minimum Gossip Spread

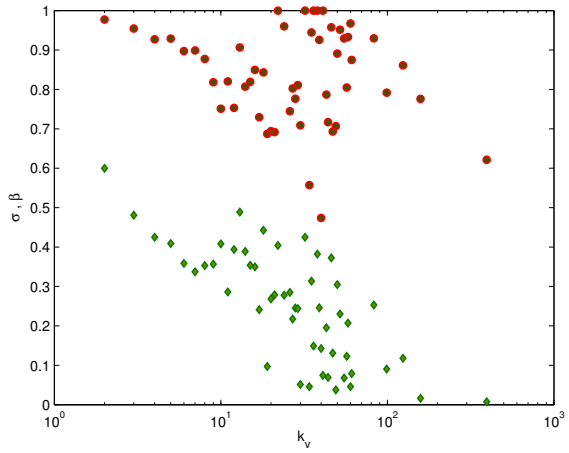
With the introduction of the decision mechanism in our model, some gossip spreading actions are blocked in the weighted network although they were possible in the corresponding unweighted network using the model of Lind *et al.* [27]. This is the result of the decision function introduced in our model where spreader evaluates that the victim is a "close friend" and stops spreading. Therefore, the spread factor of a weighted network is always smaller in value than the spread factor of the corresponding unweighted network, that is, $\beta_{k_v} \leq \sigma_{k_v}$ for all k. This can be observed in all graphs of Figure 3.3 which provides σ_{k_v} and β_{k_v} values as a function of the degree of the victim k_v.

One of the unexpected findings in [27] is the existence of a degree k_0 where gossip spreading gets to a minimum. First, we want to check this observation on the networks that we worked with. Our model is defined on weighted networks. The corresponding unweighted networks are obtained by removing the weights while keeping the connectivity. In the unweighted network, the spread factor σ_{k_v} decreases as k_v increases from 0 to some critical value k_0. As k_v further increases, σ_{k_v} starts to increase again as seen in networks C-PCM, C-REU in Figure 3.3. All the networks that we have investigated have such k_0 values. The social pattern network S-CON in Figure 3.3 is the only exception.

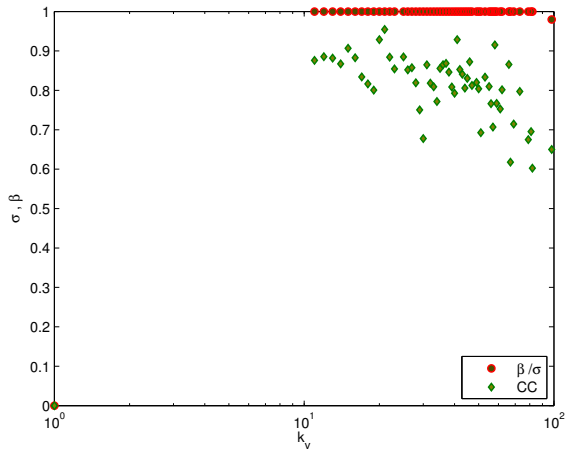
Then, we consider the weighted networks. We observe the similar pattern for the spread factor β_{k_v} in weighted networks. That is, we observe a degree k_0^w where gossip spread is a minimum. We can see a subtle k_0^w in S-CON network for the weighted case, although unweighted counterpart does not have one. k_0^w values in C-REU and S-CON networks follow similar patterns for higher degrees. This is due to the reductionist



(a) C-PCM



(b) C-REU



(c) S-CON

Figure 3.3. Average gossip spread factors versus k_v. Spread factors σ_k and β_k as a function of k_v where k_v shows the degree of set of victim nodes having k neighbors.

effect of the weight distribution of highly connected nodes in these networks (i.e., having strong connections with other nodes such that they decide not to spread a gossip about a highly connected node).

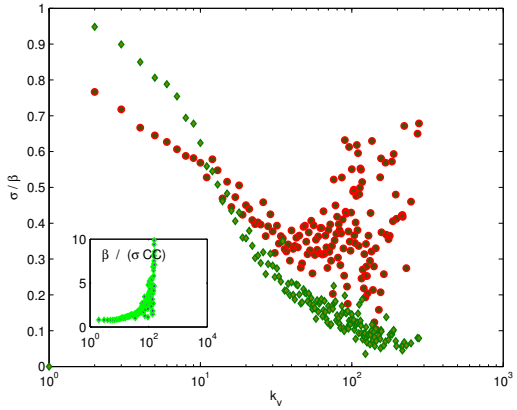
It is also observed that this degree is always higher in the weighted networks, i.e., $k_0 < k_0^w$ as seen in Table 3.1. The ration k_0^w/k_0 gives an indication of the network type. For co-occurrence networks, the ratio is high, i.e., larger than 2.0. With 1.62, the Reuters network C-REU is the only exception. On the contrary, for social pattern networks, it is low, around 1. S-M0503 network with 2.33 is an exception.

3.5.2. Discriminating Co-occurrence and Social Pattern Networks

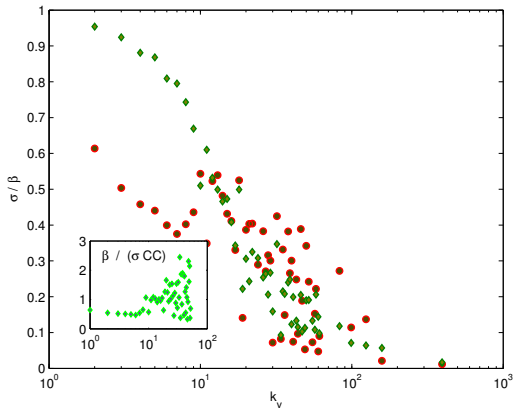
Network type plays a key role in gossip spreading. For example, co-authorship networks are formed by people coauthored in the same paper, and for this reason, authors of a paper are all connected to each other. Hence it is locally clique-like. So gossips spread mostly through on small paths, i.e., more dense connectivity inside small groups. This is a characteristic property of co-occurrence networks. This can be verified with the parallelism of the clustering coefficient and gossip spread rates in the undirected case, i.e., σ/CC.

However face-to-face proximity networks are generally formed by a person in the center and other people know each other through him. The components of these networks are like cascades, that is one person passes to another, rather than fully connected cliques where one person has access to almost all. Although gossip spread rates are very high (i.e., around 0.9), clustering coefficient values are lower than ones in co-authorship networks.

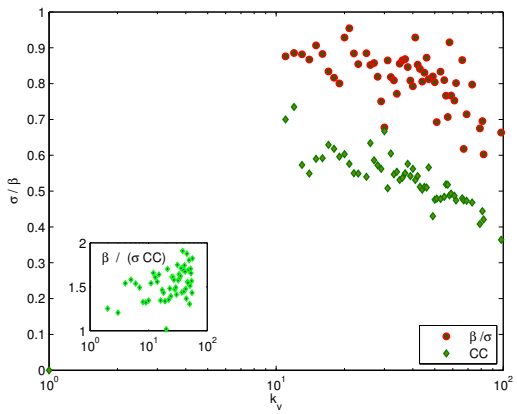
Table 3.1 has the coefficients of the overall networks. The co-occurrence group has CC in the range of $0.44-0.64$ whereas that of social patterns group is much smaller and in a range of $0.36-0.56$. The ratio σ/CC of gossip spread to the clustering coefficient is also an indicator. Networks in co-occurrence group have smaller values than networks in social patterns. More interestingly, gossip propagation in the weighted model has



(a) C-PCM



(b) C-REU



(c) S-CON

Figure 3.4. Rations of gossip metrics. $\frac{\beta_k}{\sigma_k}$ and $\frac{\beta_k}{\sigma_k \mathrm{CC}_k}$ (inset) where k is the degree of the victim.

more differential power. β/CC values of the co-occurrence group are all below 1.0. On the other hand, those of the social pattern group are all above 1.0. Just a comparison, miscellaneous networks happen to have much higher σ/CC and β/CC values than the two groups.

3.5.3. Variation by the Degree

After investigating the clustering coefficient and gossip spread factors on the average, we can analyze their correlation for each degree in the network. Figure 3.4 provides finer resolution to the degrees. The rations of β_{k_v}/σ_{k_v} and $\beta_{k_v}/(\sigma_{k_v}\mathrm{CC}_{k_v})$ are given as a function of degree k_v where $\beta_{k_v}, \sigma_{k_v}, \mathrm{CC}_{k_v}$ are the average clustering coefficient and spread factors of vertices of degree k_v in weighted and corresponding unweighted networks, respectively.

The clustering coefficient CC_{k_v} decreases as degree increases as in Figure 3.4. β/σ follows the pattern of decrease to a minimum first, then increase as in the case of σ and β in Figure 3.3.

A node with a smaller degree has relatively higher spread factor which shows that gossip about it spreads to most of its neighbors. The reason for this result is that such a node is generally a friend of a highly connected node and its small degree friends are also connected to the same highly connected node. This leads to a high clustering coefficient for the node and yields higher spreading possibility as seen in the base model [27]. This kind of connectivity is due to the power-law degree distribution frequently observed in real life networks where many nodes have fewer connections while very few nodes have many connections [132].

When we introduce the effect of edge weights, spread factor for a low degree node is close to the value in the unweighted base model. Suppose a low degree node i is connected to a high degree node j. Generally, the connection is relatively important to i. It is highly probable that this connection is one of the few connections that i has. On the other hand, i is not that important to j, because of the fact that the average

weight of j is generally larger than the weight of the connection with i. This situation leads to the concept of being popular or important in the network. The nodes with small connectivity generally have low average strength. When they are connected to a highly connected node which has greater average strength, the weight of the edge is not important to highly connected one as highly connected nodes have fewer close friends. As a result of this situation, highly connected nodes have a tendency to spread the gossips about "weak" nodes. This is an important feature of the networks that has roots in social sciences.

3.5.4. Strategies to Avoid Gossip

Some network structures have superior properties in terms of gossip avoidance. Star-like network structure, where the victim is in the center is the best topology for gossip avoidance. Because of the structure, no friend of the victim can communicate with each other without reaching the victim. For this structure, edge weights are not important: since the topology does not contain any triangles, there is no possibility of gossip.

At the other extreme, fully-connected graphs can be too gossipy. Because the topology fully enables gossip spread, the only mechanism to reduce spread is the relative edge weights in the network. A change of an edge weight in the network may affect all the other nodes in terms of gossip spreading. This is because of the fact that, decision function takes the relative importance of connections into considerations. So change of an edge value does not solely affect itself but plays an important role on other edges due to the relative evaluation of edge weights. Think about a scenario in real life; if you are the closest friend of two persons, who know each other, you will not be gossiped by them. However, if they get closer to each other and become the closest friend of each other, then you will lose your position as the closest friend, although you did nothing wrong. When this occurs, your connection becomes less important for them and they can gossip about you to each other.

According to our model, one can avoid gossip by applying some strategies:

(i) Elimination of triangles reduces the number of friends that can gossip about a victim.

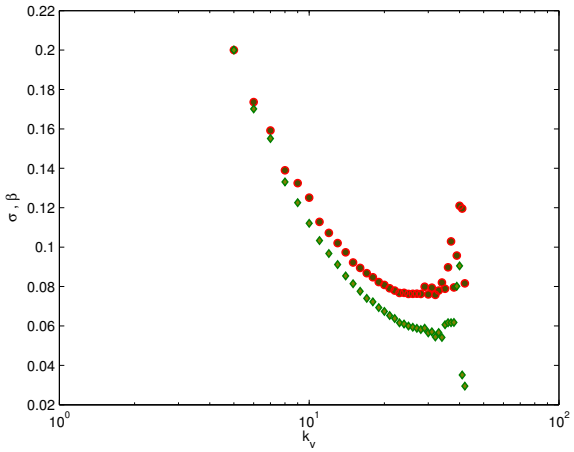
(ii) Elimination of triangle cascades can also reduce the spread. This can be achieved by having friends from different domains such that a friend from one domain does not know anybody from another domain. Therefore, gossip can spread as far as all the friends of that domain but cannot jump to another domain. As a general rule, it helps to have islands of friends such that no inter-group communication is possible. In real life, we are members of different communities such as friends from high school, from college, from work. As long as the communities do not overlap, the spread of gossip is relatively under control. The observations (i) and (ii) are valid for both unweighted and weighted networks.

(iii) In weighted networks, one can control the spread by means of the weights. Gossip does not spread if victim v has a close friendships to his friends i and j than their friendships to each other, i.e., $w_{iv} > w_{ij}$ and $w_{jv} > w_{ji}$. In a directed weighted network, this has an interesting consequence: what your friends think of you is more important than what you think of them, i.e., w_{iv} vs w_{vi}.

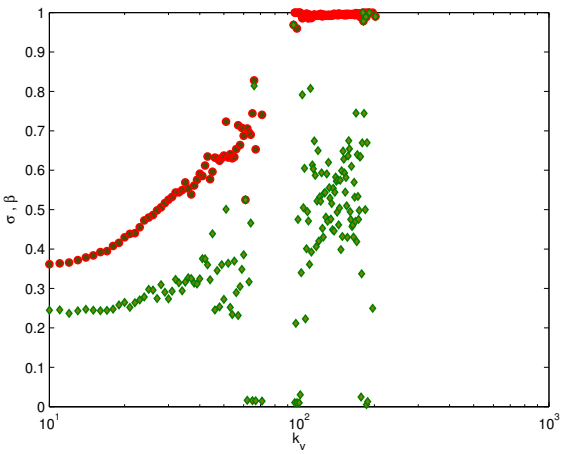
3.5.5. Generated Weighted Networks

So far, we analyze the weighted networks empirically obtained from real life. We also analyze gossip spread on the networks generated by using well-known models, namely, Erdős-Renyi (ER), Barabasi-Albert (BA), and Watts-Strogatz (WS) [130–132].

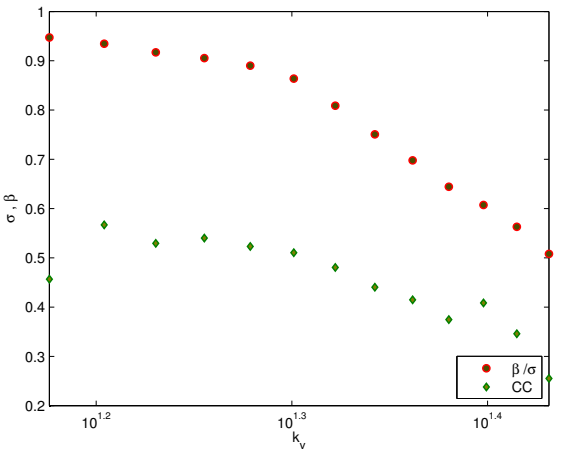
These networks are unweighted, in order obtain edge weights, we first assign node weights to all nodes (i.e., w_i for node i) using a Gaussian distribution with mean 1.0 and variance 1.0. Then we assign the edge weights w_{ij} using the node weights connected to the edges, i.e., $w_{ij} = 0.5(w_i + w_j)$.



(a) G-ER



(b) G-BA



(c) G-WS

Figure 3.5. Gossip spread factors on generated networks. σ_{k_v} and β_{k_v} values for generated networks. The values are the averages of 50 realizations.

All generated networks have $N = 1,000$ nodes and their generation parameters are set such that the number of edges is around $M = 10,000$. These values of N and M are selected arbitrarily. For each model 50 networks are generated. The results of the average of these 50 realizations are reported in Table 3.1 and Figure 3.5. The small-world networks of Watts-Strogatz (WS), denoted as G-WS, are investigated first [130]. $N = 1,000$, $M = 10,000$, $k = 20$, and rewiring probability $p = 0.1$ are used. These networks are regular networks with some re-wired links. As these networks have very few triangles, their gossip spread factor is not high. Even we increased the number of re-wiring, overall gossip spread is not large due to topology.

Power-law degree distribution networks generated by Barabasi-Albert (BA) model, are denoted by G-BA [132]. Network size is $N = 1,000$ and initial clique size is $m_0 = 40$. Each newly added node is connected to 10 existing nodes in the network (i.e., each node increases the number of edges by 10). The total number of edges is $M = 10,380$. We observe a large amount of gossip spreading on these networks. This is due to the topology of BA network, where the core of the network is a fully connected graph and newly inserted nodes have a high preferential attachment, i.e., they tend to connect to highly connected nodes. Network generation process creates highly connected hubs, each having many connections who know each other. Although weights play an important role in the decision of gossip spreading, the overall gossip spread is greater due to the topology.

Finally, random networks of Erdős-Renyi, denoted by G-ER, are investigated [131]. $N = 1,000$ with connection probability $p = 0.021$ are used. The total number of edges is $M = 10,492$. As random networks do not have scale-free property, they do not have a high clustering coefficient and preferential attachment. For this reason, the topology of random networks is not suitable for gossip spreading, i.e., there are very few triangles in these networks. This agrees with our finding that WS model with extensive rewiring decreases the gossip spread since a large number of rewiring operations on the network makes it similar to random networks. Gossip spread rates for both unweighted and weighted models are very close to each other. This is because of the lack of triangles in the network and using the random distribution for assigning edge weights.

4. COMMUNITY DETECTION USING PREFERENCE NETWORKS

4.1. Introduction

We propose a new community detection algorithm which uses a novel local approach [41]. We construct a preference network, using the same set of nodes of the original network and each node has a single (directed) outgoing edge to another node showing its preference. We try different measures to identify the preferred node of each node, but keep our focus only on the locally available information for a node, i.e., 1-neighborhood of nodes. The connected components in the preference network represent the communities of the original network. Interestingly, individual decisions of preferred nodes that are represented in the preference network collectively give the community structure of the whole network. This result is an example of emergence in complex networks.

4.2. Our Approach

Our approach is based on building a preference network where each connected component is a community [41]. Given a network, we build its corresponding preference network using the preference of each node for other nodes to be in the same community with. Every node prefers to be in the same community with certain nodes and we simply try to satisfy these requests. In this study, we implement the case where each node is allowed to select only one node, which is the most preferred node to be with. It is relatively easy to extend this approach such that nodes prefer two, three or more nodes, too. First, we describe how to satisfy such requests by means of preference network. Then we investigate ways to decide which node or nodes to be with.

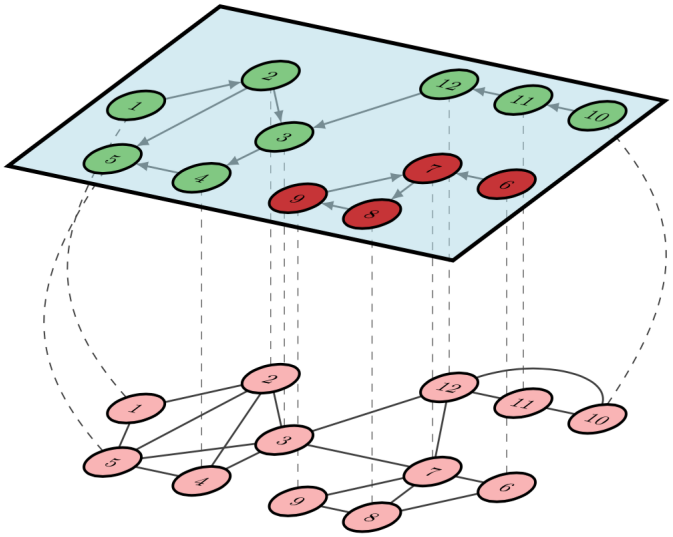


Figure 4.1. Preference network: A meta-network approach for community detection.

4.2.1. Preference Network

Let $G = (V, E)$ be an undirected network, where V and E are the sets of nodes and of edges, respectively. Define a *prefer* function $p\colon V \to V$ such that $p(i) = j$ iff node i prefers to be in the same community with node j. If we connect i to $p(i)$, clearly p induces a new directed network on V, but we will use the corresponding undirected network. Using p, we define a new undirected network $G^p = (V, E^p)$ such that nodes i and j are connected, i.e., $(i, j) \in E^p$, iff either $p(i) = j$ or $p(j) = i$. We call this network as the *preference network.* We consider the components of G^p as communities. Hence we satisfy the rule that every node is in the same community with its preferred node. Note that preference network is not a tree since it may have cycles as in the case of node a prefers node b, b prefers node c, c prefers a. A sample network and its corresponding preference network can be seen in Figure 4.1. See the algorithm of extracting communities using preference network in Section 4.4.

4.2.2. Deciding Which Node to be With

Now we can investigate a selection method of the preferred node. First of all, with the given definition, there is nothing that restricts a node to prefer any other node in the network even if the preferred node is not connected to the node. For example, a node could prefer the node with the largest betweenness centrality. This view is too general and requires global information.

We restrict the selection of the preferred node to the local neighborhood of every node. We calculate a score for each node in the local neighborhood $A(i)$ around i, with respect to i. Then select the node with the highest score (detailed later) as the preferred node. That is, we define the function p as

$$p(i) = \arg\max_{j \in A(i)} s_i(j)$$

where $s_i(j)$ being the score of node j with respect to i. In tie situations, i.e., when two or more neighbors having the highest score, the node selects one of them randomly. Note that the score $s_i(j)$ of j depends on the node i. We can interpret the score as a measure of how "important" node j is for i. If node j is the only connection of i, it has to have very big value. If i has many neighbors, then j may not be very important for i. Hence, the very same node j usually has different scores with respect to some other nodes, i.e., $s_i(j) \neq s_k(j)$.

We can define the local neighborhood in a number of ways. One can define it as the nodes whose distance is not more than ℓ to i. It can be the nodes whose distances to i are exactly ℓ. We can also include node i itself to the local neighborhood. In this case, i may prefer to be in the same community with itself. For this study, we take $A(i)$ as the 1-neighborhood of i, i.e., the set of nodes whose distance to i is exactly one, which is denoted by $\Gamma(i)$.

4.2.3. Candidates of Score Metric

There are a number of candidates for score $s_i(j)$ calculation of j with respect to i.

(i) The simplest one is to assign a random number for each neighbor of i as its score. Probably, this is not a good choice since the random function will decide independently of node j and node i; and their relations with each other.

(ii) Nodes with more connections are usually considered to be more important in a network. So, as a second choice, we can use the degree of the nodes, i.e., $s_i(j) = |\Gamma(j)|$. The degree of node j is also independent of i. So we do not incorporate what i thinks of j in the score $s_i(j)$.

(iii) A third candidate is the clustering coefficient of j, which is an indication of how densely connected its immediate neighborhood, i.e., $s_i(j) = CC_j$. This is again a value, which does not directly depend on i but may have a meaning to j and i, i.e., there is a chance that high clustering coefficient of j is at least partly because of triangles shared by j and i.

(iv) Having common neighbors is an important feature in social networks. From the definition of community, members of the community should have more edges among themselves which leads to more common neighbors of nodes inside a community. The number of common neighbors i and j is the number of triangles having i and j as two corners. For this reason, as a fourth candidate, we can use the number of common neighbors of i and j, i.e., $s_i(j) = \cap_{ij}$, defined in Eq. 2.1.

(v) As a fifth candidate, we can use the spreading capability of a neighbor j around node i defined in Eq. 2.3, i.e., $s_i(j) = \sigma_{i,j}$. It both contains the number of common neighbors and triangle cascades as discussed earlier.

(vi) And as the sixth candidate, we can use Jaccard similarity of i and j as score, i.e., $s_i(j) = J(i, j)$, defined in Eq. 2.2.

4.3. Results and Discussion

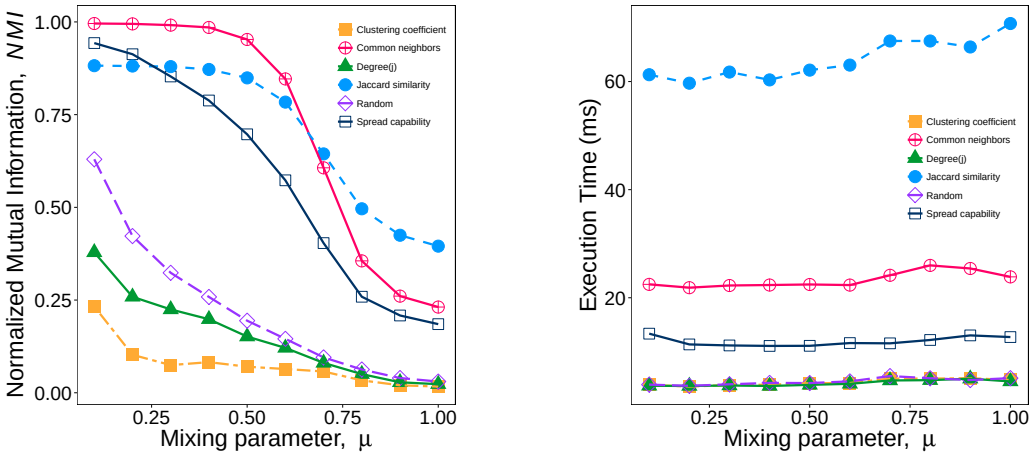
4.3.1. Test Datasets and Benchmark Algorithms

We first use the real-life networks for testing, i.e., Zachary karate club network [56], DBLP dataset, Amazon co-purchase network, YouTube network, and European email network provided by SNAP [117]. We compare the performance of our algorithm with the other algorithms, i.e., Newman's algorithm (NM) [98], Infomap (Inf) [34], Louvain (Lvn) [36] and Label Propagation (LPA) [35] on these network datasets excluding the Zachary karate club network. We also measure the execution times of all the algorithms (we use a standard laptop computer having a 2.2 GHz Intel Core i7 processor with 4-cores).

We also use the generated LFR networks for testing. In order to avoid the potential bias of an algorithm to a single network we generate 100 LFR networks for each vector and report the averages.

4.3.2. Selection of Best Score Metric

We first analyze alternative score metrics in our algorithm and try to find which one performs better in community detection. We run our algorithm on generated LFR networks [116] of $1,000$ nodes using all the score metrics, $s_i(j)$, as the method of preferred node selection. NMI values and execution times are measured. The results of our algorithm using six different score metrics on LFR networks generated with increasing mixing values (μ) are in Figure 4.2. We observe that number of common neighbors is the best score metric among six alternatives; it has the best NMI values and can identify exact community structure on networks generated with $\mu = 0.1$ and $\mu = 0.2$. Spread capability score metric has the second best results; it is better than the Jaccard similarity where Jaccard similarity metric finds 4-5 times more number of communities compared to ground-truth of LFR networks. Other three metrics; namely random score assignment, degree, and clustering coefficient can identify communities to a degree but not as successful as the ones mentioned above. In the second group of



(a) (b)

Figure 4.2. NMI comparison and execution times of different score metrics on LFR benchmark network datasets. $[N, \langle k \rangle, k_{max}, C_{min}, C_{max}] = [1000, 15, 50, 10, 50]$.

metrics, the clustering coefficient is the best one and our algorithm using the clustering coefficient score can find communities on networks generated with low μ. Interestingly, the random score metric can identify a group of communities successfully on these networks. In general, our algorithm using random score and degree-based score metrics find less number of communities compared to ground-truth.

It is trivial that calculation of simple score metrics requires less computation time compared to the calculation of other score metrics, i.e., the execution time of our algorithm using the random score, degree and clustering coefficient all have fewer execution times as given in Figure 4.2b. On the other hand, calculation of the number of common neighbors, spreading capability and Jaccard similarity require more computation time, as these metrics are calculated for each pair of nodes (i.e., the number of edges), however, these metrics have better results in terms of community detection. Hence, we select the two best performing score metrics for our algorithm, namely, common neighbors and spread capability, denoted as PCN and PSC, respectively. We use these score metrics in our algorithm for comparative analysis with other known algorithms on generated and real-life networks.

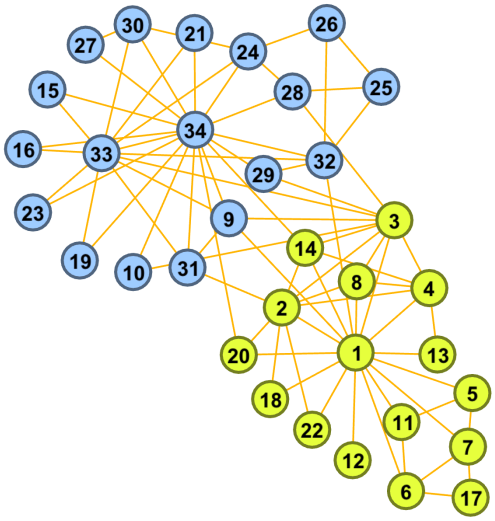


Figure 4.3. Zachary karate club: Identified communities by our PCN algorithm.

4.3.3. Result on Zachary Karate Club Network

We run our algorithm on Zachary karate club network and compare the identified communities with those of ground-truth. Our algorithm with common neighbors score metric, namely PCN, identifies two communities as seen in Figure 4.3. Only node 9 is misidentified by our algorithm. Node 9 actually has more connections in its identified community and the ground-truth metadata may not reflect the actual community. All the other nodes are identified correctly.

4.3.4. Results on Large Real-life Networks

We run our algorithm with both score metrics, PCN and PSC, on large networks with ground-truth communities provided by SNAP [117]. For comparative analysis, Infomap, Louvain, LPA and Newman's algorithm are also run on these networks. We omit Newman's algorithm on YouTube network dataset since it could not finish due to long execution time. Results are presented in Table 4.1 and Table 4.2. On all of the four real-life networks, the number of communities found by PCN, PSC, Infomap and LPA are close to each other and not far from the ground-truth (one exception is the YouTube network). On all of these networks, our algorithm finds a higher number of

communities because of its local nature.

In general, the performance of Louvain and Newman's algorithm on large real-life networks is low. Their NMI scores are very low and the number of identified communities by these algorithms are far from those of ground-truth. They find a very few number of communities compared to ground-truth. These two algorithms perform better on European-email network. Infomap and LPA algorithms generally have better NMI values compared to other algorithms, however, LPA finds only two communities in European-email network where there are 42 ground-truth communities. Our algorithm, with both score metrics (PCN and PSC), performs well on most of the networks with good NMI values. However, it performs poorly on YouTube network where all the other algorithms have similar bad results. This may be due to very small clustering coefficient of YouTube network, i.e., no trivial community structure is available.

Table 4.1. Comparison of algorithms on large real-life networks: number of communities.

Network	$\|V\|$	$\|E\|$	CC	# communities						
				GT	**PCN**	**PSC**	Inf	LPA	Lvn	NM
European-email	1,005	16,064	0.40	42	35	32	38	2	25	28
DBLP	317,080	1,049,866	0.63	13,477	28,799	28,798	30,811	36,291	565	3,165
Amazon	334,863	925,872	0.40	75,149	36,514	36,519	35,139	23,869	248	1,474
YouTube	1,134,890	2,987,624	0.08	8,385	78,021	78,053	102,125	83,256	9,616	N/A

Table 4.2. Comparison of algorithms on large real-life networks: NMI and execution times.

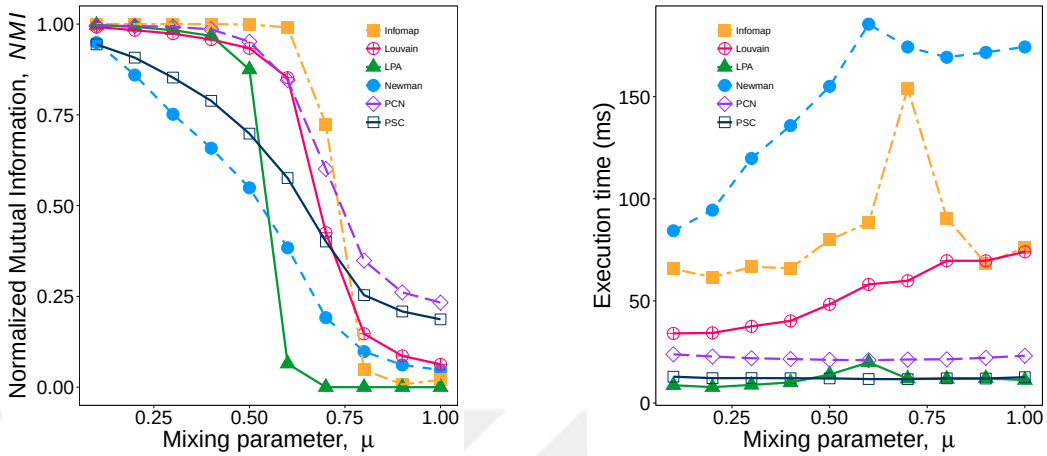
Network	NMI						execution time (ms)					
	PCN	**PSC**	Inf	LPA	Lvn	NM	**PCN**	**PSC**	Inf	LPA	Lvn	NM
European-email	0.34	0.17	0.62	0.01	0.54	0.46	192	133	133	40	69	187
DBLP	0.58	0.57	0.65	0.64	0.13	0.16	4,652	3,879	35,753	106,410	8,217	4,362,272
Amazon	0.58	0.59	0.60	0.54	0.11	0.11	2,911	3,453	43,253	83,532	8,017	1,422,590
YouTube	0.07	0.08	0.13	0.07	0.06	N/A	105,528	421,593	188,037	1,362,241	52,798	N/A

4.3.5. Results on Generated Networks

We perform the similar comparative analysis on generated LFR networks of $1,000$ and $5,000$ nodes as reported in Figure 4.4a and Figure 4.5a, respectively. We present the detailed results of algorithms on LFR networks of $5,000$ nodes in Table 4.3 and Table 4.4. As described earlier, we generate 100 LFR networks per μ value and run the algorithms on all 100 generated datasets and averaged the results for each algorithm. On LFR networks with $1,000$ nodes, our algorithm with common neighbors (PCN) is among the top three best-performing algorithms according to the NMI values; on most of the networks, Infomap and our algorithm find the best results and LPA is in the third place. Our algorithm with spread capability (PSC) has lower NMI values but still performs better than Newman's algorithm. On the networks generated with higher mixing values (i.e., $\mu > 0.5$), Infomap and LPA tend to find a small number of communities and sometimes they group all the nodes into a single community. Louvain and Newman's algorithm also find very few number of communities on these networks. However, our algorithms PSC and PCN can still find communities successfully. NMI values of our algorithm are better than those of other algorithms and the number of communities found by our method does not differ much from the ground-truth compared to other algorithms.

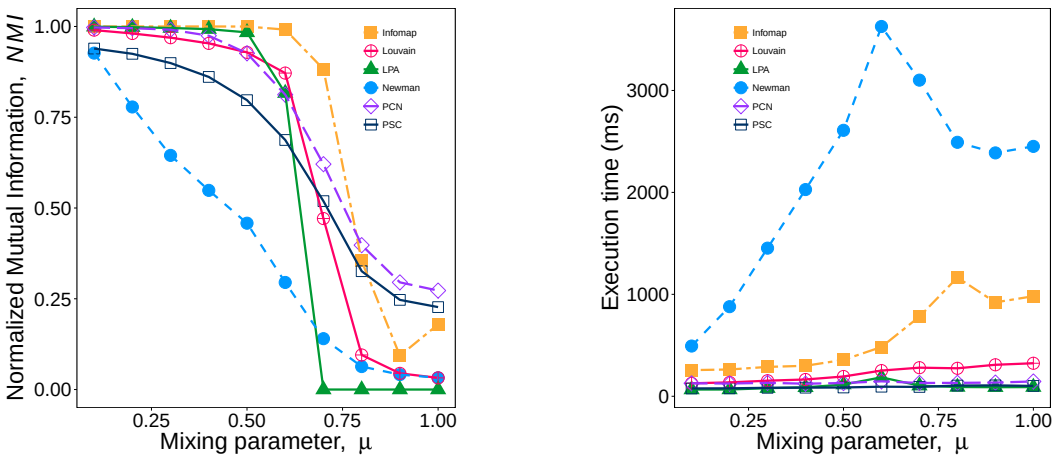
On LFR networks of $5,000$ nodes, our algorithm has better results compared to its performance on the previous set of LFR networks of $1,000$ nodes. However, with the spread capability score metric, it finds more granular communities, which leads to a greater number of communities compared to ground-truth. Newman's algorithm and Louvain algorithm find very few number of communities; they tend to merge communities which may lead to a resolution limit [47].

Infomap and LPA are both successful on large networks when mixing parameter is low, however, their quality degrades with increasing mixing parameter where our algorithm can still identify communities successfully.



(a) (b)

Figure 4.4. Comparison of our method and known algorithms on LFR benchmark network datasets (NMI and execution times).
$[N, \langle k \rangle, k_{max}, C_{min}, C_{max}] = [1000, 15, 50, 10, 50]$.



(a) (b)

Figure 4.5. Comparison of our method and known algorithms on LFR benchmark network datasets (NMI and execution times).
$[N, \langle k \rangle, k_{max}, C_{min}, C_{max}] = [5000, 15, 75, 20, 100]$.

One of the main differences between our algorithm and LPA is that we do not assign community labels to nodes but keep the information of who prefers whom to be in the same community using a preference network. During execution steps of LPA a node updates its community label according to the majority of labels of its neighbors. However, when all or part of those neighbors update their labels to be in a different community, then the node will fall apart from them, it will be in a different community (however it wanted to be in same community with them and updated its label accordingly). Using a preference network, we preserve all the preferences made by each node throughout the execution of the algorithm (because we do not update any label). And aggregation of all these preferences will eventually lead to a good community structure.

4.3.6. Performance of the Algorithm

In this section, we discuss the performance and time-complexity of our algorithm. The details are given in subsection 4.4. We use common neighbors as edge weights for PCN and spread capability for PSC. Given a network $G = (V, E)$, where maximum degree of nodes is $k_{\max}$, calculating edge weights has $O(|E|)$ and $O(|V|)$ time-complexity, for PCN and PSC, respectively.

As obtaining communities on preference network requires $O(|V|)$ time-complexity, the overall time-complexity of our algorithm is $O(|E|+|V|)$ for PCN and $O(|V|+|V|) = O(|V|)$ for PSC.

Our algorithm is fast and suitable for very large networks on a single processor environment where calculations are done in a sequential manner. Its speed can be improved further by parallel execution. As multiprocessors become available, how well an algorithm can be distributed over parallel processors becomes an important topic. Our algorithm requires calculations such as the number of common neighbors or spread capability, which can be related to edges around a node. It can be distributed to as many as N processors easily. In this case, each processor handles the local calculations around each node in the network. Then each node will have scores to decide its preferred

node in such a parallel and fast way. Network data can be redundantly replicated to all processors in order to avoid the computation overhead of data splitting process. Note that obtaining components is not easily distributed in a parallel fashion since component discovery in preference network cannot be split into independent tasks. However, this step needs fewer steps of computation and has less impact on overall performance.

Table 4.3. Comparison of algorithms on LFR benchmark networks of 5,000 nodes: number of identified communities.

Network	$\|V\|$	μ	$\|E\|$	CC	# communities						
					GT	**PCN**	**PSC**	Inf	LPA	Lvn	NM
LFR-1	5,000	0.1	38,868	0.51	101	105	240	101	100	89	64
LFR-2	5,000	0.2	38,955	0.37	101	108	254	101	100	81	31
LFR-3	5,000	0.3	38,871	0.25	101	111	266	101	98	73	18
LFR-4	5,000	0.4	38,930	0.16	101	121	283	101	96	64	12
LFR-5	5,000	0.5	38,973	0.10	100	142	294	100	91	53	9
LFR-6	5,000	0.6	38,973	0.05	100	185	300	103	74	41	11
LFR-7	5,000	0.7	38,969	0.02	101	243	280	159	1	25	14
LFR-8	5,000	0.8	38,923	0.01	100	269	243	227	1	12	13
LFR-9	5,000	0.9	38,986	0.01	102	278	238	76	1	12	13
LFR-10	5,000	1.0	38,947	0.01	101	285	242	81	1	12	13

Table 4.4. Comparison of algorithms on LFR benchmark networks of 5,000 nodes: NMI comparison and execution times.

Network	NMI						execution time (ms)					
	PCN	**PSC**	Inf	LPA	NM	Lvn	**PCN**	**PSC**	Inf	LPA	Lvn	NM
LFR-1	0.99	0.94	0.99	0.99	0.93	0.99	127	77	256	66	127	493
LFR-2	0.99	0.92	0.99	0.99	0.78	0.98	125	78	263	66	136	879
LFR-3	0.99	0.90	0.99	0.99	0.64	0.97	132	83	288	79	153	1,453
LFR-4	0.97	0.86	0.99	0.99	0.55	0.95	123	85	299	88	165	2,028
LFR-5	0.93	0.80	0.99	0.98	0.46	0.93	130	86	357	115	194	2,609
LFR-6	0.81	0.69	0.99	0.81	0.30	0.87	147	94	483	186	252	3,628
LFR-7	0.62	0.52	0.88	0.00	0.14	0.47	130	92	781	104	281	3,101
LFR-8	0.40	0.33	0.35	0.00	0.06	0.10	131	105	1,165	91	274	2,491
LFR-9	0.30	0.25	0.09	0.00	0.04	0.04	134	107	921	89	309	2,388
LFR-10	0.27	0.23	0.18	0.00	0.03	0.03	145	102	982	90	323	2,451

4.4. Details of the Algorithm and Time-complexity

The performance of the algorithm is discussed in two ways. The single processor approach deals with $O(\cdot)$ complexity. Another possibility is the scalability of the algorithm to multiprocessor running in parallel.

4.4.1. Complexity on Single Processor

In this section, we approximate the time-complexity of the algorithm for real-life networks, which are sparse networks. The algorithm is composed of three steps:

(i) For every edge (i, j) in the network, we assign a weight w_{ij}. We use two metrics for w_{ij}, namely the number of common neighbors and spread capability.

(ii) Then we construct a directed network, where each node is connected to exactly one neighbor, for which the weight is maximum.

(iii) Finally, we group the nodes into communities using the directed network. First, we investigate the complexity of calculating edge weights. Then, the complexity of obtaining communities from the directed network is investigated.

4.4.2. The Complexity of Obtaining the Number of Common Neighbors

Let $(i, j) \in E$ be the edge connecting i and j, and k_i, k_j be the degrees of i and j, respectively. In order to find out if a neighbor ℓ of i is also a neighbor of j, we need to search ℓ in the neighbors of j. Comparing ℓ with each neighbor of j would require k_j comparisons. If we keep the neighbors of j in a hash, which provides direct access, then the complexity of searching of ℓ in the hash would be $O(1)$. Since there are k_i neighbors of i, finding the common neighbors of i and j requires k_i searches in the hash. Note that if $k_i \geq k_j$, then it is better to search neighbors of j in the neighbors of i in this situation. Then the complexity finding the common neighbors of i and j is $O(\min\{k_i, k_j\})$. This is the complexity of calculating the weight of a single edge (i, j) in the network. Then the total number of comparisons required for all common

neighbors can be obtained if we consider all the edges. That is,

$$\begin{aligned}\sum_{(i,j)\in E} \min\{k_i, k_j\} &< \sum_{(i,j)\in E} \min\{k_{\max}, k_{\max}\} \\ &< \sum_{(i,j)\in E} k_{\max} \\ &< k_{\max} \sum_{(i,j)\in E} 1 \\ &< k_{\max}|E|\end{aligned}$$

where $k_{\max}$ is the maximum degree in the network. We get the worst case complexity of $O(|V|^3)$ if the network is a complete graph, where we have $k_{\max} = |V| - 1$ and $|E| = \binom{|V|}{2} \approx |V|^2$. This is not a problem for a community detection algorithm, since there is no community structure in a complete graph. Fortunately, real-life large networks are far from complete graphs. Although the number of nodes is very large, real-life networks are highly sparse, i.e., $|E| \ll |V|^2$, and their nodes are connected to a very small fraction of the nodes, i.e., $k_{\max} \ll |V|$. Note also that for networks with power-law degree distribution, $k_{\max}$ is extremely high compared to the degree of the majority of the nodes. So for real networks, we can consider $k_{\max}$ as constant, and the complexity becomes $O(|E|)$.

4.4.3. The Complexity of Obtaining Spread Capability

Suppose we want to calculate the spread capability $\sigma_{i,j}$ of gossip originator j around victim i, which is given as $\sigma_{i,j} = |\Gamma_j(i)|/|\Gamma(i)|$. The denominator $|\Gamma(i)|$ is simply the degree k_i of the victim node i. The numerator $|\Gamma_j(i)|$ needs to be calculated. We use the common neighbor algorithm to obtain spread capability as follows. In the first wave, all common neighbors of i and j will receive the gossip from j. If node ℓ is in the common neighbors of i and j, it will receive the gossip. Now ℓ starts its wave, i.e., a triangular cascade, which passes gossip to all the nodes in the common neighbors of i and ℓ. Any node that receives gossip, will start its own wave. As seen, we repeatedly use of common neighbors algorithm to propagate gossip from one node to another.

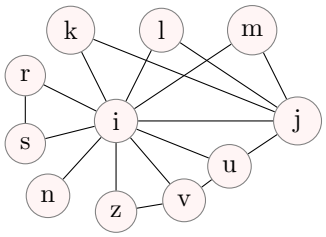


Figure 4.6. Effect of initiator selection in gossip spreading. There are three different sets of 1-neighbors of i in terms of gossip propagation, that are,
$\Gamma_k(i) = \Gamma_l(i) = \Gamma_m(i) = \Gamma_j(i) = \Gamma_u(i) = \Gamma_v(i) = \Gamma_z(i) = \{k, l, m, j, u, v, z\}$,
$\Gamma_r(i) = \Gamma_s(i) = \{r, s\}$, and $\Gamma_n(i) = \{n\}$.

The following observation of triangular cascades will enable us to do the calculation once and reuse it. Note that selection of originator makes no difference for a given cascade. As visualized in Figure 4.6, if ℓ receives gossip initiated by j, then, in return, j receives gossip initiated by ℓ. Therefore we have $\Gamma_j(i) = \Gamma_\ell(i)$, which implies $\sigma_{i,\ell} = \sigma_{i,j}$. Hence we do the calculation of $\sigma_{i,j}$ only once for the entire cascade, and use it for the remaining nodes in the cascade. This observation drastically reduces the number of calculations around i if there are triangular cascades, which is the case in networks with community structure.

Either all neighbors of i are in one cascade, or there are multiple cascades, every edge that is incident to i, has to be checked for common neighbors once, and we repeat that for k_i times. Hence the complexity is $k_i \cdot \min\{k_i, k_j\} < k_{\max}^2$. We need to do this for every node. Then the complexity becomes $O(|V|)$ if we consider $k_{\max}$ as a constant of real networks.

4.4.4. The Complexity of Obtaining Communities

Now we investigate the complexity of extraction of the communities once the preferred function p is given. We assume that nodes have aunique ID. Initially, we put all nodes in a list, mark as unvisited and label them with their unique ID, i.e., $\mathit{community}(i) = i$. We process unvisited nodes in the list one by one and terminate when all the nodes become visited as follows (See algorithm COMMUNITY-

```
COMMUNITY-EXTRACTION(V, p)
 1  // Parameters set of vertices V
 2  // and preferred function p: V → V
 3  // Uses initially empty stack
 4
 5  // Set all nodes in V as "unvisited"
 6  // and label with unique node ID
 7  while i in V
 8        visited(i) = FALSE
 9        community(i) = i
10
11  while there is i in V with visited(i) = FALSE
12        // push the nodes on the path into stack
13        while visited(i) = FALSE
14              visited(i) = TRUE
15              push(i)
16              i = p(i)
17
18        // put nodes in stack to community of i
19        C = community(i)
20        while stack not empty
21              j = pop()
22              community(j) = C
```

Figure 4.7. COMMUNITY-EXTRACTION(V, p) Algorithm.

EXTRACTION and Figure 4.8). We get an unvisited node i from the list, mark it as visited and push it into a stack. Then set the preferred node of i as i, i.e., $i \leftarrow p(i)$, and repeat the process. Eventually, i becomes a node that is already visited. Preserve the community of it in C, i.e., $C \leftarrow community(i)$. Label all the nodes in the stack with C while popping out nodes from the stack. Then repeat the process with a new unvisited node, if there is any.

Note that the algorithm passes every node twice. Once unvisited nodes are visited and push into the stack. Then once more when they are popped from the stack. So the complexity is $O(|V|)$.

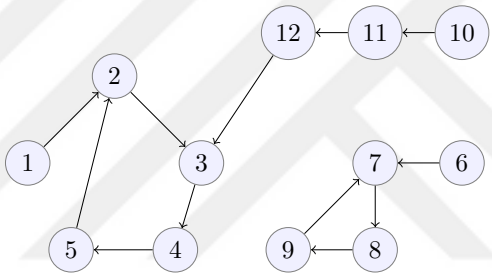


Figure 4.8. Obtaining communities from preference network. Initially, all nodes have their IDs as labels, i.e., C_1 ="1", C_2 ="2" and so on. Start with node 1 and C_1= "1", mark it as visited. Follow the arcs until arriving at an already visited node.

In Figure 4.8, there is an example of community label identification. Initially, all nodes have their IDs as labels, i.e., C_1 ="1", C_2 ="2" and so on. We start with node 1 and C_1= "1", mark it as visited. Then, we follow the arcs until arriving at an already visited node. Arc $(5, 2)$ arrives at already visited node 2, so the label of community is C_2="2", backtracking resets 5, 4, 3, 2, 1 to "2" again in this order. Then, get the next unvisited node in the list, which is node 6. Set it as visited and visit the nodes 7, 8, 9 until Arc $(9, 7)$ arrives at already visited node 7. Community labels of 9, 8, 7, 6 are set to "7". The next unvisited node is node 10. Following the arcs, the arc of $(12, 3)$ will arrive at already visited node 3. Backtrack labeling is triggered, which relabels 12, 11, and 10 as "2" and the labels of all the nodes are identified.

4.4.5. The Overall Complexity on Single Processor

Considering all the major parts of our algorithm, the overall complexity of the algorithm will include edge weight calculation and obtaining communities using the preference network. We use two methods as edge weights which require a different number of calculations, i.e., calculating the number of common neighbors used in PCN and has $O(|E|)$ time-complexity, while calculating spread capability used in PSC has $O(|V|)$ time-complexity.

As obtaining communities using preference network regardless of weight method has $O(|V|)$ time-complexity, the overall time-complexity of PCN algorithm is $O(|E| + |V|)$ and overall time-complexity of PSC algorithm is $O(|V| + |V|) = O(|V|)$.

4.4.6. The Complexity on Parallel Processors

Using local information for community detection is a good candidate for parallel execution. Let's consider the case of using common neighbors as edge weights in the network. Our algorithm has three steps of operation:

(i) Calculate the edge weights either as the number of common neighbors or as spread capability.
(ii) Connect the node to the node with the highest edge weight as the preferred node.
(iii) Identify communities using the preference network.

Step (iii) is not a good candidate for parallel execution but steps (i) and (ii) are perfect candidates since each processor can do its calculations without exchanging data with another processor.

Suppose we have P number of processors which can run in parallel. We can get P fold speed up. We dispatch entire network data to processors and each processor calculates one edge weight, then it calculates the preferred node for each node.

5. COMMUNITY DETECTION USING BOUNDARY NODES IN COMPLEX NETWORKS

5.1. Introduction

We propose a new community detection algorithm that has a local approach and tries to find communities by identifying borderlines between them using boundary nodes [42]. It is based on label propagation algorithm (LPA) [35], and it uses additional local information around each node in the network to uncover local similarities of the node with its neighbors. Using the boundary nodes approach, our algorithm also eliminates many unnecessary steps of label updates in the original label propagation algorithm, which leads to detection of more stable and accurate communities in networks. Initially, every node is considered to be a boundary node. Our algorithm naturally decreases their numbers by identifying communities of them. In the final situation, only the actual boundary nodes remain in the network and they constitute the borderlines between communities.

5.2. Background

5.2.1. Notation

Let $G = (V, E)$ be an unweighted and undirected graph, where V is the set of nodes and E is the set of edges. A *community structure* is a partition of V. We *label* each block in the partition using a symbol in the set of labels $\mathbb{L} = \{1, \ldots, |V|\}$. We define function $L\colon V \to \mathbb{L}$ which maps each node in V to a community label in $\mathbb{L}$. That is, the community of node $i \in V$ is given as $L(i)$. If two nodes i and j are in the same community, then we have $L(i) = L(j)$.

We use the concepts of Xie and Szymanski [43] to mark the nodes. A node i is called an *interior node* if it is in the same community with all of its 1-neighbors. If it is not an interior node, it is called a *boundary node*. Note that boundary nodes are positioned between nodes from different communities.

5.2.2. Network Datasets and Algorithms Used for Benchmarking

We start testing our algorithm with real-life networks with known community structure. The first network is the small network of Zachary karate club network [56]. Then we use larger networks provided by SNAP [117], namely; DBLP network, Amazon co-purchase network, YouTube network and European-email network, which all have known ground-truth communities.

On these networks, we also run some of the known community detection algorithms, namely Newman's algorithm (NM) [33], Infomap (Inf) [34], Louvain (Lvn) [36], Label Propagation Algorithm of Raghavan *et al.* (LPA) [35], and Xie and Syzmanski's algorithm (LPAc) [43] and compare their results with the ground-truth. Execution times of the algorithms are also measured and reported.

We use LFR generated networks for testing of the proposed algorithm. In order to avoid the potential bias of an algorithm to a single network, we generate 100 LFR networks for each vector and report the averages.

5.3. Our Approach

We propose a new community detection algorithm that finds communities by identifying borderlines between communities based on boundary nodes [42]. We first provide an overview of the algorithm, then we discuss the details.

The algorithm Community-By-BoundryNodes is given in Figure 5.1. The algorithm keeps track of a set S of boundary nodes. We start with $|V|$ communities of size one, i.e., each node is a community by itself. Since each node is a boundary node,

```
COMMUNITY-BY-BOUNDRYNODES
 1  // V = {1, ··· , |V|}: set of nodes
 2  // S: set of boundary nodes
 3  // L[i]: the community of i ∈ V
 4  // INITIAL-HEURISTIC(): an initial heuristic
 5  // isBOUNDRYNODE(i): TRUE if i ∈ V is a boundary node
 6  // bestCOMMUNITY(i): best community for i ∈ V
 7
 8  // initialization
 9  while i in V
10        L[i] = i
11  INITIAL-HEURISTIC()
12  while i in V
13        if isBOUNDRYNODE(i)
14              S = S ∪ {i}
15
16  // iteration
17  while S ≠ ∅
18        i = randomly selected node in S
19        S = S ∖ {i}
20        communityOld = L[i]
21        L[i] = bestCOMMUNITY(i)
22        if communityOld ≠ L[i]
23              while j in Γ(i)
24                    if isBOUNDRYNODE(j)
25                          S = S ∪ {j}
```

Figure 5.1. COMMUNITY-BY-BOUNDRYNODES Algorithm.

the set initially would have all the nodes in it.

Set S with $|V|$-elements is too large. We apply a heuristic, INITIAL-HEURISTIC in Figure 5.2, to reduce the initial number of communities, hence, the initial number of boundary nodes. For each connected pair of nodes $i, j \in V$, we calculate the "benefit score", $b_i(j)$, if i assumes the community of $j \in \Gamma(i)$. Note that $b_i(j)$ is calculated synchronously. We set the community of i to that of j with the maximum benefit score. Then, using procedure ISBOUNDRYNODE in Figure 5.3, we identify the boundary nodes and insert them into the set S.

```
INITIAL-HEURISTIC
 1  // b_i(j): benefit score if i assumes the community of j
 2
 3  while i in V
 4        maxBenefit = 0
 5        maxNode = 0
 6        while j in Γ(i)
 7              if b_i(j) > maxBenefit
 8                    maxBenefit = b_i(j)
 9                    maxNode = j
10        L[i] = L[maxNode]
```

Figure 5.2. INITIAL-HEURISTIC Procedure.

```
ISBOUNDRYNODE(i)
1  while j in Γ(i)
2        if L[j] ≠ L[i]
3              return TRUE
4  return FALSE
```

Figure 5.3. ISBOUNDRYNODE Procedure.

As long as the set is not empty, the algorithm repeats the following steps. A node i in the set is selected at random and removed from the set. We reconsider the community of the selected node based on its 1-neighborhood. A new community assignment, which produces the largest "benefit score", is made. If the old and the new communities of i are the same, i.e., no effective change, then we are done with this pass. If the community of i is changed, then this may cause some of its 1-neighbors to become boundary nodes. In this case, the new boundary nodes are inserted into the set. Note that the selected node is not added to the set during this iteration even if it is still a boundary node. It is possible that it may be inserted into the set in some other iteration, in which one of its 1-neighbors is processed. Boundary node check is done with procedure isBoundryNode.

This iteration process terminates when the set S becomes empty, which indicates that the system reaches to a steady state, where no further change in community assignment is possible with a larger "benefit score".

5.3.1. Best Community

Given a community assignment, we want to reconsider the community $L(i)$ of a node i by investigating options in its 1-neighborhood $\Gamma(i)$. This is the function of the procedure bestCommunity, which is described below. There are two different approaches:

Individual Approach: Consider each neighbor $j \in \Gamma(i)$ of i individually. Switching to the community of j produces a benefit of $b_i(j)$. Therefore, i switches to the community of j, which produces the largest benefit. That is,

$$L(i) = L\left(\arg\max_{j \in \Gamma(i)} b_i(j)\right).$$

If there is more than one community with the maximum benefit, one of them is selected randomly.

For the value of $b_i(j)$, we consider three metrics:

(i) *I-R*: Assign a uniformly random number in a range of $[0, 1]$ to $b_i(j)$. Clearly, this will not reflect any information regarding the properties of a node or its neighborhood.

(ii) *I-CC*: Use the clustering coefficient of j as the benefit score, i.e., $b_i(j) = CC_j$.

(iii) *I-CN*: Use the number of common neighbors of i and j, i.e., $b_i(j) = \cap_{ij}$.

Community Groups Approach: We consider the communities represented by the neighbors. The neighbors are grouped according to their communities. We look at the collective contribution of each group. The community of the group with the largest benefit score is selected as the new community of i. That is

$$L(i) = L\left(\arg\max_k B_i(k)\right)$$

where $B_i(k)$ is the collective benefit score of community k in 1-neighborhood of node i and defined as

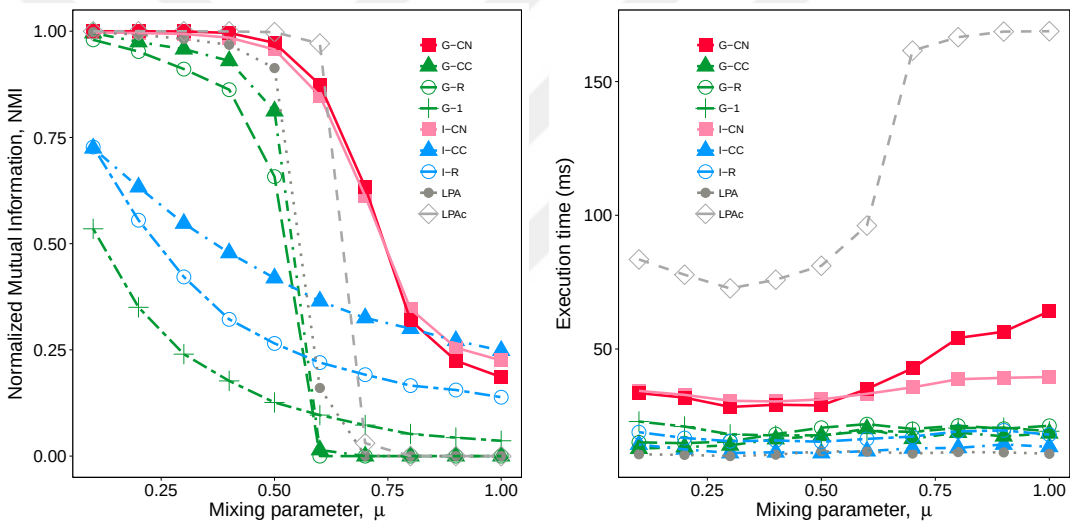
$$B_i(k) = \sum_{\substack{L(j)=k \\ j \in \Gamma(i)}} b_i(j).$$

For the value of $b_i(j)$, we consider the three metrics that we used in the individual approach. The group versions are denoted by (iv): *G-R*, (v): *G-CC*, and (vi): *G-CN*. In addition to these, we consider one more measure: (vii): *G-1*: We assign $b_i(j) = 1$ to each neighbor j. Note that this is similar to the majority rule of labels in LPA.

5.4. Experiments and Discussion

5.4.1. Deciding the Benefit Score

We define seven metrics for benefit score. In order to decide on which metric to use, we try each one on LFR generated networks of $1,000$ nodes. NMI scores of identified partitions and execution times of our algorithm are presented in Figure 5.4a and Figure 5.4b, respectively. We also run LPA and LPAc algorithms on these networks for comparison. The c parameter of LPAc is set as 0.25.



(a) NMI Scores (b) Execution Times

Figure 5.4. Comparison of various benefit score candidates. (a) NMI scores and (b) execution times of our algorithm with benefit score candidates, compared to LPA and LPAc with $c = 0.25$ on LFR benchmark network datasets. LFR parameters: $[N, \langle k \rangle, k_{max}, C_{min}, C_{max}] = [1000, 15, 50, 10, 50]$ (Average of 100 realizations).

We observe that all the group-based benefit scores have better results than the individual ones. Even group-random value assignment, G-R, has good results. Surprisingly, the uniform benefit score using the group approach, G-1, has the worst performance of all. Although it is similar to the majority rule of labels in the LPA, it is not a good fit for our algorithm. As an exception among the individual ones, I-CN outperforms LPA.

Benefit scores based on common neighbors, both at individual and group level, i.e., I-CN and G-CN, produce better results in our tests. G-CN is slightly better than I-CN in terms of NMI values. Both LPA and LPAc algorithms have good NMI values, but when $\mu > 0.6$, their performances degrade while our algorithm still finds communities. LPAc performs better than LPA. However, when we look at the execution times of algorithms, LPAc has the worst performance. Its elapsed time is two to three times higher than our G-CN algorithm. We conclude that our algorithm using the number of common neighbors with the community-groups approach, namely G-CN, produces the best results in our algorithm. We use G-CN for the rest of the paper.

5.4.2. Zachary Karate Club Network

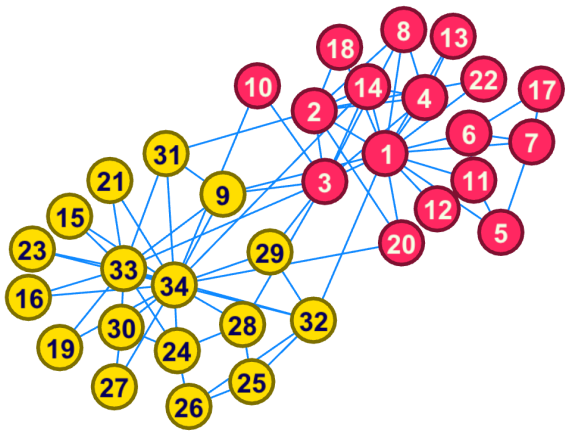


Figure 5.5. Zachary karate club: Identified communities by our G-CN algorithm.

We run our algorithm on Zachary karate club network and compare the identified communities with the ground-truth. Our algorithm G-CN identifies two communities as seen in Figure 5.5. Only the node 10 is misidentified by our algorithm. There is a tie-situation among the benefit scores exhibited to node 10 by its neighbors; so with random selection among alternatives, our algorithm sometimes selects the *wrong* community. The community labels of all the other nodes are identified correctly with respect to ground-truth community structure.

Table 5.1. Comparison of algorithms on large real-life networks with ground-truth: number of communities.

Network	$\|V\|$	$\|E\|$	CC	# communities						
				GT	**G-CN**	Inf	LPA	LPAc	Lvn	NM
European-email	1,005	16,064	0.40	42	23	38	3	20	25	28
DBLP	317,080	1,049,866	0.63	13,477	26,873	30,811	36,780	30,242	565	3,165
Amazon	334,863	925,872	0.40	75,149	33,395	35,139	24,045	30,908	248	1,474
YouTube	1,134,890	2,987,624	0.08	8,385	116,082	102,125	89,449	69,817	9,616	N/A

Table 5.2. Comparison of algorithms on large real-life networks: NMI and execution times.

Network	NMI						Execution time (ms)					
	G-CN	Inf	LPA	LPAc	Lvn	NM	**G-CN**	Inf	LPA	LPAc	Lvn	NM
European-email	0.14	0.62	0.13	0.31	0.54	0.46	146	133	32	704	69	187
DBLP	0.56	0.65	0.64	0.61	0.13	0.16	8,825	35,753	26,413	894,858	8,217	4,362,272
Amazon	0.57	0.60	0.54	0.57	0.11	0.11	7,552	43,253	30,931	997,088	8,017	1,422,590
YouTube	0.07	0.13	0.07	0.05	0.06	N/A	295,935	188,037	324,641	76,129,367	52,798	N/A

5.4.3. Large Real-life Networks

We run our G-CN algorithm on large real-life networks with ground-truth communities, provided by SNAP [117]. For comparative analysis, Newman's algorithm (NM) [98], Infomap (Inf) [34], Louvain (Lvn) [36], Label Propagation (LPA) [35] and neighborhood-strength driven LPA (LPAc) [43] are also run on these networks. Newman's algorithm is omitted for YouTube network due to its long execution time. The results are presented in Table 5.1 and Table 5.2. There is no clear winner in Table 5.2, which is a good news for local algorithms. That is, although the local algorithms cannot see the global picture, they perform good enough.

The number of detected communities by G-CN, Infomap, LPA, and LPAc are close to each other and not far from the ground-truth. There are exceptions; on the YouTube network, all four detect too many communities. On European-email network with 42 ground-truth communities, LPA merges many communities together and detects only three communities while the other three do a better job. On DBLP and Amazon networks; both Louvain and Newman's algorithm detect very few number of communities. Louvain has the best detection on YouTube network, while Newman's algorithm experiences performance problems. Our G-CN algorithm performs well on most of the networks. However, it performs poorly on the YouTube network, which has the smallest clustering coefficient of these four networks.

Considering NMI values of all six algorithms, it is possible that YouTube network may have subtle community structure. On this network, the best performing algorithm, Infomap, only gets NMI value of 0.13. On DBLP and Amazon networks; Infomap, LPA, LPAc and our G-CN algorithm obtain similar NMI values, and they are much better than Louvain and Newman's algorithm. On European-email network, local algorithms like LPA, LPAc and ours are not good enough. It is possible that the network is not a good one for local approaches. On all of the networks, LPAc has highest execution times among the local algorithms. Its execution time on YouTube network is very high compared to our algorithm and LPA.

For all the real-life networks, we use the provided ground-truth community structure to evaluate the quality of partitions found by each algorithm. However, there is no single algorithm that performs good on all networks or there is not a single network on which some algorithms perform very good. This may be due to the fact that supposed ground-truth for these networks do not reflect the original ground-truth community structure or show different aspects of the network structure, as discussed in the work of Peel *et al.* [118]. For this reason, our test on real-life networks gives an idea about the relative performance of algorithms compared to each other on different networks; but does not lead to a conclusion on whether they perform well on these networks or not.

5.4.4. Generated Networks

We test our algorithm, G-CN, also on generated LFR networks of $1,000$ and $5,000$ nodes as reported in Figure 5.6a and Figure 5.6c, respectively. The same algorithms that we run on real-life networks are also used for comparative analysis on these networks. We also measure the execution times of the algorithms and report the results in Figure 5.6b and Figure 5.6d. We present the details of the results on LFR networks of 5,000 nodes in Table 5.3 and Table 5.4. For each parameter set, we generate 100 LFR networks for a given μ and run algorithms on all these datasets and then average the results for each algorithm. On LFR networks with 1,000 nodes, our G-CN algorithm is the best algorithm with Infomap when $0.1 < \mu < 0.5$. For $0.5 < \mu < 0.8$, our algorithm is in the second place after Infomap. When $\mu > 0.7$, most of the algorithms tend to find a small number of communities while our algorithm still identifies a reasonable set of communities. LPA and LPAc find a single community that leads to the NMI value of 0. Louvain and Newman's algorithm also find very few number of communities on these networks.

The second set of test is performed on LFR networks of $5,000$ nodes. Infomap, LPA and LPAc are successful in identifying communities when mixing parameter is low, however, their quality degrades with increasing mixing parameter. LPA and LPAc have slightly better results on the networks of $5,000$ nodes generated with $0.4 < \mu <$

Table 5.3. Comparison of algorithms on LFR benchmark networks of 5,000 nodes: communities.

Network	$\|V\|$	μ	$\|E\|$	CC	# communities						
					GT	**G-CN**	Inf	LPA	LPAc	Lvn	NM
LFR-1	5,000	0.1	38,928	0.52	102	102	102	102	102	89	65
LFR-2	5,000	0.2	38,834	0.37	101	102	101	100	101	81	32
LFR-3	5,000	0.3	38,883	0.26	101	103	101	98	101	73	18
LFR-4	5,000	0.4	38,939	0.16	101	109	101	97	102	64	12
LFR-5	5,000	0.5	38,965	0.10	101	131	101	94	104	53	9
LFR-6	5,000	0.6	38,935	0.05	102	203	104	87	110	41	11
LFR-7	5,000	0.7	38,857	0.02	101	356	159	5	114	24	15
LFR-8	5,000	0.8	38,873	0.01	101	530	239	1	1	12	13
LFR-9	5,000	0.9	38,909	0.01	102	614	86	1	1	12	13
LFR-10	5,000	1.0	38,923	0.01	101	618	79	1	1	12	13

Table 5.4. Comparison of algorithms on LFR benchmark networks of 5,000 nodes: NMI and execution times.

Network	NMI wrt GT						Execution time (ms)					
	G-CN	Inf	LPA	LPAc	Lvn	NM	**G-CN**	Inf	LPA	LPAc	Lvn	NM
LFR-1	1.00	1.00	1.00	1.00	0.99	0.93	161	261	51	612	132	508
LFR-2	1.00	1.00	1.00	1.00	0.98	0.78	167	273	52	647	142	914
LFR-3	1.00	1.00	1.00	1.00	0.97	0.65	167	287	55	655	157	1,504
LFR-4	0.98	1.00	0.99	1.00	0.95	0.55	169	309	56	691	174	2,117
LFR-5	0.93	1.00	0.98	1.00	0.93	0.46	169	362	59	749	196	2,644
LFR-6	0.82	1.00	0.85	0.98	0.87	0.30	175	441	58	859	241	3,106
LFR-7	0.65	0.88	0.19	0.72	0.46	0.14	192	767	53	1,368	279	3,099
LFR-8	0.46	0.37	0.00	0.00	0.10	0.06	193	1,238	49	1,681	290	2,645
LFR-9	0.37	0.11	0.00	0.00	0.04	0.04	195	939	47	1,522	305	2,456
LFR-10	0.35	0.09	0.06	0.00	0.03	0.03	189	913	47	1,553	303	2,450

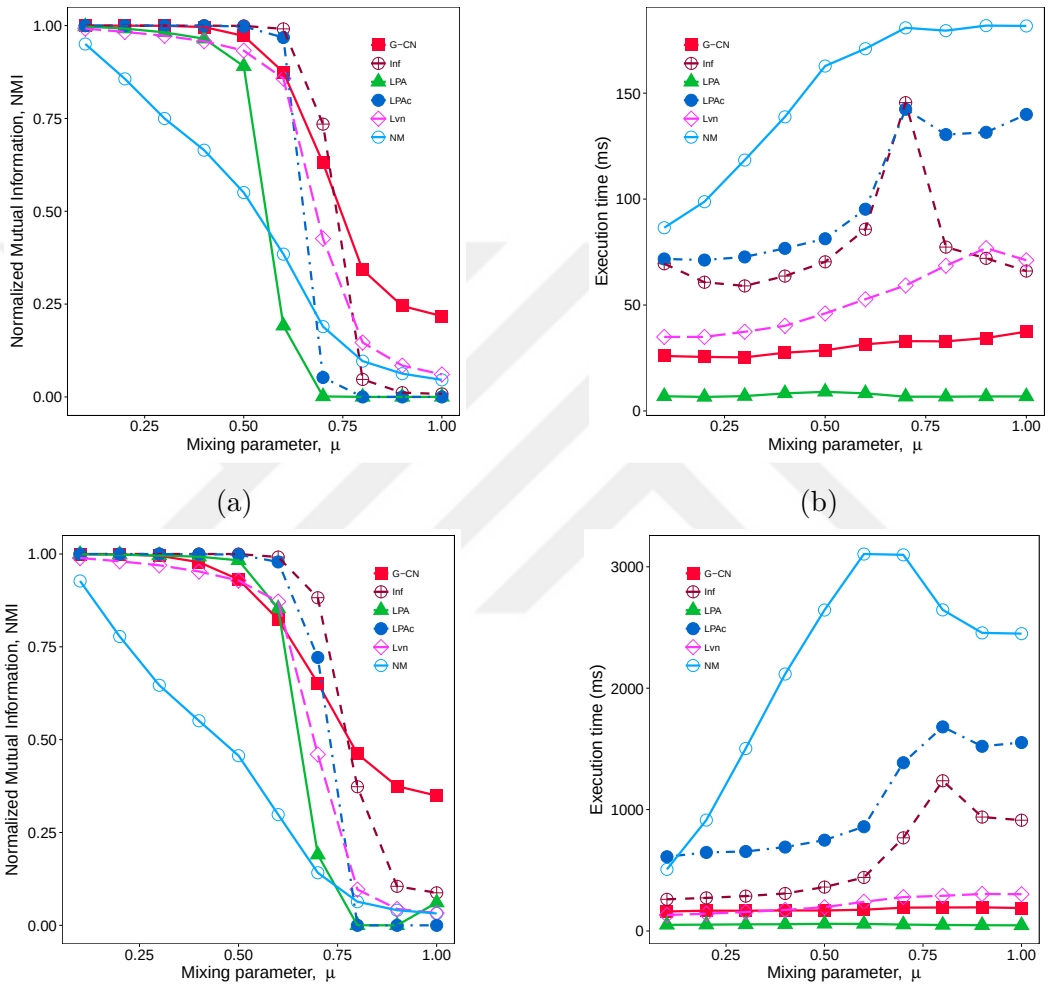


Figure 5.6. Comparison of G-CN algorithm and known algorithms on LFR networks. (a), (b) are for LFR networks generated with $[N, \langle k \rangle, k_{max}, C_{min}, C_{max}] = [1000, 15, 50, 10, 50]$. (c), (d) are generated with $[N, \langle k \rangle, k_{max}, C_{min}, C_{max}, \mu] = [5000, 15, 75, 20, 100]$ (Average of 100 realizations).

0.6 compared to the previous set of networks of $1,000$ nodes. With increasing value of the μ, performances of LPA and LPAc get worse and they tend to find a single community after $\mu > 0.7$. Infomap has the similar tendency but has better results on LFR networks of $5,000$ nodes compared to the previous set of networks of $1,000$ nodes. Newman's algorithm and Louvain find a small number of communities; they tend to merge communities, which may lead to a resolution limit [47].

Our G-CN algorithm identifies communities with high accuracy when μ is low. It is the only algorithm to identify communities when community identification becomes very hard, i.e., $\mu > 0.75$. Its execution times are lower than most of the algorithms; only LPA has better execution times. However, considering the quality of identified communities and corresponding execution times, G-CN algorithm performs better than LPA. Newman's algorithm and LPAc have the highest execution times on these networks.

6. A FRAMEWORK FOR COMMUNITY DETECTION ON COMPLEX NETWORKS

6.1. Introduction

During our research on developing a new community detection algorithm, besides the core work of building a new community detection algorithm, we have to put much effort on manual tasks during the process. Some of these tasks were finding real-life networks with community structure, using network generators, understanding and using various network dataset formats, evaluating the quality of a community detection algorithm, comparison with various existing algorithms, benchmarking the results and analyzing the networks or community partitions using different metrics. These tasks are mostly repetitive, manual and error-prone. During the development of a new community detection algorithm, a researcher may need to repeat these tasks many times when she makes a small change in her algorithm and re-evaluate the outcome of that change. We experienced the same problems, during our studies on defining a new community detection algorithm. Throughout the process, we build a community detection framework in order to speed up and automate manual tasks and define an end-to-end workflow for community detection analysis. The framework enables us to shorten the process of development and testing of a new community detection algorithm. It is built in an open and modular fashion so that one can incorporate new components (i.e., network datasets, algorithms, metrics, functions, etc.) to the framework. A paper describing the framework is at work in progress stage.

6.2. Community Detection Framework

6.2.1. Why Do We Need a Community Detection Framework?

When a new community detection algorithm is proposed, one should expect that it is doing something better than existing algorithms; i.e., running in a shorter time, more scalable, more accurate results of detected communities, etc. In order to show the proposed improvement, the algorithm should be tested on both real-life and generated networks. The first problem of the researcher is to find real-life network datasets with community structure and their ground-truth community information. If the researcher has real-life networks available with ground-truth community structure, then she should run her algorithm on these networks and compare the identified partitions with the corresponding ground-truth. If the algorithm is 100 % successful in finding the communities in a network, then identified partition should be exactly the same with that of ground-truth. When there are differences, then the success rate falls below 100 %, however, the decrease in success rate cannot be measured with manual inspection. We need a metric or method to evaluate similarity (or dissimilarity) between the identified partition and the ground-truth.

Real-life network datasets may lack ground-truth community structure as well. In some cases, meta-data that represents the communities can be misleading, so the ground-truth produced with that meta-data may not reflect the actual community structure [118]. In such cases, generated networks with planted community structure can be useful. Generated networks can be produced by generator algorithms using various input parameters. Input parameters can control many aspects of the generated network, from community strength to the average degree of nodes, etc. The researcher may need to generate many network datasets using these generators with varying input parameters and test her algorithm on all of these networks. This may lead to a big amount of work especially if it is done in a manual fashion. Also, if the researcher does not have these generators readily available, he may need to re-code or implement them.

6.2.2. Implementation Issues

After testing the algorithm on various real-life networks and generated ones, the researcher should perform a benchmarking with some of the existing algorithms. The newly proposed algorithm's results (i.e., quality of identified communities, execution times, etc.) should be compared with the results of those existing algorithms. Another major problem for the researcher is to find the correct implementation of known algorithms. Most of the time, well-known existing algorithms have available implementations by the original authors. The researcher can obtain these codes and compile them to get the executables of these algorithms. However, when implementations are missing, the researcher should implement them using the guidance of the related papers or description of those algorithms. After getting or implementing these known algorithms, the researcher can run her algorithm and these known algorithms on the same set of network datasets for benchmarking. This task is very time-consuming and error-prone even on a single network dataset. When we think of many real-life networks and especially a bunch of generated networks created with varying parameters, it is impossible to handle this benchmarking in a short time manually.

6.2.3. Format Issues

Network datasets are generally represented in various formats for the use of different algorithms. The researcher should be careful about the network dataset format and how the network is represented in that format, i.e., she needs to adapt her algorithm for each network dataset format. When ready implementations of the known algorithms are used, format problem can get complicated. First of all, many of the known algorithms accepts different formatted datasets, i.e., Pajek's .net format, GML format or .dat format. Network generators also have their own format of the generated network. For some of the different network file formats see Figure 6.1. A popular network format is the *Pajek's .net* file format, which requires to declare the number of vertices (i.e., nodes) in advance and use internal identifiers where they need to be continuous numbers starting from 0 (zero). It also has a section for edges where it requires the number of edges declared in advance. In this format, the label of vertices

can be provided in quotes and edge weights can be provided in **Edges* section. Another popular file format is GML network file format. In GML format, information about each node and edge is provided inside brackets. This format also has its internal identifiers, however they may be numbers or letters, so algorithm using this format needs its own implicit identifiers (i.e., numbers) when using this file format. Third network file format in Figure 6.1 is the *dat* file format. In this format, nodes are not explicitly provided; instead, each line has two numbers indicating an edge from the source node to target node. These numbers representing edges can be used to identify nodes and an algorithm should use its own implicit node identifiers. A major difference in this network file format is that numbers showing edges from the source node to target node do not need to start from zero and do not need to have continuous numbers, i.e., a node can be represented with 100 while another with 123.

```
*Vertices 5
0 ‘‘Adam’’
1 ‘‘Joy’’
2 ‘‘Jenny’’
3 ‘‘Alice’’
4 ‘‘Mike’’
*Edges 7
0 1
0 2
1 2
1 4
2 4
3 4
2 3
```

Pajek's .net format

```
graph
[ node
   [ id A
     label ‘‘Adam’’]
  node
   [ id 4
    label ‘‘Joy’’ ]
   ......
  edge
  [
   source A
   target 4
  ]
  ......
]
```

GML format

```
100 123
100 3
123 3
123 400
3   400
711 400
123 711
```

.dat format

Figure 6.1. A sample network represented by 3 different network file formats.

When processing the very same network, different algorithms may use different internal identifiers (ID) to represent the nodes and edges of corresponding network data. When network data is processed, i.e., for community detection etc., and results of these algorithms are externalized for comparison, then this difference in node representation may become a critical issue. We call it as *ID mismatch* problem, that is, a node is represented by different IDs by different algorithms because of their handling of the network data, i.e., node i is represented by ID(1) by $algorithm_A$ while it is represented by ID(2) by $algorithm_B$, and when their results need to be compared by externalizing data, then this single node is treated as two different nodes by two algorithms. In Figure 6.1, the same network is represented in three different file formats. In this sample network, *Mike* is represented with ID(4) in .net file format while *Joy* is represented with ID(4) in GML format and moreover an algorithm reading GML network file will have its own internal identifier which is totally different than ID(4). In .dat file format, *Joy* is represented with ID(3), which is used for *Alice* in .net format. This situation may lead to erroneous comparison results without giving a programmatic error, i.e., when a node is not represented with the same unique ID by all algorithms then the comparison is misleading. So, algorithms using different network file formats need to handled carefully to avoid this problem.

Community detection results are also represented in various formats and some of these formats are not well-documented. Again, the researcher should be careful about how communities are represented in the corresponding format. During benchmarking of different algorithms, the researcher needs to unify all the outputs from different algorithms in a common way for each network to avoid the ID mismatch, this time in community representations. There are many different output formats of community detection algorithms which may need post-processing to use for benchmarking. Some of the examples of output formats are given in Figure 6.2. In Infomap's tree file format, communities are represented with unique number identifiers and nodes belong to that community are represented with their labels in quotes. The second format for community representation, .joins file format, represent the communities as a dendrogram where each node is merged with another node or another group already merged previously. Merge order is from top of the file to the bottom and the delta increase in

network modularity is also given in the file in order to decide where to cut the dendrogram to reveal communities. The third file format, i.e., cmty file format, is relatively simple; it represents a community in a line where each line contains node identifiers.

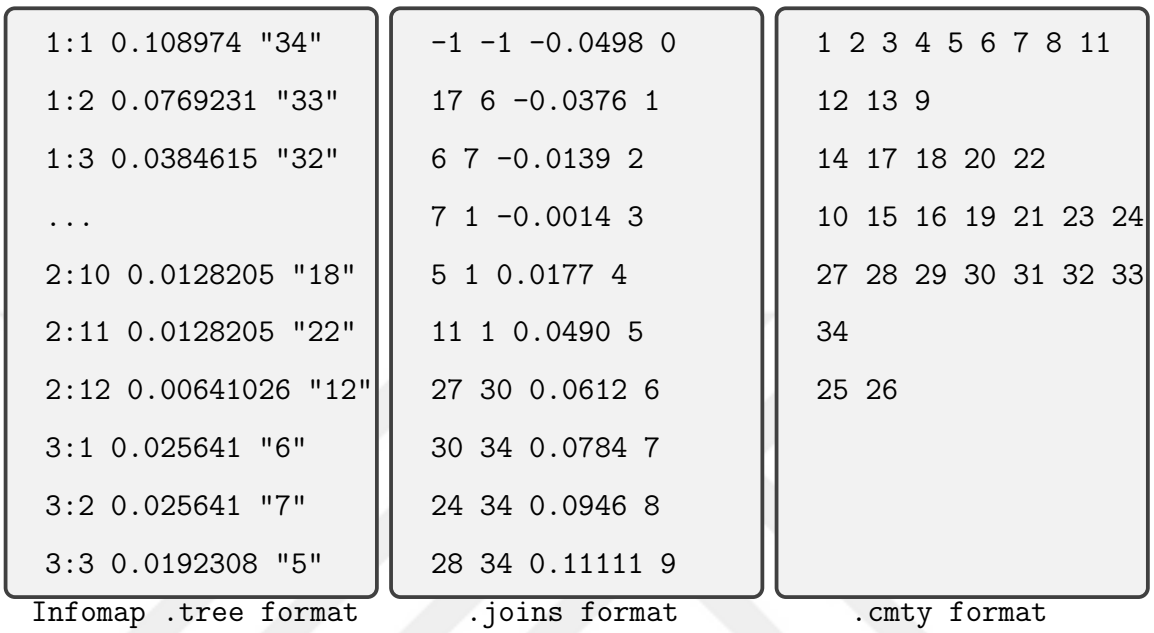


Figure 6.2. Example file formats for community detection results.

A community detection algorithm's results can also be reported using some network or community detection metrics. From the definition of community, there are some defined metrics to evaluate the quality of a partition. *Conductance*, *expansion*, *contraction* are some of the metrics that can be used to asses the quality of the identified communities by a community detection algorithm. These metrics can show whether the found partition has a good community structure; i.e., the higher the conductance, the higher the quality of the found communities. The researcher should implement these metrics to perform a self-assessment for the new algorithm.

6.2.4. Manual Tasks

Besides all the challenges mentioned so far, a general challenge is the big amount of manual tasks which are also error-prone and time-consuming. Some of these tasks are running existing algorithms on many network datasets, conversion of input files,

conversion of output files, unification of community partitions and comparison of them, reporting results, etc.

6.3. Solutions Provided in the Framework

6.3.1. Network Datasets: Real-life and Generated Networks

In our framework, we incorporate many of the real-life networks and generators for creating generated networks into the framework for the use of researchers; and if available, we include the corresponding ground-truth community structure for the real-life networks as well, i.e., co-authorship networks of condensed-matter-2005 and astrophysics collaborations [122], network of political books [133], Facebook dataset of SNAP [117], Les Miserables dataset [123], C.Elegans dataset [134], trust network of Epinions consumer review site [135] and scientific collaboration network of DBLP [136], YouTube dataset from SNAP [117] and Dogster dataset [137]. We also incorporate network generators to produce generated networks with given input parameters. Some of the network generators are Girvan-Newman [32] and LFR benchmark network generators [116]. These generators can produce many networks using a batch parameter file automatically.

There is a need for format converters both for network datasets used as inputs to community detection algorithms and community partitions produced as outputs. We incorporate many file format converters that can be used for both purposes. In order to avoid ID mismatch problem, we always use a common representation of network data and make necessary conversions for the use of algorithms in the benchmark. We ensure that the network is represented in the same way for all algorithms, regardless of their internal representations.

6.3.2. Implementations of Community Detection Algorithms

For benchmarking with existing known algorithms, we get the source code of some of existing known algorithms as implemented by the authors of those algorithms. Currently we incorporated Newman's algorithm (NM) [98], Infomap (Inf) [34], Louvain (Lvn) [36] and Label Propagation (LPA) [35]. We also incorporate our newly proposed algorithms in the framework; namely, community detection using preference networks (PCN) [41] and community detection using boundary nodes (G-CN) [42]. Executables of these algorithms are incorporated such that they can be called using system calls within the framework, without any manual work. This eliminates any possible mismatch between the original algorithm and its implementation by the researcher herself.

6.3.3. Format Converters

Input and output converters are also used here to feed all of these algorithms and convert their outputs. The output of various algorithms representing the identified communities need post-processing, so output converters are used to avoid ID mismatch and unify results for comparison with each other.

6.3.4. Comparions of the Results

We need a quantitative metric to analyze how similar identified communities in two different partitions. For this purpose, *Normalized Mutual Information* (NMI) [115] is proposed as a metric for comparison of two partitions. If NMI of two partitions is close to 1.0, then they are very similar; i.e., the number of communities and the common members of communities are more similar; and when it is close to zero, two partitions are not similar to each other. In our framework, we incorporated NMI and one can compare any two algorithms with each other using NMI. Other metrics like F-score and variations of NMI score can be added to the framework. However, there are cases where NMI has issues on measuring the similarity accurately; especially when the number of communities in different partitions varies too much. For very large networks,

time-complexity of partition comparison is high. We use efficient data structures and implement a fast version of NMI comparison that is suitable for very large networks.

6.3.5. Other Metrics

We also implemented many functions and metrics essential for network analysis and assessment of the quality of identified communities, i.e., clustering coefficient, average path length and other community detection related metrics like modularity, conductance, expansion, contraction, etc. The researcher can readily use these metrics to analyze the used networks and communities identified by her algorithm and other known algorithms.

6.3.6. Using the Framework

One of the most important aspects of our framework is that it has an automatic end-to-end structure without any manual work. It can be run by providing a batch file for necessary parameters, i.e., network input file, algorithms to use, output reporting, etc. When generated networks are needed, input parameters can be provided in the batch file and all the process are carried automatically. An example process of the framework is as follows: A batch file is prepared that contains all the details of the work that needs to be carried by the framework. The batch file contains the necessary parameters for input file names, repository, generated network parameters, algorithms to use, etc. An example batch file is in Figure 6.3, which can be used for community detection comparison of different algorithms on generated LFR networks that will be produced with this batch file.

Then framework reads the input network file and converts it to all necessary formats for the algorithms to be used, then all algorithms are run on same representation of the network and they produce partitions in their own format. When algorithms finish their run, output converters unify all these partitions and make them ready for comparison. NMI comparator compares the partitions and reports the results. If other network quality parameters are requested, they are also created using necessary

```
LFRSampleCount=10 allNodeWeightMethods=0 commentStr="BulkLFR1"
printDetails=0  allBorderNodeMethods=1 bnScoreMethods=0
borderNodeExecute="_XIE025CNPLUS1_"
GCDSampleCount=0 randomSampleCount=1 printGML=0 printCommunities=0
gossipAlgorithm="batchGossipNew" communityMethod=7
expectedOn=12 contributionType=2 model=5
networkRepo="15_LFRBenchmark"
N=1000 avgDeg=15 maxDeg=50 minc=10 maxc=50
mut=0.1 copyCSVtoFile="LFR.csv"
```

Figure 6.3. Sample batch input file for community detection framework.

functions and procedures. When generated networks are used as network inputs, all the networks are generated in the beginning with the given inputs and automatically passed to converters and algorithms. There are no manual tasks in this end-to-end workflow. All this workflow is provided in a batch file for the use of the framework. A simple layout of the framework and an example workflow is illustrated in Figure 6.4.

6.3.7. Extendable Structure

The framework is also in an open format and can be extended by the researcher. It has reusable components, data structures and easy to use building blocks to create new capabilities. The researcher can add new network datasets, new network generators, new format converters easily. She can also incorporate new community detection algorithms either by implementing them or using executables of existing algorithms easily (with necessary format converters if needed). New network metrics, as well as partition comparison techniques, can be incorporated. The source code of the framework is soon available at [138].

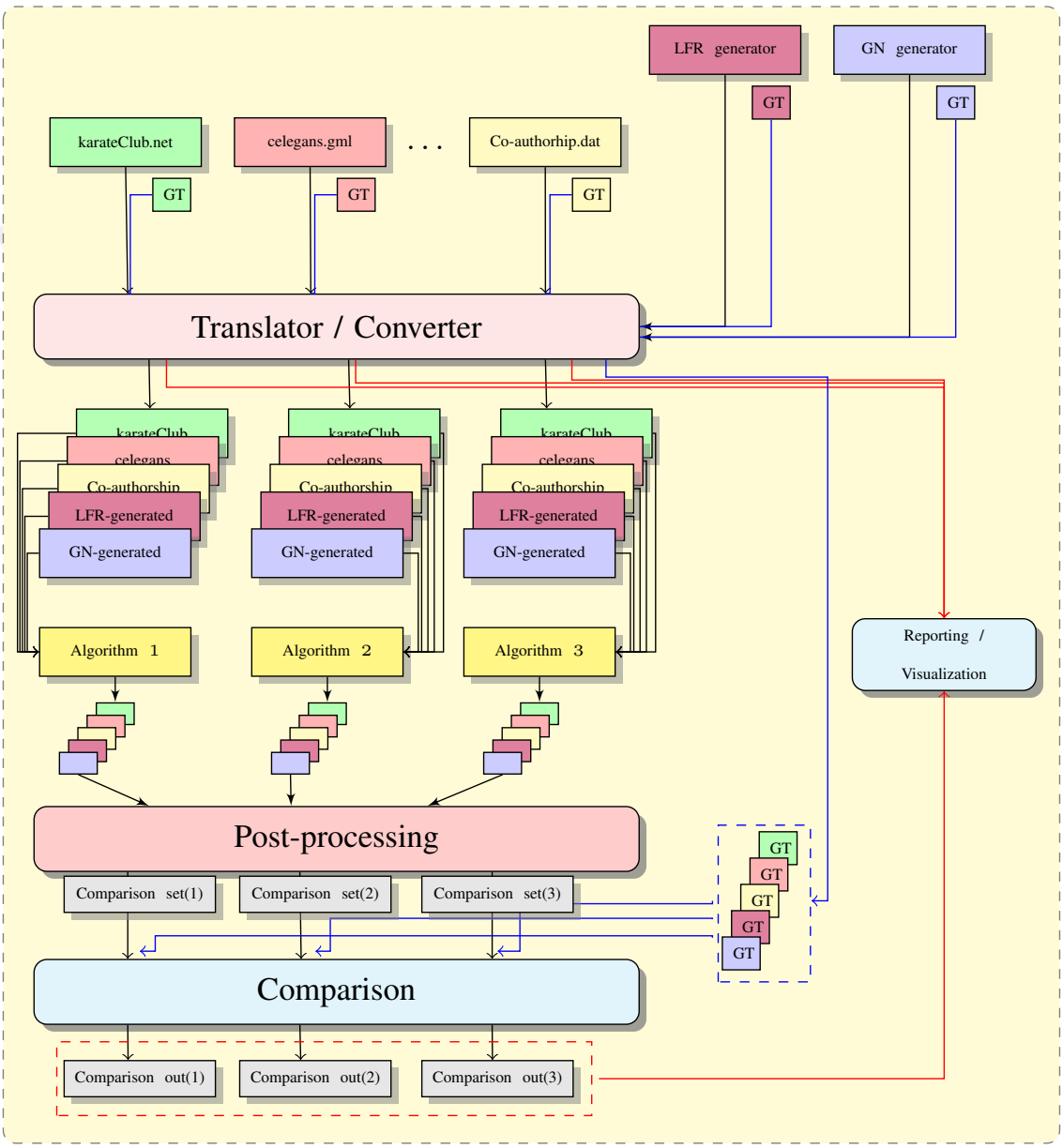


Figure 6.4. Community detection framework's layout and workflow.

7. CONCLUSION

During our studies in complex networks, we first focus on information diffusion; namely gossip spreading in complex networks. We propose a new gossip spread model for weighted networks. When we introduce the role of edge weights on the gossip spreading model, we can learn more about the connection patterns, edge weight distributions and topology of the networks. Using this, we define a new metric in our model which reveals more information about a network, i.e., edge weight distribution, topology, etc., so it can be used to classify weighted networks using gossip spreading. The metric successfully discriminates co-occurrence networks and social pattern networks. Details are given in Chapter 3.

Gossip spreading model on weighted networks uncovers some other interesting features of the networks as well. Both high-degree and low-degree victims are more vulnerable to being gossiped even when the weight promotes not spreading the gossip. The first reason is that an edge or connection is perceived differently by the nodes connected with that edge: while a node may see the connection as "important" the other node can see it as "not so important". Low degree nodes have highly connected neighbors for whom the lower degree nodes are not seen as "important". The second reason for this behavior is that there is a threshold degree after which additional connections are not close friends to the victim. Most of the time, these connections are not because of "close friendship" to the victim but as a result of attraction due to the high degree of victim node, that is, the result of preferential attachment in real networks. We can interpret this threshold degree as the optimal friendship capacity one can manage.

Information spreading and specifically gossip spreading can help us on understanding the complex networks in more detail. Gossip spreading model can be used to discriminate or classify networks with subtle differences. It is a powerful way to analyze how connection patterns lead to information diffusion and can be used to infer strategies to avoid it in a particular network.

Gossip can spread in networks with many triangles, i.e., triangles are building blocks of strong relations in many networks. Triangles can be interpreted as the smallest pattern in networks representing the common neighbors. They are building blocks of groups or communities in networks. Communities are emergent features in complex networks which is investigated by many researchers. During our research, we deeply worked on community detection algorithms and proposed two new local algorithms.

Firstly, we propose a new local community detection, *Community detection using preference networks*. The algorithm is a local algorithm which builds a meta-network representing the communities based on individual community preferences. It has two variants, PCN and PSC, where each one builds a preference network using two different node similarity score metrics; namely common neighbors and gossip spread capability.

The algorithm uses only local information to identify and represent the communities as a *preference network*. Its performance in identifying communities is good especially when community structure is not easily detectable. We used computer-generated LFR networks with planted community structure. Community structure of an LFR network is weak when it is generated with higher mixing value ($\mu > 0.7$). However our algorithm performs better than all the other benchmark algorithms used for comparison, details are in Chapter 4. On such networks, algorithms like Infomap and LPA merge all the nodes into a single community, where these algorithms are stuck in a local optimum and fail to identify communities.

We think that building a preference network to identify communities is a simple and powerful approach. With this, we preserve the preferences of all of the nodes for being in the same community with another node, whether they are highly connected or have a few connections. This approach prevents loss of granular community information, especially in very large networks.

Due to its local nature, our algorithm is scalable and fast, i.e., it needs only a single pass on the whole network to construct a preference network and similarity metric used for this construction can be evaluated in 1-neighborhood of each node. It

can run on very large networks without loss of quality and performance.

Secondly, we propose another local community detection algorithm, G-CN, which is based on identifying borderlines of communities using boundary nodes in the network. It is based on label propagation algorithm, however, it both eliminates unnecessary steps of original LPA and incorporates additional information, i.e., the number of common neighbors, for the use of nodes to choose the best community label. The proposed algorithm is able to run on very large networks with low execution times. It can identify communities with high quality, regardless of the network size.

On the networks with subtle community structure, it outperforms other algorithms, details are in Chapter 5. On these networks; Infomap, LPA, and LPAc merge all the nodes into a single community. This is due to the heuristics of these algorithms, where they lose granular structures and fail to identify communities for certain kinds of networks. However, our approach keeps granular communities by focusing on the similarity of nodes even when it has many dissimilar neighbors but only a few similar ones. It does not force small communities to join to a larger group. Our algorithm performs successfully on generated LFR networks. However, on real-life networks where ground-truth is created by using some meta-data, all of the algorithms in benchmark find different results. This may be due to the fact that meta-data does not reflect the actual ground-truth communities or metadata shows different aspects of the network structure as discussed in the work of Peel *et al.* [118].

Both of the community detection algorithms, i.e., community detection using preference network approach and boundary nodes approach, are scalable algorithms and they are suitable for distributed and parallel processing. These algorithms can be deployed as agents on different parts of a large real-life network which is evolving over time. On such a network, collecting the data of the whole network is costly (time, space, computation), while information about small parts of the network can easily be obtained and analyzed by each nearby agent for community detection. Agents can easily identify community structures of that particular area without knowing the rest of the network; which is a valuable information at that scale and can be used in real-time

by systems like peer-to-peer networks.

The source codes of the algorithms are available; community detection using preference networks is available at [139] and community detection algorithm using boundary nodes algorithm is available at [140].

Defining a new community detection algorithm requires many tasks repetitively. While working on community detection, a researcher needs to carry out manual tasks repetitively like finding and converting network datasets, comparing with other algorithms, calculating metrics to describe networks or found communities. Especially when a new algorithm is developed, every little change in the algorithm needs to be tested and analyzed over and over. During our research, we build a set of tools and a new community detection framework with modular structure and ready to use built-in features. In order to eliminate repetitive manual tasks, our framework provides a structured and automated way of handling many of these tasks. These manual tasks also may lead to human errors during these works. We also provide methods to analyze the results of a community detection algorithm; i.e., comparison of partitions with ground truth or with other algorithms' outputs, metrics showing the quality of partitions. Our framework is modular and open to new additions of network datasets, network generators, community detection algorithms and new metric calculators.

We build and use our framework during our research on community detection algorithms. With its availability, manual and repetitive tasks consumed less time and we find more time for focusing on the quality of the proposed algorithms. It also eliminates human errors while dealing with many network datasets and comparison with various different formats of benchmark algorithms. The framework has a principled workflow that enables us to try many changes in our algorithms and see the results easily. This leads to the easy development of a new community detection algorithm. Source code of the framework is soon available at [138].

REFERENCES

1. Euler, L., "Solutio Problematis ad Geometriam Situs Pertinensis", *Commentarii Academiae Scientiarum Petropolitanae*, Vol. 8, pp. 128–140, 1741.

2. Wasserman, S. and K. Faust, *Social Network Analysis: Methods and Applications*, Vol. 8, Cambridge University Press, Cambridge, 1994.

3. Onnela, J. P., J. Saramäki, J. Hyvönen, G. Szabó, D. Lazer, K. Kaski, J. Kertész and A. L. Barabási, "Structure and Tie Strengths in Mobile Communication Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 104, No. 18, pp. 7332–7336, 2007.

4. Newman, M. E. J., "The Structure of Scientific Collaboration Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 98, No. 2, pp. 404–409, 2001.

5. Leskovec, J., J. Kleinberg and C. Faloutsos, "Graphs Over Time: Densification Laws, Shrinking Diameters and Possible Explanations", *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*, pp. 177–187, ACM, New York, NY, 2005.

6. Chen, J. and B. Yuan, "Detecting Functional Modules in the Yeast Protein-Protein Interaction Network", *Bioinformatics*, Vol. 22, No. 18, pp. 2283–2290, 2006.

7. Sporns, O., "Network Attributes for Segregation and Integration in the Human Brain", *Current Opinion in Neurobiology*, Vol. 23, No. 2, pp. 162–171, 2013.

8. Newman, M. E. J. and M. Girvan, "Finding and Evaluating Community Structure in Networks", *Physical Review E*, Vol. 69, p. 026113, 2004.

9. Newman, M. E. J., "The Structure and Function of Complex Networks", *SIAM Review*, Vol. 45, No. 2, pp. 167–256, 2003.

10. Radicchi, F., C. Castellano, F. Cecconi, V. Loreto and D. Parisi, "Defining and Identifying Communities in Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 101, No. 9, pp. 2658–2663, 2004.

11. Simmel, G., *The Sociology of Georg Simmel*, Vol. 92892, Simon and Schuster, New York, NY, 1950.

12. Gilmore, D., "Varieties of Gossip in a Spanish Rural", *Ethnology*, Vol. 17, No. 1, pp. 89–99, 1978.

13. Shaw, A. K., M. Tsvetkova and R. Daneshvar, "The Effect of Gossip on Social Networks", *Complexity*, Vol. 16, No. 4, pp. 39–47, 2010.

14. Dunbar, R., "Why Gossip is Good for you", *New Scientist*, Vol. 21, pp. 28–31, 1992.

15. Dunbar, R., *Grooming, Gossip and the Evolution of Language*, Faber & Faber, London, 1996.

16. Bergmann, J. R., *Discreet Indiscretions: The Social Organization of Gossip*, Aldine de Gruyter, New York, NY, 1993.

17. Derlega, V. J. and A. L. Chaikin, "Privacy and Self-Disclosure in Social Relationships", *Journal of Social Issues*, Vol. 33, No. 3, pp. 102–115, 1977.

18. Haviland, J. B., "Gossip as Competition in Zinacantan", *Journal of Communication*, Vol. 27, No. 1, pp. 186–191, 1977.

19. Szwed, J. F., "Gossip, Drinking and Social Control: Consensus and Communication in a Newfoundland Parish", *Ethnology*, Vol. 5, No. 4, pp. 434–441, 1966.

20. Hannerz, U., "Gossip, Networks and Culture in a Black American Ghetto", *Ethnos:Journal of Anthropology*, Vol. 32, No. 1-4, pp. 35–60, 1967.

21. Feinberg, M., R. Willer, J. Stellar and D. Keltner, "The Virtues of Gossip: Reputational Information Sharing as Prosocial Behavior", *Journal of Personality and Social Psychology*, 2012.

22. Guala, F., "Reciprocity: Weak or Strong? What Punishment Experiments do (and do not) Demonstrate", *The Behavioral and Brain Sciences*, Vol. 35, No. 1, pp. 1–15, 2012.

23. Foster, E. K., "Research on Gossip: Taxonomy, Methods, and Future Directions", *Review of General Psychology*, Vol. 8, No. 2, pp. 78–99, 2004.

24. Yerkovich, S., "Gossip as a way of Speaking", *Journal of Communication*, Vol. 27, No. 1, pp. 192–196, 1977.

25. Gluckman, M., "Gossip and Scandal", *Current Anthropology*, Vol. 4, No. 3, pp. 307–316, 1963.

26. Turner, M. M., M. A. Mazur, N. Wendel and R. Winslow, "Relational Ruin or Social Glue? The Joint Effect of Relationship Type and Gossip Valence on Liking, Trust, and Expertise", *Communication Monographs*, Vol. 70, No. 2, pp. 129–141, 2003.

27. Lind, P. G., L. R. Da Silva, J. S. Andrade and H. J. Herrmann, "The Spread of Gossip in American Schools", *EPL (Europhysics Letters)*, Vol. 78, p. 68005, 2007.

28. Lind, P., L. da Silva, J. Andrade and H. Herrmann, "Spreading Gossip in Social Networks", *Physical Review E*, Vol. 76, No. 3, pp. 1–10, 2007.

29. Demers, A., D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart and D. Terry, "Epidemic Algorithms for Replicated Database

Maintenance", *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pp. 1–12, ACM, New York, NY, 1987.

30. Tasgin, M. and H. O. Bingol, "Gossip on Weighted Networks", *Advances in Complex Systems*, Vol. 15, No. supp01, p. 1250061, 2012.

31. Nekovee, M., Y. Moreno, G. Bianconi and M. Marsili, "Theory of Rumour Spreading in Complex Social Networks", *Physica A: Statistical Mechanics and its Applications*, Vol. 374, No. 1, pp. 457–470, 2007.

32. Girvan, M. and M. E. J. Newman, "Community Structure in Social and Biological Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 99, No. 12, pp. 7821–7826, 2002.

33. Newman, M. E. J., "Fast Algorithm for Detecting Community Structure in Networks", *Physical Review E*, Vol. 69, No. 6, p. 066133, 2004.

34. Rosvall, M. and C. T. Bergstrom, "An Information-theoretic Framework for Resolving Community Structure in Complex Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 104, No. 18, pp. 7327–7331, 2007.

35. Raghavan, U. N., R. Albert and S. Kumara, "Near Linear Time Algorithm to Detect Community Structures in Large-scale Networks", *Physical Review E*, Vol. 76, No. 3, p. 036106, 2007.

36. Blondel, V. D., J.-L. Guillaume, R. Lambiotte and E. Lefebvre, "Fast Unfolding of Communities in Large Networks", *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 2008, No. 10, p. P10008, 2008.

37. Gregory, S., "Finding Overlapping Communities in Networks by Label Propagation", *New Journal of Physics*, Vol. 12, No. 10, p. 103018, 2010.

38. Lancichinetti, A., F. Radicchi, J. J. Ramasco and S. Fortunato, "Finding Statistically Significant Communities in Networks", *Plos One*, Vol. 6, No. 4, p. e18961, 2011.

39. De Meo, P., E. Ferrara, G. Fiumara and A. Provetti, "Mixing Local and Global Information for Community Detection in Large Networks", *Journal of Computer and System Sciences*, Vol. 80, No. 1, pp. 72–87, 2014.

40. Eustace, J., X. Wang and Y. Cui, "Community Detection Using Local Neighborhood in Complex Networks", *Physica A: Statistical Mechanics and its Applications*, Vol. 436, pp. 665–677, 2015.

41. Tasgin, M. and H. O. Bingol, "Community Detection Using Preference Networks", *Physica A: Statistical Mechanics and its Applications*, Vol. 495, pp. 126 – 136, 2018.

42. Tasgin, M. and H. O. Bingol, "Community Detection Using Boundary Nodes in Complex Networks", *Physica A: Statistical Mechanics and its Applications*, Vol. 513, pp. 315 – 324, 2019.

43. Xie, J. and B. K. Szymanski, "Community Detection Using a Neighborhood Strength Driven Label Propagation Algorithm", *2011 IEEE Network Science Workshop (NSW)*, pp. 188–195, IEEE, IEEE Computer Society, Washington, DC, 2011.

44. Newman, M. E. J., "Finding Community Structure in Networks Using the Eigenvectors of Matrices", *Physical Review E*, Vol. 74, No. 3, p. 22, 2006.

45. Tasgin, M., A. Herdagdelen and H. Bingol, "Community Detection in Complex Networks Using Genetic Algorithms", *arXiv preprint arXiv:0711.0491*, 2007.

46. Fortunato, S., "Community Detection in Graphs", *Physics Reports*, Vol. 486, No. 3, pp. 75–174, 2010.

47. Fortunato, S. and M. Barthélemy, "Resolution Limit in Community Detection", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 104, No. 1, pp. 36–41, 2007.

48. Schaub, M. T., J.-C. Delvenne, M. Rosvall and R. Lambiotte, "The Many Facets of Community Detection in Complex Networks", *Applied Network Science*, Vol. 2, No. 1, p. 4, 2017.

49. Simmel, G., *Soziologie: Untersuchungen über die Formen der Vergesellschaftung*, Duncker und Humblot, Leipzig, 1908.

50. Burt, R. S., *Structural Holes: The Social Structure of Competition*, Harvard University Press, Cambridge, MA, 1992.

51. Csermely, P., "Creative Elements: Network-based Predictions of Active Centres in Proteins and Cellular and Social Networks", *Trends in Biochemical Sciences*, Vol. 33, No. 12, pp. 569–576, 2008.

52. Rice, S. A., "The Identification of Blocs in Small Political Bodies", *American Political Science Review*, Vol. 21, No. 3, pp. 619–627, 1927.

53. Davis, A., B. B. Gardner, M. R. Gardner and W. L. Warner, *Deep South: A Social Anthropological of Caste and Class*, University of Chicago Press, Chicago,Ill, 1941.

54. Homans, G. C., *The Human Group*, Harcourt, Brace, Oxford, 1950.

55. Weiss, R. S. and E. Jacobson, "A Method for the Analysis of the Structure of Complex Organizations", *American Sociological Review*, Vol. 20, No. 6, pp. 661–668, 1955.

56. Zachary, W. W., "An Information Flow Model for Conflict and Fission in Small Groups", *Journal of Anthropological Research*, Vol. 33, No. 4, pp. 452–473, 1977.

57. Bech, M. L. and E. Atalay, "The Topology of the Federal Funds Market", *Physica*

A: Statistical Mechanics and its Applications, Vol. 389, No. 22, pp. 5223 – 5246, 2010.

58. Lusseau, D., K. Schneider, O. J. Boisseau, P. Haase, E. Slooten and S. M. Dawson, "The Bottlenose Dolphin Community of Doubtful Sound Features a Large Proportion of Long-lasting Associations", *Behavioral Ecology and Sociobiology*, Vol. 54, No. 4, pp. 396–405, 2003.

59. Liu, Z., J.-L. He, K. Kapoor and J. Srivastava, "Correlations Between Community Structure and Link Formation in Complex Networks", *Plos One*, Vol. 8, No. 9, p. e72908, 2013.

60. Zhang, P., F. Wang, X. Wang, A. Zeng and J. Xiao, "The Reconstruction of Complex Networks with Community Structure", *Scientific Reports*, Vol. 5, p. 17287, 2015.

61. Xin, L., E. Hai-Hong, J.-j. TONG and M.-n. SONG, "Collaborative Recommendation Based on Social Community Detection", *The Journal of China Universities of Posts and Telecommunications*, Vol. 21, pp. 20–45, 2014.

62. Abdrabbah, S. B., R. Ayachi and N. B. Amor, "Collaborative Filtering Based on Dynamic Community Detection", *Dynamic Networks and Knowledge Discovery*, Vol. 85, 2014.

63. Sahebi, S. and W. W. Cohen, "Community-based Recommendations: A Solution to the Cold Start Problem", *Proceedings of WOODSTOCK'97*, 1997.

64. Boratto, L., S. Carta, A. Chessa, M. Agelli and M. L. Clemente, "Group Recommendation with Automatic Identification of Users Communities", *Proceedings of IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies (WI-IAT '09)*, Vol. 3, pp. 547–550, IEEE, Washington, DC, 2009.

65. Qiang, H. and G. Yan, "A Method of Personalized Recommendation Based on Multi-label Propagation for Overlapping Community Detection", *3rd International Conference on System Science, Engineering Design and Manufacturing Informatization (ICSEM)*, Vol. 1, pp. 360–364, IEEE, New York, NY, 2012.

66. Zhao, G., M. L. Lee, W. Hsu, W. Chen and H. Hu, "Community-based User Recommendation in Uni-directional Social Networks", *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pp. 189–198, ACM, New York, NY, 2013.

67. Fatemi, M. and L. Tokarchuk, "A Community Based Social Recommender System for Individuals & Groups", *International Conference on Social Computing (SocialCom)*, pp. 351–356, IEEE Computer Society, Washington, DC, 2013.

68. Udrescu, L., L. Sbârcea, A. Topîrceanu, A. Iovanovici, L. Kurunczi, P. Bogdan and M. Udrescu, "Clustering Drug-drug Interaction Networks with Energy Model Layouts: Community Analysis and Drug Repurposing", *Scientific Reports*, Vol. 6, p. 32745, 2016.

69. Zhang, P., F. Wang, J. Hu and R. Sorrentino, "Label Propagation Prediction of Drug-drug Interactions Based on Clinical Side Effects", *Scientific Reports*, Vol. 5, p. 12339, 2015.

70. Nacher, J. C. and J.-M. Schwartz, "Modularity in Protein Complex and Drug Interactions Reveals New Polypharmacological Properties", *Plos One*, Vol. 7, No. 1, p. e30028, 2012.

71. Vanunu, O., O. Magger, E. Ruppin, T. Shlomi and R. Sharan, "Associating Genes and Protein Complexes with Disease via Network Propagation", *Plos Computational Biology*, Vol. 6, No. 1, p. e1000641, 2010.

72. Shen, X., L. Yi, X. Jiang, Y. Zhao, X. Hu, T. He and J. Yang, "Neighbor Affinity Based Algorithm for Discovering Temporal Protein Complex from Dynamic PPI

Network", *Methods*, Vol. 110, pp. 90 – 96, 2016.

73. Cantini, L., E. Medico, S. Fortunato and M. Caselle, "Detection of Gene Communities in Multi-networks Reveals Cancer Drivers", *Scientific Reports*, Vol. 5, p. 17386, 2015.

74. Bargigli, L. and M. Gallegati, "Finding Communities in Credit Networks", *Economics: The Open-Access, Open-Assessment E-Journal*, Vol. 7, No. 2013-17, pp. 1–39, 2013.

75. Guerra, S. M., T. C. Silva, B. M. Tabak, R. A. de Souza Penaloza and R. C. de Castro Miranda, "Systemic Risk Measures", *Physica A: Statistical Mechanics and its Applications*, Vol. 442, pp. 329 – 342, 2016.

76. Liu, J., E. Bier, A. Wilson, J. A. Guerra-Gomez, T. Honda, K. Sricharan, L. Gilpin and D. Davies, "Graph Analysis for Detecting Fraud, Waste, and Abuse in Healthcare Data", *AI Magazine*, Vol. 37, No. 2, pp. 33–46, 2016.

77. Akoglu, L., H. Tong and D. Koutra, "Graph Based Anomaly Detection and Description: A Survey", *Data Mining and Knowledge Discovery*, Vol. 29, No. 3, pp. 626–688, 2015.

78. Palla, G., A.-L. Barabási and T. Vicsek, "Quantifying Social Group Evolution", *Nature*, Vol. 446, No. 7136, p. 664, 2007.

79. Xu, K. S. and A. O. Hero, "Dynamic Stochastic Blockmodels for Time-evolving Social Networks", *IEEE Journal of Selected Topics in Signal Processing*, Vol. 8, No. 4, pp. 552–562, 2014.

80. Lin, Y.-R., Y. Chi, S. Zhu, H. Sundaram and B. L. Tseng, "Analyzing Communities and Their Evolutions in Dynamic Social Networks", *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Vol. 3, No. 2, p. 8, 2009.

81. Yang, T., Y. Chi, S. Zhu, Y. Gong and R. Jin, "A Bayesian Approach Toward Finding Communities and Their Evolutions in Dynamic Social Networks", *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp. 990–1001, SIAM, Philadelphia, PA, 2009.

82. Ning, H., W. Xu, Y. Chi, Y. Gong and T. Huang, "Incremental Spectral Clustering with Application to Monitoring of Evolving Blog Communities", *Proceedings of the 2007 SIAM International Conference on Data Mining*, pp. 261–272, SIAM, Philadelphia, PA, 2007.

83. Nguyen, N. P., T. N. Dinh, Y. Shen and M. T. Thai, "Dynamic Social Community Detection and Its Applications", *Plos One*, Vol. 9, No. 4, p. e91431, 2014.

84. Cazabet, R., F. Amblard and C. Hanachi, "Detection of Overlapping Communities in Dynamical Social Networks", *2010 IEEE Second International Conference on Social Computing (SocialCom)*, pp. 309–314, IEEE Computer Society, Washington, DC, 2010.

85. Han, J., W. Li, L. Zhao, Z. Su, Y. Zou and W. Deng, "Community Detection in Dynamic Networks via Adaptive Label Propagation", *Plos One*, Vol. 12, No. 11, p. e0188655, 2017.

86. Evans, T. S. and R. Lambiotte, "Line Graphs, Link Partitions, and Overlapping Communities", *Physical Review E*, Vol. 80, p. 016105, 2009.

87. Psorakis, I., S. Roberts, M. Ebden and B. Sheldon, "Overlapping Community Detection Using Bayesian Non-negative Matrix Factorization", *Physical Review E*, Vol. 83, p. 066114, 2011.

88. Lancichinetti, A., S. Fortunato and J. Kertész, "Detecting the Overlapping and Hierarchical Community Structure in Complex Networks", *New Journal of Physics*, Vol. 11, No. 3, p. 033015, 2009.

89. Chen, D., M. Shang, Z. Lv and Y. Fu, "Detecting Overlapping Communities of Weighted Networks via a Local Algorithm", *Physica A: Statistical Mechanics and its Applications*, Vol. 389, No. 19, pp. 4177 – 4187, 2010.

90. Gan, G., C. Ma and J. Wu, *Data Clustering: Theory, Algorithms, and Applications*, Vol. 20, SIAM, Philadelphia, PA, 2007.

91. Kernighan, B. W. and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *The Bell System Technical Journal*, Vol. 49, No. 2, pp. 291–307, 1970.

92. Barnes, E. R., "An Algorithm for Partitioning the Nodes of a Graph", *SIAM Journal on Algebraic Discrete Methods*, Vol. 3, No. 4, pp. 541–550, 1982.

93. Ford, L. R. and D. R. Fulkerson, "Maximal Flow Through a Network", *Canadian Journal of Mathematics*, Vol. 8, No. 3, pp. 399–404, 1956.

94. Goldberg, A. V. and R. E. Tarjan, "A new Approach to the Maximum-flow Problem", *Journal of the ACM (JACM)*, Vol. 35, No. 4, pp. 921–940, 1988.

95. Flake, G. W., S. Lawrence, C. L. Giles and F. M. Coetzee, "Self-organization and Identification of Web Communities", *Computer*, Vol. 35, No. 3, pp. 66–70, 2002.

96. Rattigan, M. J., M. Maier and D. Jensen, "Graph Clustering with Network Structure Indices", *Proceedings of the 24th International Conference on Machine Learning*, pp. 783–790, ACM, New York, NY, 2007.

97. Holme, P., M. Huss and H. Jeong, "Subnetwork Hierarchies of Biochemical Pathways", *Bioinformatics*, Vol. 19, No. 4, pp. 532–538, 2003.

98. Clauset, A., M. E. J. Newman and C. Moore, "Finding Community Structure in Very Large Networks", *Physical Review E*, Vol. 70, p. 066111, 2004.

99. Danon, L., A. Díaz-Guilera and A. Arenas, "The Effect of Size Heterogeneity on Community Identification in Complex Networks", *Journal of Statistical Mechan-*

ics: Theory and Experiment, Vol. 2006, No. 11, p. P11010, 2006.

100. Wakita, K. and T. Tsurumi, "Finding Community Structure in Mega-scale Social Networks", *Proceedings of the 16th International Conference on World Wide Web*, pp. 1275–1276, ACM, New York, NY, 2007.

101. Pujol, J. M., J. Béjar and J. Delgado, "Clustering Algorithm for Determining Community Structure in Large Networks", *Physical Review E*, Vol. 74, No. 1, p. 016107, 2006.

102. Du, H., M. W. Feldman, S. Li and X. Jin, "An Algorithm for Detecting Community Structure of Social Networks Based on Prior Knowledge and Modularity", *Complexity*, Vol. 12, No. 3, pp. 53–60, 2007.

103. MacQueen, J., "Some Methods for Classification and Analysis of Multivariate Observations", *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, pp. 281–297, University of California Press, Berkeley, CA, 1967.

104. Hagen, L. and A. B. Kahng, "New Spectral Methods for Ratio Cut Partitioning and Clustering", *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, Vol. 11, No. 9, pp. 1074–1085, 1992.

105. Shi, J. and J. Malik, "Normalized Cuts and Image Segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 22, No. 8, pp. 888–905, 2000.

106. Holland, P. W., K. B. Laskey and S. Leinhardt, "Stochastic Blockmodels: First Steps", *Social Networks*, Vol. 5, No. 2, pp. 109–137, 1983.

107. Yun, S.-Y. and A. Proutiere, "Accurate Community Detection in the Stochastic Block Model via Spectral Algorithms", *arXiv preprint arXiv:1412.7335*, 2014.

108. Reichardt, J. and D. R. White, "Role Models for Complex Networks", *The European Physical Journal B*, Vol. 60, No. 2, pp. 217–224, 2007.

109. Karrer, B. and M. E. J. Newman, "Stochastic Blockmodels and Community Structure in Networks", *Physical Review E*, Vol. 83, p. 016107, 2011.

110. Decelle, A., F. Krzakala, C. Moore and L. Zdeborová, "Asymptotic Analysis of the Stochastic Block Model for Modular Networks and its Algorithmic Applications", *Physical Review E*, Vol. 84, No. 6, p. 066106, 2011.

111. Lind, P. G. and H. J. Herrmann, "New Approaches to Model and Study Social Networks", *New Journal of Physics*, Vol. 9, No. 7, pp. 228–228, 2007.

112. Newman, M. E. J., "Clustering and Preferential Attachment in Growing Networks", *Physical Review E*, Vol. 64, No. 2, p. 025102, 2001.

113. Xiang, J., K. Hu, Y. Zhang, M.-H. Bao, L. Tang, Y.-N. Tang, Y.-Y. Gao, J.-M. Li, B. Chen and J.-B. Hu, "Enhancing Community Detection by Using Local Structural Information", *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 2016, No. 3, p. 033405, 2016.

114. Jaccard, P., "Étude Comparative de la Distribution Florale dans une Portion des Alpes et des Jura", *Bulletin del la Société Vaudoise des Sciences Naturelles*, Vol. 37, pp. 547–579, 1901.

115. Danon, L., A. Diaz-Guilera, J. Duch and A. Arenas, "Comparing Community Structure Identification", *Journal of Statistical Mechanics: Theory and Experiment*, Vol. 2005, No. 09, p. P09008, 2005.

116. Lancichinetti, A., S. Fortunato and F. Radicchi, "Benchmark Graphs for Testing Community Detection Algorithms", *Physical Review E*, Vol. 78, No. 4, p. 046110, 2008.

117. Leskovec, J. and A. Krevl, *SNAP Datasets: Stanford Large Network Dataset Collection*, 2014, `http://snap.stanford.edu/data`, accessed at December 2018.

118. Peel, L., D. B. Larremore and A. Clauset, "The Ground Truth About Metadata and Community Detection in Networks", *Science Advances*, Vol. 3, No. 5, 2017.

119. Finch, S. and D. D. Lewis, *Reuters-21578 dataset containing collections in Reuters newswire in 1987*, 1997, `http://www.daviddlewis.com/resources/testcollections/reuters21578`, accessed at December 2018.

120. Ozgur, A. and H. Bingol, "Social Network of Co-occurrence in News Articles", *Computer and Information Sciences-ISCIS 2004*, pp. 688–695, 2004.

121. Ozgur, A., B. Cetin and H. Bingol, "Co-occurrence Network of Reuters News", *International Journal of Modern Physics C*, Vol. 19, No. 05, p. 689, 2008.

122. Newman, M. E. J., "The Structure of Scientific Collaboration Networks", *Proceedings of the National Academy of Sciences of the United States of America*, Vol. 98, No. 2, pp. 404–409, 2001.

123. Knuth, D. E., *The Stanford GraphBase: A Platform for Combinatorial Computing*, ACM, New York, NY, 1993.

124. Barrat, A., *The SocioPatterns Project*, 2008, `http://www.sociopatterns.org`, accessed at December 2018.

125. Isella, L., J. Stehlé, A. Barrat, C. Cattuto, J.-F. Pinton and W. Van den Broeck, "What's in a Crowd? Analysis of Face-to-Face Behavioral Networks", *Journal of Theoretical Biology*, Vol. 271, pp. 166–180, 2010.

126. Stehlé, J., N. Voirin, A. Barrat, C. Cattuto, L. Isella, J.-F. Pinton, M. Quaggiotto, W. Van den Broeck, C. Régis, B. Lina and P. Vanhems, "High-Resolution

Measurements of Face-to-Face Contact Patterns in a Primary School", *Plos One*, Vol. 6, No. 8, p. e23176, 2011.

127. Kiss, G., C. Armstrong, R. Milroy and J. Piper, "An Associative Thesaurus of English and its Computer Analysis", A. Aitken, R. Bailey and N. Hamilton-Smith (Editors), *The Computer and Literary Studies*, Edinburgh University Press., Edinburgh, 1973.

128. Colizza, V., R. Pastor-Satorras and A. Vespignani, "Reaction-diffusion Processes and Metapopulation Models in Heterogeneous Networks", *Nature Physics*, Vol. 3, No. 4, pp. 276–282, 2007.

129. White, J. G., E. Southgate, J. N. Thomson and S. Brenner, "The Structure of the Nervous System of the Nematode Caenorhabditis Elegans", *Philosophical Transactions of the Royal Society B: Biological Sciences*, Vol. 314, No. 1165, pp. 1–340, 1986.

130. Watts, D. J. and S. H. Strogatz, "Collective Dynamics of 'Small-world' Networks", *Nature*, Vol. 393, No. 6684, pp. 440–402, 1998.

131. Erdos, P. and A. Renyi, "On Random Graphs I.", *Publicationes Mathematicae Debrecen*, Vol. 6, pp. 290–297, 1959.

132. Barabasi, A. L. and A. Reka, "Emergence of Scaling in Random Networks", *Science*, Vol. 286, pp. 509–512, 1999.

133. Newman, M. E. J., *Network Dataset of Co-authorships Between Scientists Posting Pre-prints on the Condensed Matter E-Print Archive Between 1999-2005*, 2006, `http://www-personal.umich.edu/~mejn/netdata`, accessed at December 2018.

134. Duch, J. and A. Arenas, "Community Detection in Complex Networks Using Extremal Optimization", *Physical Review E*, Vol. 72, No. 2, p. 027104, 2005.

135. Richardson, M., R. Agrawal and P. Domingos, "Trust Management for the Semantic Web", *The Semantic Web-ISWC 2003*, pp. 351–368, Springer, Berlin, 2003.

136. Yang, J. and J. Leskovec, "Defining and Evaluating Network Communities Based on Ground-truth", *Knowledge and Information Systems*, Vol. 42, No. 1, pp. 181–213, 2015.

137. Kunegis, J., *Network Dataset of Catster/dogster Family Links/friendships*, 2017, `http://konect.uni-koblenz.de/networks/petster-carnivore`, accessed at December 2018.

138. Tasgin, M., *A Framework and Toolset for Community Detection in Complex Networks - Java Source Code Repository*, 2018, `https://github.com/murselTasginBoun/CommunityDetectionFramework`, accessed at December 2018.

139. Tasgin, M., *Community Detection Using Preference Networks - Source Code Repository*, 2018, `https://github.com/murselTasginBoun/CDPN`, accessed at December 2018.

140. Tasgin, M., *Community Detection Using Boundary Nodes - Java Source Codes*, 2018, `https://github.com/murselTasginBoun/CDBN`, accessed at December 2018.