Spring 2021

Overlapping Community Detection in Social Networks

Akshar Panchal
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the [OS and Networks Commons](#), and the [Other Computer Sciences Commons](#)

Recommended Citation

Panchal, Akshar, "Overlapping Community Detection in Social Networks" (2021). *Master's Projects*. 1011.
DOI: <https://doi.org/10.31979/etd.dpgH-2kx6>
https://scholarworks.sjsu.edu/etd_projects/1011

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Overlapping Community Detection in Social Networks

A Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Akshar Panchal

May 2021

© 2021

Akshar Panchal

ALL RIGHTS RESERVED

The Designated Project Committee Approves the Project Titled

Overlapping Community Detection in Social Networks

by

Akshar Panchal

APPROVED FOR THE DEPARTMENTS OF COMPUTER SCIENCE

SAN JOSE STATE UNIVERSITY

May 2021

Katerina Potika	Department of Computer Science
-----------------	--------------------------------

Christopher Pollett	Department of Computer Science
---------------------	--------------------------------

William Andreopoulos	Department of Computer Science
----------------------	--------------------------------

ABSTRACT

Overlapping Community Detection in Social Networks

by Akshar Panchal

Social networking sites are important to connect with the world virtually. As the number of users accessing these sites increase, the data and information keeps on increasing. There are communities and groups which are formed virtually based on different factors. We can visualize these communities as networks of users or nodes and the relationships or connections between them as edges. This helps in evaluating and analyzing different factors that influence community formation in such a dense network. Community detection helps in revealing certain characteristics which makes these groups in the network unique and different from one another. We can use such information to find trends in the network which might help in understanding complex systems.

In this project, we will study the problem of detecting local overlapping communities in a stream graph, by proposing a new metric that of common communities of the endpoints of a new edge. Moreover, we discover good seed nodes by finding an offline non overlapping community structure of a small sub graph as a preprocessing step. Additionally, we also evaluate these methods with different and extensive datasets. We experiment with a new web graph dataset along with some other more commonly used datasets. F1 score is used for datasets that have the ground truth. The proposed algorithm outperforms the traditional methods by 17%. For the new web graph dataset we use the overlapping modularity metric to evaluate our approach. This approach yields accurate modularity scores up to 0.71 for the web graph dataset increasing the accuracy by 10%. In the end, we discuss the approaches used and their results along with scope to improve for the future.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor, Prof. Katerina Potika, whose expertise in algorithms and social networks helped me through my graduate education and my master's project. Her knowledge and enthusiasm for exploring and diving deep into the world of social networks always inspired me to keep up with my research. Her advice, guidance and mentorship helped me and motivated me to complete this project. I can't thank her enough for her efforts in research, help and time for constantly pushing me.

I am grateful to have Prof. Christopher Pollett and Prof. William Andreopoulos for being on my defense committee and supporting this project. I would like to thank them for their time and efforts. Special thanks to my friend David Bui for his help with the web crawl dataset parser.

I would also like to thank all my friends and family for their constant support and motivation which always kept me inspiring to achieve more.

TABLE OF CONTENTS

CHAPTER

1	Introduction	1
1.1	Community Detection	2
2	Terminologies	4
2.1	Graph Terminology	4
2.2	Degree of a Node	5
2.3	Count Min Sketch	5
2.4	Edge Betweenness	6
2.5	Modularity for overlapping communities in directed graph . .	6
2.6	Community Participation	8
2.7	Overlapping Communities	8
2.8	Community degree	8
2.9	Common Community Score	9
2.10	Seed set nodes	10
3	Related Work	11
3.1	Non Overlapping algorithms	11
3.1.1	Girvan-Newman algorithm approach	11
3.1.2	Louvain algorithm	13
3.1.3	Minimum Spanning Tree approach	13
3.2	Overlapping algorithms	14
3.2.1	Community Detection via Seed-set Expansion on Graph Streams (COEUS)	14

3.2.2	Online local communities with motifs	16
3.2.3	OST algorithm	16
4	Implementation Plan	18
4.1	Problem Definition for Overlapping Communities	18
4.2	COEUS Algorithm	18
4.3	Community Detection via Seed-set Expansion on Graph Streams using Common Communities	23
4.3.1	Common Community Score Algorithm	24
4.3.2	COEUS algorithm with Common Community	25
4.3.3	Community updates with Edge Quality	30
4.3.4	Drop Tail Algorithm for size determination	31
4.4	Using seed set communities with offline community detection with COEUS	34
4.5	COEUS with common communities with overlapping modular- ity score	35
5	Experimental Results	37
5.1	Dataset	37
5.1.1	Amazon	37
5.1.2	DBLP	38
5.1.3	Web Crawl	39
5.1.4	YouTube	40
5.2	COEUS	40
5.3	COEUS using Common Communities	42
5.4	Using seed set communities with offline community detection with COEUS	43

5.5	Web Crawl dataset results	44
5.6	Comparing experimental results	46
6	Conclusion and Future Work	49
	LIST OF REFERENCES	52

LIST OF TABLES

1	Dataset graph details	38
2	Amazon dataset: COEUS	41
3	DBLP dataset: COEUS	41
4	YouTube dataset: COEUS	41
5	Amazon dataset: COEUS with Common Communities	42
6	DBLP dataset: COEUS with Common Communities	42
7	YouTube dataset: COEUS with Common Communities	42
8	Amazon dataset: COEUS with initial seed set communities	43
9	DBLP dataset: COEUS with initial seed set communities	43
10	YouTube dataset: COEUS with initial seed set communities	43
11	Comparison between original COEUS and proposed COEUS with common communities for Amazon dataset	49
12	Comparison between original COEUS and proposed COEUS with common communities for DBLP dataset	50
13	Comparison between original COEUS and proposed COEUS with common communities for YouTube dataset	50

LIST OF FIGURES

1	Graph structure with 6 nodes connected by different edges	4
2	Graph structure with degree 4 for node 3	5
3	Count Min Sketch mapping process	6
4	Edge connecting two smaller densely connected communities . . .	7
5	Overlapping communities C_1 and C_2 sharing a node 10	9
6	COEUS algorithm	20
7	Stream of edges as input from undirected graph to a set of overlapping communities defined with 3 seed nodes.	21
8	Pruning algorithm	23
9	Common community score algorithm	26
10	Drop tail algorithm	34
11	Community graph for web crawl dataset using COEUS	45
12	Community graph for web crawl dataset using COEUS with Common Communities	45
13	Community graph for web crawl dataset using offline community detection with seed set communities	46

CHAPTER 1

Introduction

The advancement and rise of new technology has made it possible to connect with anyone around the world using technologies that didn't exist few decades back. Social Network is a network of interactions and relationships between people. Social networking sites like Linked In, Facebook, Instagram and Twitter provide a platform for users to interact, share their thoughts and express themselves while being in touch with their connections. Social networking sites are important to connect with the world virtually and share information.

As the number of users accessing these sites increase, the data and information keeps on increasing. This data can be leveraged to extract different statistical and structural patterns in the way communities are formed. There are communities and groups which are formed virtually based on different factors in the network. This helps in observing and predicting certain characteristics which are useful in certain operations. We can visualize these communities as nodes representing users and the relationships or connections between them as edges. This helps in evaluating and analyzing different factors that influence community formation in such a dense network.

A lot of research is committed to the community detection task and we will focus our attention in the detection of overlapping communities instead of non overlapping communities [1], where each user can belong to more than one community. We will focus on exploring the need to find efficient ways to analyze and find overlapping communities in such big data networks.

Analyzing, retrieving and storing huge data from the social networks becomes tedious as the size of network keeps on increasing. Therefore, we need effective and fast methods for processing the increasing size of the network as well as for data retrieval. Community Detection is one such method which can help in identifying key areas relevant to this problem and provide an efficient solution.

1.1 Community Detection

Communities are dense group of nodes in a network which share common traits or similar characteristics which distinguishes them apart from other nodes in the same network. The nodes which share common characteristics or properties are often densely connected to each other. This helps us in finding communities which exhibit certain properties which are shared amongst these tightly connected nodes. In some cases the nodes exhibit common properties to more than one group and they overlap with other communities. Detecting such overlapping communities has been a popular research subject for many researchers since the last decade.

Community detection helps in revealing certain characteristics which makes these groups in the network unique and different from one another. We can use such information to find trends in the network which might help in understanding complex systems. There are a lot of community detection algorithms which interpret the graph structure and evaluate those to detect communities. Some of those deal with finding centrality measures with respect to the edges between the nodes. Another approach is to expand around seed node and then keep forming communities. The community detection task is difficult and costly, since it finds communities and uncovers the relations amongst multiple communities. Some of the existing methods use different heuristics such as approximate page ranks [2], minimum spanning trees [3] and a

streaming input [4] for graphs with seed set expansion. These methods have different approaches with respect to the way graph structure is parsed.

The paramount objective of this project is to extend the COEUS algorithm [4] to local communities with seed nodes to discover overlapping communities around the seed nodes and optimize it. Our goal will be to improve the overall quality of the formed communities formed in a social network, as well as optimizing the needed resources, like time and space. Additionally, this new approach would be tested on different benchmark datasets which will help in evaluating the performance and efficiency of our approach.

CHAPTER 2

Terminologies

This chapter highlights some basic concepts and terminologies used in graph structures and community detection. Let us have a look at some of them.

2.1 Graph Terminology

A graph structure is formed using a set of vertices. These vertices are connected to each other by edges. It is denoted by $G = (V, E)$ where V and E represent the set of vertices and edges respectively.

Figure 1 shows a graph structure having a set of vertices (1,2,3,4,5,6) along with the edges which connect different vertices (1-2, 1-3, 1-5, 2-3, 2-5, 3-5, 3-4, 4-6, 6-2).

Figure 1: Graph structure with 6 nodes connected by different edges

2.2 Degree of a Node

The degree of a node N may be specified as the number of connections that node N has with the other nodes in the graph structure. Figure 2 shows a graph structure where node 3 has degree 4 as it is connected to 4 different nodes.

Figure 2: Graph structure with degree 4 for node 3

2.3 Count Min Sketch

Count Min Sketch is an efficient data structure which is probabilistic in nature. It is effective in representing high-dimensional vectors as it operates with sub linear space. It is useful where we have high volume of input data streamed at high rates. Count Min Sketch also helps in updating and manipulating data with high accuracy [5].

It is a 2 dimensional array of depth d and width w . It uses hash functions h ($h_1, h_2, \dots, h_n$) to map the data into the matrix.

When an update operation (i, u) is performed, the corresponding hash functions try to update the count values by one in each of the rows of count min sketch by

Figure 3: Count Min Sketch mapping process

choosing the appropriate column.

2.4 Edge Betweenness

Edge Betweenness is an evaluation of the amount of network traffic that crosses the given edge. It also indicates how important the edge is for all the shortest path in the graph network. Edge Betweenness can be expressed as a ratio of number of shortest path passing through the edge between any two vertices v_1 and v_2 over all the shortest path between v_1 and v_2 . This indicates how important an edge is. Edges with maximum traffic flow would have higher edge betweenness value.

In Figure 4 there is an edge that acts as a bridge between two different sub structures of the graph [6]. The removal of this edge will cause a partition in the graph resulting in two densely connected graphs or communities. The edge betweenness of this edge is 16.

2.5 Modularity for overlapping communities in directed graph

Modularity is the measure of how well the communities are formed in the network. It is a scalar value ranging from -1 to 1. It is measured as a ratio of number of edges that are part of the same community compared to the expected number of edges

Figure 4: Edge connecting two smaller densely connected communities

under a random distribution [4]. Networks with large modularity score have better community partitioning.

In [7] the author has proposed a unique way of calculating the modularity for overlapping communities in a directed graph. As each node in graph can be a part of multiple communities, each node is assigned a value called belonging factor with each community denoted by $\alpha_{i,c}$ for any i th node in community c . The modularity M of graph G with m edges and C communities for and edge l going from node i to j is given by:

$$M = \frac{1}{m} \sum_{c \in C} \sum_{i,j \in V} (\beta_{l(i,j),c} A_{ij} - \frac{\beta_{l(i,j),c}^{out} k_i^{out} \beta_{l(i,j),c}^{in} k_j^{in}}{m})$$

where,

$$\beta_{l(i,j),c} = \frac{\sum_{i \in V} f(\alpha_{i,c}, \alpha_{j,c})}{|V|}$$

where A_{ij} is the adjacency matrix, k_i and k_j are degrees for nodes i and j , and function $f(a_{i,c}, a_{j,c})$, depicts the way we choose to weight the contribution of each edge to the modularity calculated for community c .

2.6 Community Participation

Community participation in a graph structure is defined as a node's participation level in a community. In simple terms, community participation is a fraction of the adjacent nodes in the graph structure which belong to the community [4]. Community Participation of node u in edge (u, v) in community C is denoted by:

$$cp(u) = \frac{|\{(u, v) \in E : v \in C\}|}{|\{(u, v) \in E\}|}$$

Nodes with higher values of community participation are well connected to the community while the nodes with lower values are less connected to the community.

2.7 Overlapping Communities

Overlapping communities can be defined as group of communities that share some common nodes amongst themselves [8]. The nodes belong to one or more communities based on certain characteristics. As shown in Figure 5, consider a node 10 which is part of a community C_1 as well as C_2 . Then communities C_1 and C_2 are said to be overlapping communities.

2.8 Community degree

Community degree in a sub-graph or a community of each node is the degree of each node within that community. Each node in a graph structure has a degree, but within it's community the number of nodes connected to it might be different.

Figure 5: Overlapping communities C_1 and C_2 sharing a node 10

2.9 Common Community Score

Common Community Score is relatively an efficient way to compute the participation level of a node in a community. For each node in the community the score is computed and stored with them for faster retrievals and updates.

Common community score of an edge (u, v) is calculated as:

$$\text{Common Community score} = \frac{\text{intersection_count}}{\text{intersection_count} + \text{union_count}}$$

where,

$\text{intersection_count}$ = Total number of communities having both node u and v

union_count = Total number of communities having either node u or v

2.10 Seed set nodes

Before we expand our network of communities, we need to start from an initial node which is considered to be the first node in the community. These nodes are called seed set nodes as they are the first members of the community.

CHAPTER 3

Related Work

This chapter discusses some of the popular existing methods for community detection. Since we are looking for robust approaches and possibilities to increase the efficiency, the methods proposed are useful in analyzing different algorithms. Each approach uses different graph characteristics that they use. These methods are diverse with respect to their implementation and structural approaches. Each approach has some drawbacks which are overcome using an improvised algorithm.

3.1 Non Overlapping algorithms

This section discusses some of the algorithms which are non overlapping in nature. Non overlapping algorithms have a property that the nodes from the graph structure are unique to each community and are not present in any other communities [9]. Hence, we can say that each node in a graph is part of just one community in the entire network. This methodology ensures that the communities are formed without considering multiple characteristic properties of each node.

3.1.1 Girvan-Newman algorithm approach

In this paper [10], Girvan, Newman et al. has devised a new algorithm in order to overcome the drawbacks of the existing approach for community detection. The paper discusses traditional methods along with their drawbacks and a new algorithm is proposed to overcome those.

3.1.1.1 Traditional methods

Hierarchical clustering method is one of the traditional methods described in [10] for community detection. Initially, we calculate the weights of all the nodes and then sort them according to their weights. As per their weights, the nodes are linked together. As we go on linking the nodes, a hierarchical structure is formed. There are two ways to define edge weights in the network. The first one is by tracking node independence where two paths are node independent if they do not have any common points but share the same end points. The second way is to find all the paths between nodes.

This approach has a huge drawback for nodes with just one edge connected with the rest of the graph structure. This type of nodes are not categorized into any group of communities. They should be grouped with the cluster to which they are already connected.

3.1.1.2 Girvan-Newman algorithm

The above drawback is solved by the proposed Girvan-Newman algorithm [10]. The algorithm find important nodes in the graph using edge betweenness instead of vertex betweenness. It first calculates edge betweenness and then removes edges based on the highest edge betweenness ratio. As the edge is removed, the edge betweenness and shortest path of other edges change. We keep on repeating this process until every edge in the network is removed.

If a network has some communities which are connected just with a single edge then during the process, such communities are identified. The algorithm calculates the maximum modularity of the communities in the end.

3.1.2 Louvain algorithm

In [11] the author has proposed a novel non overlapping community detection algorithm which is faster as compared to [10]. The algorithm works in two phases. In the first phase the modularity value of each node is calculated by including it within the neighbouring communities. The community offering the highest modularity value is selected and the node forms a part of that community. The second phase builds a meta graph where communities are represented as meta nodes. These two phases are iteratively repeated on the meta graph.

The drawback of this algorithm is that smaller communities are ignored [12]. Communities which exhibit the modularity value less than the threshold may not be included in the final set of communities. This effect is observed usually when you try optimizing the modularity to detect communities and is often referred to as resolution limit effect [13]. The louvain algorithm is widely used in a distributed fashion with respect to community detection [14].

3.1.3 Minimum Spanning Tree approach

This approach considers an algorithm for finding over-lapping communities via spanning tree computations [3]. A modified ST algorithm is used to improve the quality of the results where the edge weights in the graph are modified. This algorithm is quite helpful when it comes to running it in a distributed fashion which is an advantage for a network of huge size.

3.1.3.1 ST algorithm

This algorithm avoids the time intensive process of computations of edge betweenness on the input graph. It considers an unweighted undirected graph and

then assigns weight to each edge e of the graph. Using these weights, we calculate a minimum spanning tree using Kruskal’s algorithm. The edge betweenness value for each node is calculated and the edge with maximum weight is removed. The graph is partitioned and the modularity score is calculated and stored. This procedure is repeated several times till the network has no more edges left. The best part of this algorithm is that it avoids calculating edge betweenness on the entire graph.

3.2 Overlapping algorithms

This section discusses overlapping algorithms which are more generic as compared to the non overlapping algorithms [8]. The main characteristics of overlapping algorithms are that it can help detect communities with multiple point of interests where each node can be a part of more than one community. In real world scenarios as well, we tend to be connected to different group of people at different places based on our preferences.

This makes sure that all the nodes in one particular community are classified with respect to one property whereas the same node can be present in some other community based on a different property which makes it relatable with practical scenarios. We will base our proposed algorithm with respect to overlapping algorithms in the next chapter.

3.2.1 Community Detection via Seed-set Expansion on Graph Streams (COEUS)

This section discusses the COEUS algorithm for community detection using graph streams [4]. It operates on a stream of input edges in a graph structure rather than analyzing entire input graph at a time. Hence the problem of storing the en-

tire graph and consuming space is already resolved with this approach. The graph stream manages large inputs by processing one edge at a time and thereby consuming significantly less amount of memory. As the space for storing processed nodes needs to be efficient enough for quick manipulations, we utilize COUNT-MIN sketch data structures to store count as well as participation values for each nodes in the graph network.

This algorithm is different from the the two proposed before. The algorithm works on graph stream as input. With respect to other algorithms that use an entire graph as an input, this algorithm is very competitive.

- In this phase, the communities are initialized using the seed sets. The number of seed sets are equivalent to the number of communities we wish to find. We create seed set and hold all the nodes with respect to that seed node together so that we can efficiently query the existence of a node with respect to the community.
- This phase processes the input stream of graph nodes one by one. Since the size of graph is huge we process degree of each node along with its degree in each community. We increment each adjacent node's degree by one for each incoming edge. For each community, we examine if all the nodes in it are a member of the community. If yes, the degree of the other node is incremented by 1. If it is not a member of the community, we add it back to the community.
- COEUS can be stopped at any moment of time as the input is graph stream and member node with respect to any community are available at any point of time. COEUS is capable of handling infinite stream of nodes as well.

3.2.2 Online local communities with motifs

In [15], the author uses the concept of [4] and proposes a new method for online local community detection which considers higher order motifs which are triangles of nodes. Since, the approach by [4] does not take triangle of nodes into account, the information contained within such structures is lost.

In the proposed method, a stream of incoming edges of triangle of nodes is processed which has 3 nodes. Each of these nodes are added in to a set of initial seed set nodes in a set to extend it. Once all the nodes are streamed and processed, we get a set of communities with corresponding seed sets.

The algorithm is similar to [4] where the only change is instead of an edge with two nodes, we process three nodes at a time and update their corresponding community participation values. Instead of updating the degree of each node in the triangle by one, it updates the degree by two as each node in the triangle is connected to two other nodes. The author has proposed this variation of the COEUS which can handle only triangle of nodes and fails to consider the streaming edge inputs with respect to two nodes.

3.2.3 OST algorithm

This part of the algorithm deals with modifying ST algorithm from [3]. The modification is done to detect the over-lapping communities hence this algorithm is called OST. The initial part is same as ST algorithm where we take unweighted undirected graph as input and use Kruskal's algorithm to computer minimum spanning tree. The edge betweenness is calculated and assigned to each edge. Next we compute the max split betweenness and compare that with max edge betweenness. If the max split betweenness is greater than max edge betweenness then the vertex is split. The

algorithm repeats itself till it terminates when all the edges of the tree are removed.

CHAPTER 4

Implementation Plan

We will first discuss the problem definition in Section 4.1 and then go in detail with respect to the proposed COEUS algorithm in Section 4.2. Later on we propose additional methods to improve the F1 score.

4.1 Problem Definition for Overlapping Communities

Given an unweighted undirected graph G with N nodes and E edges, find different over-lapping communities which lie in the network. Consider the input graph consisting of N users which represent nodes and E relationships which represent edges between the nodes. The Community Detection problem is to find optimal number of over lapping communities in the entire graph network. The solution to this problem is achieved by introducing a new metric to measure the participation of a node in a community. Hence, the proposed implementation would be the extension of the COEUS algorithm based on seed set expansion.

4.2 COEUS Algorithm

This section discusses the COEUS algorithm [4] on the basis of which the proposed algorithm is based. The algorithm takes two parameters as input, stream of input edges and a set of seed nodes.

We keep on adding nodes in the respective set of communities $C=\{C_1, C_2, \dots, C_n\}$ where each community C_i belongs to the corresponding seed set K_i . The entire flow of COEUS is illustrated in the Figure 6. The algorithm can be divide into two parts namely grouping the communities and pruning the excess nodes from a community.

As the algorithm processes the graph stream, we need to keep track of following three parameters to effectively determine new communities with respect to incoming edges:

1. Degree of each node.
2. Community Degree of each node.
3. Communities formed.

As shown in Figure 6, the COEUS algorithm is as follows:

- Before we start processing stream of edges, we need to use the seed sets to initialize the communities. This ensures that we create different sets for communities with respect to each seed set. We initialize an empty set for communities which will be created as C' .
- As long as we have a stream of input edges, we need to process and add the nodes of the respective edges into the correct communities in C' . As for COEUS, a stream of the input edges will keep on updating the community structure formed at any point of time. This helps us to update the graph structure without having any limitations on when the nodes are processed and in what fashion.
- If the COEUS is initialized with three seed nodes then we have three communities to begin with. Consider an example in Figure 7 where we have incoming edges (2,3) and (9,8) into an existing set of communities $\{1, 2, 8\}$, $\{3, 4, 8\}$ and $\{6, 7, 9\}$. As the graph size is very large, we tend to focus on individual degree of each node along with its degree in the community.

Figure 6: COEUS algorithm

- Increment the degree of each node from the incoming edge. We store the degree of each community in Count Min Sketch as it is highly effective to manipulate and update the community degree. The Count Min Sketch data structure will contain mapping of all the communities along with the degrees.
- The degrees of each node in the edge are incremented as well in order to keep track of number of nodes that are adjacent to each node. If we consider our

Figure 7: Stream of edges as input from undirected graph to a set of overlapping communities defined with 3 seed nodes.

example, with the arrival of edge (9,8) all the communities will check for the nodes 8 and 9. As we can see node 8 is part of two communities which are overlapping. Hence, we add node 9 to both these communities and increment the degree of both 8 and 9. We also need to increment the community degree of the overlapping communities in the Count Min Sketch.

- When we keep adding and growing communities with a stream of edges, often times the communities grow and scale up to be huge. The diameters of communities in real world networks are quite small and they tend to decrease as the size of network grows. In COEUS, we take care of the growing community size by having a predefined window size W .

- Once all the edges in the stream are processed and the window closes, COEUS prunes the communities to make sure the community sizes are less than W . This ensures that the nodes which are a part of the community are tightly bonded to the community and its members. This process helps in removing all the loosely attached nodes in the network. The pruning process is outlined in Figure 8.
- The community participation values of each node which are part of the existing communities are used to determine how well they are attached to the community. If the size of community formed exceeds the maximum allowed window size W , the pruning algorithm is executed. This will make sure that none of the communities in the end have a size more than W .
- The algorithm uses a min heap to add all the edges in a community based on their community participation values. If the min heap reaches the size limit W , it then removes the nodes with the least community participation values and replaces it with the nodes with values more than the replaced node. In the end we are left with a min heap of size W which contains all the nodes which are most relevant and related to the community.
- As the number of edges processed in a stream might be high, we need to make sure that the size of communities formed are within limits. The algorithm takes care of the community size by keeping track of number of nodes processed at any given time. If the number of nodes processed cross the threshold limit, we go ahead and prune the community. Pruning is performed based on the community participation values of the nodes.

Figure 8: Pruning algorithm

4.3 Community Detection via Seed-set Expansion on Graph Streams using Common Communities

This section will discuss the proposed implementation of COEUS using common communities as a metric to measure how well connected the nodes are with respect to the communities formed. For community detection, there is a need to classify and detect communities based on factors that ensure the quality of communities formed is

enhanced. With respect to [4] we saw that community participation was the driving factor for detecting communities. The common community feature will replace the community participation score which was used in the previous section.

The proposed method using common communities has a few advantages over the community participation score. Few of them are:

1. The time needed to compute common communities has less overhead as compared to community participation.
2. The common community score is maintained for every node and updates itself once a better score is computed based on incoming edges.

For this approach, as we are evaluating continuous stream of input edges from a graph structure, we need to keep track of some key aspects to effectively form communities using common community score. We store the following attributes at any point of time:

1. Degree of each node.
2. Degree of the communities formed.
3. Nodes along with their common community score.

4.3.1 Common Community Score Algorithm

The overview of the Common Community Score algorithm is shown in Figure 9. It is determined as follows:

- Consider incoming stream of edges with an edge (u, v) . We need to check if the nodes u and v are present in any communities.

- For each communities in the set with respect to seed set nodes, check if u and v are both present in the same community.
- If we find node u and v to be a part of the same community then we increment the intersection value.
- If nodes u and v are not a part of the same community then check if the community contains either one of the nodes. Increment the union value if the community has node u or v .
- Check if the value of the sum of intersection and union values is greater than 0 or not.
- If the sum is greater than zero, calculate the common community value and store the same with node u and v .
- The common community values are stored with each node and are updated only when we get a value greater than the current common community score for each node.

4.3.2 COEUS algorithm with Common Community

This section discusses the COEUS proposed in the previous section with common community as a measure to form communities with quality edges. The proposed algorithm takes two inputs:

1. A set of community seed nodes.
2. A stream of input edges.

Figure 9: Common community score algorithm

Line 1-7 of Algorithm 1 processes all the seed set nodes which are present in K' and creates communities as a starting point with the nodes. The communities created by adding initial seed set nodes are then added into a set which contains the resulting desired communities. In this way, we manage to store the communities efficiently in a way where querying a node's existence is simplified. Once we have processed all the seed set nodes and initialized the communities in the set, COEUS

processes a continuous stream of input edges. The entire process of streaming and segregating nodes into their respective communities is performed in this step as shown in Algorithm 1 from Line 8-30.

When we process an edge with nodes u and v , we first increment their degrees by 1. In order to calculate common communities with respect to each node, we need to be sure that as soon as we have an incoming node it is updated. We initialize counter to keep track of communities which are common to nodes u and v along with the communities that either one of them are part of. Lines 13-20 iterates over all the communities we have till now and tries to find the common as well as disjoint communities with respect to nodes u and v . We increment the intersection and union counters as per the conditions.

If the nodes u and v are not present in the graph, then we might have the counter values of intersection and union to be zero. In this case, the common community score is not updated. If the sum of intersection and union count is greater than zero then we calculate the common community score as on Line 23. The common community score is the same for both the nodes. Once we have the score, we need to update the score in the Count Min Sketch.

The Algorithm 2 manages updating the common community scores of nodes into the count min sketch and updating the community degrees as well. We iterate over all the communities in the set and update the degrees of u and v in the count min sketch by 1. Then we examine if each node is a part of the community which we wish to extend. If the node is a part of the community, then the community degree of the other node is incremented. Along with updating the community degree, if the other node is not a part of the community, we then add the other node to the respective community.

Algorithm 1: COEUS Simple with Common Communities

Input : *graph*: The input graph stream

community_seed_set: Set of community seed sets K'

Output: *set_of_communities* C' : Set of communities

```
1 for {seed_set_nodes} in  $K'$  do
2    $C \leftarrow \{\}$ ;
   // Create a community for each seed node
3   for seed_node in  $K$  do
4      $C[\text{seed\_node}] = 1$ ;
5   end
6    $C'.\text{put}(C)$ ;
7 end
8 for {edge( $u, v$ )} do
9   degree_count_min_sketch[ $u$ ] += 1;
10  degree_count_min_sketch[ $v$ ] += 1;
11  union_count = 0, intersection_count = 0;
12  for  $C$  in  $C'$  do
13    if  $u$  in  $C$  AND  $v$  in  $C$  then
14      intersection_count += 1;
15    end
16    if  $u$  in  $C$  OR  $v$  in  $C$  then
17      union_count += 1;
18    end
19  end
20  common_community_score = 0;
21  if intersection_count + union_count > 0 then
22    common_community_score =
      intersection_count / (intersection_count + union_count);
23  end
24  update_communities( $C'$ , edge( $u, v$ ), common_community_score,
    degree_count_min_sketch);
25  processed_elements += 1;
26  if processed_elements >  $W$  then
27    // Prune the community is size exceeds  $W$ 
     $C \leftarrow \text{prune}(C, s)$ ;
28  end
29 end
30 return  $C'$ ;
```

Algorithm 2: Updating Common Communities with Count Min Sketch

Input: $nodes(u,v)$: Current edge nodes being processed
 $common_community_score$: Common Community score for node
 $degree_count_min_sketch$: Count Min Sketch for degree of nodes
 C' : Set of communities formed

// Iterate over each community in the set and calculate the
 common community score

```
1 for  $C$  in  $C'$  do
2   if  $u$  in  $C$  then
3     |  $degree\_count\_min\_sketch[v] + = 1$ ;
4   end
5   if  $v$  in  $C$  then
6     |  $degree\_count\_min\_sketch[u] + = 1$ ;
7   end
8   // Check if the common community score of node u is less
      than the current
9   if  $u$  in  $C$  then
10    | // Update the common community score of node u in the
      community
11    | if  $u.common\_community\_score < common\_community\_score$  in  $C$ 
      then
12    | |  $C.put(v, common\_community\_score)$ ;
13    | end
14  end
15  // Check if the common community score of node v is less
      than the current
16  if  $v$  in  $C$  then
17    | // Update the common community score of node v in the
      community
18    | if  $v.common\_community\_score < common\_community\_score$  in  $C$ 
      then
19    | |  $C.put(v, common\_community\_score)$ ;
20    | end
21  end
22 end
```

We need to check and update the common community scores as well. Lines 8-18 in Algorithm 2 checks if the node is in a community and then updates the common community score for that node.

Algorithm 3 takes care of the pruning part of communities. In real world, the communities are small with respect to the entire graph structure and it makes sense to keep the community size smaller as the network scales and grows. The algorithm proposed will keep on adding nodes to the communities as a stream. hence, we need to make sure that the communities formed do not end up scaling into large graph structure themselves. We have a window size W which is the maximum graph size that a community can be grown to.

We achieve this by pruning each communities individually when the size exceeds the maximum window size W . Lines 2-11 in Algorithm 3 iterates over all the nodes in the community and adds them in to a min heap with respect to their common community values. Nodes with higher common community values are well connected with the community. Hence we make sure that while adding the nodes to the heap, we only add if the heap size is less than W . If not then we compare the common community values of the latest node with the node with the least common community value on top of the heap. We adjust the nodes accordingly and in the end return the community with the most well connected nodes in it.

4.3.3 Community updates with Edge Quality

The edge quality updates in [4] with respect to adjacent nodes helps in increasing the overall F1 score where we make sure that we consider the community participation score. The process outlined in [4] checks if an adjacent node is a member of the community and if it is, it just updates the community degree of the adjacent node by 1. In this case, we miss out on looking at the level of involvement of the adjacent nodes.

Algorithm 3: Pruning Algorithm

Input : *size*: Maximum size of the community
 C: Community being pruned
Output: *set_of_communities C*: Set of communities

```
1 minHeap  $\leftarrow$  [];  
  // Iterate over each node in the community and calc  
2 for node in C do  
3   | common_community_score = C.get(node);  
4   | if minHeap.size() < max_size then  
5   |   | minHeap.push(node, common_community_score);  
6   | else  
7   |   | if common_community_score > minHeap[0] then  
8   |   |   | minHeap.pop();  
9   |   |   | minHeap.push(node, common_community_score);  
10  |   | end  
11  | end  
12 end  
13 return set(minHeap);
```

All the nodes increment their adjacent node's degree by 1 in a community without considering how well connected the node is in the community. In our proposed algorithm we improve over the edge quality by considering updating the adjacent node's community degree with common community score instead of just incrementing the degree by 1. Algorithm 4 describes the process of updating the adjacent nodes' community degree by common communities.

4.3.4 Drop Tail Algorithm for size determination

This section describes an algorithm from [4] and extends that to use common community score. In COEUS, as nodes are processed, size of the community keeps on increasing and we need to efficiently find a way to manage the size of the community.

Algorithm 5, identifies nodes that exhibit low common community values in a community and removes them automatically without a set window size. In the

Algorithm 4: Update Community by common community score for adjacent nodes

Input: $nodes(u,v)$: Current edge nodes being processed
 $common_community_score$: Common Community score for node
 $community_count_min_sketch$: Count Min Sketch for community degree
 C' : Set of communities formed

```

// Iterate over each community in the set and calculate the
// common community score
1 for  $C$  in  $C'$  do
2   if  $u$  in  $C$  then
3     |  $community\_count\_min\_sketch[v] += 1$ ;
4   end
5   if  $v$  in  $C$  then
6     |  $community\_count\_min\_sketch[u] += 1$ ;
7   end
8   // Check if the common community score of node u is less
8   // than the current
9   if  $u$  in  $C$  then
10    |  $C.put(v, common\_community\_score)$ ;
10   end
11  // Check if the common community score of node v is less
11  // than the current
12  if  $v$  in  $C$  then
13    |  $C.put(v, common\_community\_score)$ ;
13   end
14 end

```

previous Algorithm 1, we saw that we had a fixed windows size for pruning the large communities. But consider a practical scenario where we would have no idea on the ideal community size. We might need a process which would handle dropping off nodes less connected to the community. Figure 10 shows mapping of common community score along with number of nodes. We see that there are nodes in a community that will have relatively lower values of common community and will form a tail like structure which can be dropped from the community.

The graph shows nodes which are ranked based on their common community

Algorithm 5: DropTail algorithm for different community size

Input: *community_count_min_sketch*: Count Min Sketch for community degree
 C': Community from a set of communities

```
1  $C \leftarrow \{reverseSort(C)\};$ 
2  $totalDifference \leftarrow \{0\};$ 
3  $previous \leftarrow \{0\};$ 
4  $size \leftarrow \{sizeOf(C)\};$ 
  // Iterate over each node in the set and calculate total
  difference
5 for  $c$  in  $C'$  do
6   if  $previous > 0$  then
7      $totalDifference \leftarrow \{getCommunityScore(c) - previous\};$ 
8   end
9    $previous \leftarrow \{getCommunityScore(c)\};$ 
10 end
11  $averageDifference \leftarrow \{totalDifference/size - 1\};$ 
12 for  $c$  in  $C'$  do
13   if  $previous > 0$  then
14      $difference \leftarrow \{getCommunityScore(c) - previous\};$ 
15   end
16    $previous \leftarrow \{getCommunityScore(c)\};$ 
17   if  $difference < averageDifference$  then
18      $C.remove(c);$ 
19   else
20     break;
21   end
22 end
```

score. We can see that the nodes that exhibit lower common community scores

We need to find a way to form clusters of nodes which have low common community participation values first. To do this we calculate average distance between two nodes with respect to their common community values as shown from Line 5-9 in Algorithm 5. Once we have the average difference between two nodes in the community, drop tail will iteratively find the nodes with the common community score difference of more than the average difference as shown from Line 12-21. Once it finds

Figure 10: Drop tail algorithm

the node, it will remove the same from the community.

4.4 Using seed set communities with offline community detection with COEUS

In this section, we will look at an approach which considers having offline communities with a few seed set nodes initially. Once we have initial communities, we can stream input edges similar to the COEUS with common communities approach. In this approach [16], we have an offline setting for forming communities with a few nodes from the dataset.

This ensures that we have a set of communities to start with instead of random seed sets as we did in COEUS with common communities. The advantage with this approach is that the communities formed with initial nodes might have more than one node and once we stream the input edges, it might help for a better classification of the nodes within the communities.

The algorithm considers input dataset from a file where each edge is represented as pair of two nodes separated by space. We limit the dataset to just $m/100$ edges, where m is the size of the dataset. This helps us to run the COEUS with initial communities instead of random seed sets. Once we have few initial communities to begin with, we run COEUS with common communities similar to the approach discussed in the previous section.

4.5 COEUS with common communities with overlapping modularity score

In this section, we will discuss a modification in our proposed algorithm in the way it processes the output accuracy score. The algorithm calculates the F1 score in the end by comparing the community set with the ground truth communities. Since, we are using publicly available datasets we can utilize ground truth communities available. This approach however would be invalid in case of a dataset without ground truth communities.

We need to address the way datasets without ground truth communities are processed. The approach described in [7] helps our proposed algorithm to process datasets without ground truth. The proposed process is as follows:

- We first run our COEUS with common communities algorithm using a dataset without ground truth.
- Once we have the community set output after running the algorithm, instead of calculating the F1 score, we store all the community sets for processing.
- We calculate the overlapping modularity score [7] for the graph from the community sets formed.

- Once we have the overlapping modularity scores with respect to the graph, we can plot the graph for visual representation of the communities within the streamed graph structure.
- For visual representation, we can export the file of community sets and add it to Gephi¹. The tool allows to set different color and size options for visualizing the output of communities formed in a graph.

¹<https://gephi.org/>

CHAPTER 5

Experimental Results

In this chapter, the outcome of the algorithms along with experimental results are discussed. In Section 5.1 we will discuss the public datasets [17] which are consumed for the experimental outcomes. In Section 5.2, we will assess the results obtained by using the datasets on COEUS algorithm and in the end we will evaluate the results of the datasets on our proposed COEUS algorithm with common communities. In the final Section 5.4, we will compare and contrast the results obtained using different algorithms and discuss the shortcomings.

5.1 Dataset

Our experimental analysis includes 3 datasets, Amazon, DBLP and YouTube. The Amazon dataset [17] is based on the data crawled from Amazon with respect to a product feature. The DBLP dataset [17] is a co-authorship network of connected authors. The Web Crawl dataset is extracted using a web crawler which contains information about web pages and the associated links with it. The YouTube dataset [17] contains information about web pages and the associated links with it. All the datasets contain undirected and unweighted edges in the network. Table 1 provides basic details for datasets with statistics.

5.1.1 Amazon

The Amazon dataset from the SNAP library [17] is a co-purchasing network which consists of graph network formed by crawling the amazon website. The graph structure is based on the 'Customers who bought this item, also bought this' product

Table 1: Dataset graph details

Dataset	Nodes	Edges
Amazon	334,863	925,872
DBLP	317,080	1,049,866
YouTube	1,134,890	2,987,624
Web Crawl	1,977,975	2,484,651

feature.

If any product m is purchased along with product n , the network consists of an edge from m to n . The ground truth communities are defined with respect to each product category. The ground truth communities with less than three nodes are excluded from the dataset. Each connected component is considered separately with respect to each product category.

5.1.2 DBLP

The DBLP dataset is obtained from the SNAP library [17] which consists of a comprehensive list of computer science research papers in the form of bibliography. The dataset is derived by forming a relationship between co authors if they have contributed to one or more research paper together.

An individual ground truth community for this dataset is defined by factors like publication venue e.g. journal or conference name where the co authors published the research paper. This forms a single community. Each connected component is considered separately with respect to each ground truth community. The ground truth communities with less than three nodes are excluded from the dataset.

5.1.3 Web Crawl

This dataset comprises of webpages crawled from web crawler called Common Crawl ¹ available online. The web crawler consists of raw petabytes of data collected since 2008. It consists of extractions and raw data from web pages.

The November 2015 crawl archive consists of all types of file formats like WARC, WAT and WET. These files contains information of web pages which are linked to each other via web pages. The dataset contains 188,721,679 number of pairs of web pages. These webpages are parsed on the basis of particular file format which is used as input as a stream of edges for COEUS.

The dataset after extraction using a parser ² has a format where each line contains two web page links separated by space which are considered as graph nodes. These web pages are used as a stream of input nodes for COEUS and COEUS with Common Communities. As the dataset is quite huge, we consider the first 2 million nodes from the stream and process those to form communities.

This dataset basically has links that redirect each web page from one functionality to another. Hence the entire graph represents links of web pages which are related to each other and connected via redirecting through images, links or any other source. The dataset has no ground truth communities and hence we will evaluate the results using a metric called modularity for overlapping communities [18].

¹<https://commoncrawl.org/2015/12/november-2015-crawl-archive-now-available/>

²<https://github.com/bbdavidbb/warcfilter>

5.1.4 YouTube

The YouTube dataset is obtained from the SNAP library [17] which comprises of user groups for sharing videos where new users can join as per interests. An individual ground truth community for this dataset is defined by different user defined groups. This forms a single community. Each connected component is considered separately with respect to each ground truth community. The ground truth communities with less than three nodes are excluded from the dataset.

5.2 COEUS

We experimented the existing COEUS algorithm in [4] using the above three datasets. For this experiment, we have considered three seed nodes initially. The seed nodes are selected randomly from available ground truth communities and we have a window size W which is fixed for the maximum community size. We will be using F1 score to measure the accuracy of the algorithm as we have ground truth communities for the publicly available datasets.

We have three different techniques in order to calculate the F1 score using COEUS. The first one is SIMPLE technique, where we increment the community degree of a node by 1 when we process the input stream. The second technique is called EDGE_QUALITY where we increment the community degree of a node by the community degree of the adjacent node.

The F1 score is calculated by comparing the results obtained by using this technique against the ground truth community, GROUND_TRUTH. The last approach is using a clustering algorithm, DROP_TAIL which automatically determines the size of the communities by eliminating the nodes which exhibit weak ties to the community.

The results of these approach with the Amazon dataset are listed in Table 2 along with results of DBLP dataset in Table 3. In the end we have the results of YouTube dataset in Table 4.

Table 2: Amazon dataset: COEUS

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.76
EDGE_QUALITY	GROUND_TRUTH	0.85
EDGE_QUALITY	DROP_TAIL	0.80

Table 3: DBLP dataset: COEUS

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.38
EDGE_QUALITY	GROUND_TRUTH	0.43
EDGE_QUALITY	DROP_TAIL	0.25

Table 4: YouTube dataset: COEUS

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.69
EDGE_QUALITY	GROUND_TRUTH	0.60
EDGE_QUALITY	DROP_TAIL	0.59

5.3 COEUS using Common Communities

This section discusses the results obtained after experimenting the Amazon, DBLP and YouTube datasets using the COEUS with Common Communities algorithm. This algorithm operates with less execution time as compared to the COEUS and processes the stream of input edges in a fashion similar to COEUS. Table 5 lists the results of Amazon dataset. The results of DBLP dataset are listed in table 6. The results of YouTube dataset are listed in table 7.

Table 5: Amazon dataset: COEUS with Common Communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.75
EDGE_QUALITY	GROUND_TRUTH	0.86
EDGE_QUALITY	DROP_TAIL	0.82

Table 6: DBLP dataset: COEUS with Common Communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.40
EDGE_QUALITY	GROUND_TRUTH	0.52
EDGE_QUALITY	DROP_TAIL	0.30

Table 7: YouTube dataset: COEUS with Common Communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.71
EDGE_QUALITY	GROUND_TRUTH	0.70
EDGE_QUALITY	DROP_TAIL	0.50

5.4 Using seed set communities with offline community detection with COEUS

This section discusses the results obtained after experimenting the Amazon, DBLP and YouTube datasets using the seed set communities for COEUS with common communities.

Table 8: Amazon dataset: COEUS with initial seed set communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.71
EDGE_QUALITY	GROUND_TRUTH	0.80
EDGE_QUALITY	DROP_TAIL	0.84

Table 9: DBLP dataset: COEUS with initial seed set communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.33
EDGE_QUALITY	GROUND_TRUTH	0.55
EDGE_QUALITY	DROP_TAIL	0.24

Table 10: YouTube dataset: COEUS with initial seed set communities

Parameter 1	Parameter 2	F1 Score
SIMPLE	GROUND_TRUTH	0.69
EDGE_QUALITY	GROUND_TRUTH	0.68
EDGE_QUALITY	DROP_TAIL	0.55

Table 8 lists results from the Amazon dataset. We can see that the results for test cases (SIMPLE, GROUND_TRUTH) and (EDGE_QUALITY, GROUND_TRUTH) are less accurate as compared to results of COEUS with common communities. But the test case (EDGE_QUALITY, DROP_TAIL) has score more than that of COEUS with common communities.

Table 9 lists results from the DBLP dataset. We can see that the results for test case (EDGE_QUALITY, GROUND_TRUTH) outperforms the accuracy from COEUS with common communities. In the remaining two test cases, COEUS with common communities performs better as compared.

Table 10 lists results from the YouTube dataset. We can see that the results for the test case (EDGE_QUALITY, DROP_TAIL) are better as compared to results of COEUS with common communities. But the test case (EDGE_QUALITY, GROUND_TRUTH) and (SIMPLE, GROUND_TRUTH) has score almost comparable to that of COEUS with common communities.

5.5 Web Crawl dataset results

This section discusses the results obtained using Web Crawl dataset and the variations with respect to the algorithms COEUS, proposed COEUS with Common Communities and offline community detection.

The ground truth communities are not available for this dataset and hence we evaluate it based on the overlapping modularity score for community detection [7]. The modularity score for COEUS using web crawl dataset turns out to be 0.64. Figure 11 shows the overall community structure obtained by using COEUS on web crawl dataset. All the visualization for communities in graph is done using Gephi ³.

The modularity score for COEUS using Common Communities with web crawl dataset is 0.71. Figure 12 shows the overall community structure obtained by COEUS using Common Communities on web crawl dataset.

³<https://gephi.org/>

Figure 11: Community graph for web crawl dataset using COEUS

Figure 12: Community graph for web crawl dataset using COEUS with Common Communities

The modularity score for offline community detection using seed set communities with web crawl dataset is 0.69. Figure 13 shows the overall community structure obtained by offline community detection using seed set communities on web crawl

dataset.

Figure 13: Community graph for web crawl dataset using offline community detection with seed set communities

5.6 Comparing experimental results

In this section, we will compare the results obtained from COEUS, COEUS using Common Communities and the offline community detection technique. On comparing Table 2 and 5 for Amazon dataset, we infer that the F1 scores for test case 1 (SIMPLE, GROUND_TRUTH) are almost similar with a negligible difference of 0.005. For test case 2 (EDGE_QUALITY, GROUND_TRUTH), COEUS with Common Communities performs better as compared to the COEUS. For test case 3 (EDGE_QUALITY, DROP_TAIL), COEUS with Common Communities again performs better as compared to COEUS. The offline community detection technique shows the best results for DROP_TAIL technique in comparison to COEUS and COEUS with common communities. This is due to the fact that offline community detection focuses on building communities from an initial seed set stage of communities rather than ran-

dom seed nodes. In case of SIMPLE, GROUND_TRUTH and EDGE_QUALITY, the accuracy is less than the other approaches.

For DBLP dataset, on comparing Table 3 and 6 for test case (SIMPLE, GROUND_TRUTH) we see that COEUS with Common Communities performs better as compared to the COEUS. For test case (EDGE_QUALITY, GROUND_TRUTH), the results are similar using both the approaches. As community participation score is relative to the common communities, the ranking of nodes sometimes is similar so the score turns out to be similar. For the final test case (EDGE_QUALITY, DROP_TAIL), COEUS performs better as compared to COEUS with Common Communities. For offline community detection approach, EDGE_QUALITY and GROUND_TRUTH test case has better accuracy as compared to both approaches. It shows similar accuracy for other two approaches with different test cases.

For YouTube dataset, on comparing Table 4 and 7 for test case (EDGE_QUALITY, DROP_TAIL) we see that the results for COEUS are better as compared to COEUS with common communities. For test case (EDGE_QUALITY, GROUND_TRUTH), COEUS with Common Communities outperforms COEUS with a good score. Again for test case (SIMPLE, GROUND_TRUTH), COEUS with Common Communities has a better outcome as compared to COEUS. The offline community detection approach shows average accuracy for this dataset as compared to both COEUS and COEUS with community detection. The accuracy scores on an average lie in between the scores for COEUS and COEUS for common communities. The common community algorithm performs well in this case as it's calculation is based on intersection as well as union scores for both the nodes in an edge.

For Web Crawl dataset, we will compare the overlapping modularity score for COEUS, COEUS with common communities and offline community detection using seed set communities. On comparing the scores, we conclude that COEUS with Common Communities outperforms the COEUS and offline community detection approaches with a score of 0.71. The offline community detection has a better accuracy of 0.70 as compared to COEUS which has 0.64. The performance of common community score algorithm is high due to the fact that we consider participation of each node from incoming edge to check if any one of the nodes is a part of the community.

CHAPTER 6

Conclusion and Future Work

For this project, we proposed a Community Detection using Seed Set Expansion with Common Communities algorithm for community detection which is a modification of COEUS [4] which is based on expansion using seed set of communities. The proposed algorithm uses a new approach for knowing how well connected the nodes in a community are. This helps in evaluating alternative approaches to community participation which is used in COEUS.

Table 11 shows that the proposed algorithm gives better results in most of the cases as compared to the original COEUS. We observe that our proposed algorithm does better than COEUS in EDGE_QUALITY and DROP_TAIL techniques. For SIMPLE and GROUND_TRUTH scenario, the score of proposed algorithm is almost comparable to COEUS. We can try to improve the score by using additional metrics instead community participation where we can consider sum of both the community participation and common communities.

Table 11: Comparison between original COEUS and proposed COEUS with common communities for Amazon dataset

Test Case	F1 score for COEUS	F1 score for COEUS with common community
SIMPLE, GROUND_TRUTH	0.76	0.75
EDGE_QUALITY, GROUND_TRUTH	0.85	0.86
EDGE_QUALITY, DROP_TAIL	0.80	0.82

Table 12 shows that the proposed algorithm with common communities outperforms COEUS with great scores with respect to all the test cases and scenarios.

Table 12: Comparison between original COEUS and proposed COEUS with common communities for DBLP dataset

Test Case	F1 score for COEUS	F1 score for COEUS with common community
SIMPLE, GROUND_TRUTH	0.38	0.40
EDGE_QUALITY, GROUND_TRUTH	0.43	0.52
EDGE_QUALITY, DROP_TAIL	0.25	0.30

Table 13 shows that the proposed algorithm with common communities performs better than COEUS in both EDGE_QUALITY and SIMPLE scenarios whereas for EDGE_QUALITY and GROUND_TRUTH, the performance is less than expected.

Table 13: Comparison between original COEUS and proposed COEUS with common communities for YouTube dataset

Test Case	F1 score for COEUS	F1 score for COEUS with common community
SIMPLE, GROUND_TRUTH	0.69	0.71
EDGE_QUALITY, GROUND_TRUTH	0.60	0.70
EDGE_QUALITY, DROP_TAIL	0.59	0.50

With respect to proposed COEUS algorithm in this project, we can try to improve the accuracy for the same by focusing on SIMPLE and GROUND_TRUTH scenarios where the algorithm fails to performs better than the COEUS. We can use different or combination of metrics like common communities with community participation in order to ensure increased accuracy.

As the ratio of intersection and union considered by common community score will be higher for edges where both the nodes are present in community, the possibility of getting better accuracy increases. As ground truth communities can be different for each dataset depending upon the features of classification, we can try experimenting the approach with parameters like modularity for overlapping communities for increasing the accuracy. Designing an algorithm not dependent on ground truth will help in better classification and accuracy.

LIST OF REFERENCES

- [1] V. da Fonseca Vieira, C. R. Xavier, and A. G. Evsukoff, “A comparative study of overlapping community detection methods from the perspective of the structural properties,” *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.
- [2] R. Andersen, F. Chung, and K. Lang, “Using PageRank to Locally Partition a Graph,” *Internet Mathematics*, vol. 4, no. 1, pp. 35 – 64, 2007. [Online]. Available: <https://doi.org/>
- [3] A. Pagourtzis, D. Souliou, P. Potikas, and K. Potika, “Overlapping community detection via minimum spanning tree computations,” in *2020 IEEE Sixth International Conference on Big Data Computing Service and Applications (Big-DataService)*, 2020, pp. 62–65.
- [4] P. Liakos, K. Papakonstantinou, A. Ntoulas, and A. Delis, “Rapid detection of local communities in graph streams,” *IEEE Transactions on Knowledge and Data Engineering*, 07 2020.
- [5] G. Cormode and S. Muthukrishnan, “An improved data stream summary: The count-min sketch and its applications,” *J. Algorithms*, vol. 55, no. 1, p. 58–75, Apr. 2005. [Online]. Available: <https://doi.org/10.1016/j.jalgor.2003.12.001>
- [6] L. Lu and M. Zhang, *Edge Betweenness Centrality*. New York, NY: Springer New York, 2013, pp. 647–648. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_874
- [7] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri, “Extending the definition of modularity to directed graphs with overlapping communities,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2009, no. 03, p. P03024, mar 2009. [Online]. Available: <https://doi.org/10.1088/1742-5468/2009/03/p03024>
- [8] S. Gregory, *Finding Overlapping Communities Using Disjoint Community Detection Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 47–61. [Online]. Available: https://doi.org/10.1007/978-3-642-01206-8_5
- [9] J. Yang and J. Leskovec, “Community-affiliation graph model for overlapping network community detection,” in *2012 IEEE 12th International Conference on Data Mining*, 2012, pp. 1170–1175.
- [10] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004. [Online]. Available: <https://link.aps.org/doi/10.1103/PhysRevE.69.026113>

- [11] M. Seifikar, S. Farzi, and M. Barati, “C-blondel: An efficient louvain-based dynamic community detection algorithm,” *IEEE Transactions on Computational Social Systems*, vol. 7, no. 2, pp. 308–318, 2020.
- [12] P. De Meo, E. Ferrara, G. Fiumara, and A. Provetti, “Generalized louvain method for community detection in large networks,” in *2011 11th International Conference on Intelligent Systems Design and Applications*, 2011, pp. 88–93.
- [13] S. Fortunato and M. Barthélemy, “Resolution limit in community detection,” *Proceedings of the National Academy of Sciences*, vol. 104, no. 1, pp. 36–41, 2007. [Online]. Available: <https://www.pnas.org/content/104/1/36>
- [14] S. Ghosh, M. Halappanavar, A. Tumeo, A. Kalyanaraman, H. Lu, D. Chavarrià-Miranda, A. Khan, and A. Gebremedhin, “Distributed louvain algorithm for graph community detection,” in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2018, pp. 885–895.
- [15] M. Murali, K. Potika, and C. Pollett, “Online local communities with motifs,” in *2020 Second International Conference on Transdisciplinary AI (TransAI)*, 2020, pp. 59–66.
- [16] L. S. Ahn YY., Bagrow J., “Link communities reveal multiscale complexity in networks,” *Nature*, vol. 761-764, June 2010.
- [17] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” <http://snap.stanford.edu/data>, June 2014.
- [18] M. Elyasi, M. Meybodi, A. Rezvanian, and M. A. Haeri, “A fast algorithm for overlapping community detection,” in *2016 Eighth International Conference on Information and Knowledge Technology (IKT)*, 2016, pp. 221–226.