Efficient Community Detection for Large Scale Networks via Sub-sampling

Venkata Pavan Kumar Bellam

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Computer Engineering

Srijan Sengupta, Co-chair
Jia-Bin Huang, Co-chair
Amos Lynn Abbott

December 13, 2017
Blacksburg, Virginia

Keywords: Spectral clustering, Extreme points, Sub-sampling, PABM

Efficient Community Detection for Large Scale Networks via Sub-sampling

Venkata Pavan Kumar Bellam

(ABSTRACT)

Many real-world systems can be represented as network-graphs. Some of the networks have an inherent community structure based on interactions. The problem of identifying this grouping structure given a graph is termed as community detection problem which has certain existing algorithms. This thesis contributes by providing specific improvements to various community detection algorithms such as spectral clustering and extreme point algorithm. One of the main contributions is proposing a new sub-sampling method to make existing spectral clustering method scalable by reducing the computational complexity. Also, we have implemented extreme points algorithm for a general multiple communities detection case along with a sub-sampling based version to reduce the computational complexity. We have also developed spectral clustering algorithm for popularity-adjusted block model (PABM) model based graphs to make the algorithm exact thus improving its accuracy.

Efficient Community Detection for Large Scale Networks via Sub-sampling

Venkata Pavan Kumar Bellam

(GENERAL AUDIENCE ABSTRACT)

We live in an increasingly interconnected world, where agents constantly interact with each other. This general agent-interaction framework describes many important systems, such as social interpersonal systems, protein interaction systems, trade and financial systems, power grids, and the World Wide Web, to name a few. By denoting agents as nodes and their interconnections as links, any such system can be represented as a network. Such networks or graphs provide a powerful and universal representation for analyzing a wide variety of systems spanning a remarkable range of scientific disciplines. Networks act as conduits for many kinds of transmissions. For instance, they are influential in the dissemination of ideas, adoption of technologies, helping find jobs and spread of diseases. Thus networks play a critical role both in providing information and helping make decisions making them a crucial part of the Data and Decisions Destination Area. A well-known feature of many networks is community structure. Nodes in a network are often found to belong to groups or communities that exhibit similar behavior. The identification of this community structure, called community detection, is an important problem with many critical applications. For example, communities in a protein interaction network often correspond to functional groups. This thesis focuses on cutting-edge methods for community detection in networks. The main approach is efficient community detection via sub-sampling. This is applied to two different approaches. The first approach is optimization of a modularity function using a low-rank approximation for multiple communities. The second approach is a spectral clustering where we aim to formulate an algorithm for community detection by exploiting the eigenvectors of the network adjacency matrix.

I would like to dedicate this to my parents and my sister.

Acknowledgments

First, I would like to express my sincere gratefulness and gratitude to Dr. Srijan Sengupta, who has been my advisor for the thesis. He has not just been an advisor, he was also a guide and a mentor during my Masters degree at Virginia Tech. He has been of tremendous support throughout the period. He always had time to listen to my ideas and has always been supportive, encouraging new ideas and maintaining good energy. He has always been flexible of things and providing freedom to work in the most comfortable manner possible. This thesis would have never been possible to even start with without him. I would once again like to express my sincere thankfulness to him.

I would also like to thank Prof. Jia-Bin Huang who is a co-chair and Prof. Lynn Abbott who is a member of my committee. They have been of great help throughout the thesis. They have always been supportive and encouraging making it easier for me to work on my thesis.

I would like to extend my gratitude to Prof. Madhav, Prof. Anil and all the people at NDSSL for being so supportive and also for kindly offering me lab space. They have been very kind and helpful throughout and the environment in the lab helped me do my work in a much better manner.

I am also thankful to all the people in Prof. Srijan's NRG research group. They have been working on some really exciting stuff making it inspiring and motivating to be a part of the group. I would like to thank every one of them for all the suggestions throughout. It has been fun spending time with all of you.

I would like to thank all of my friends who have been of great support to me throughout the thesis and also during my entire Masters program. They have helped me out of all the emotionally difficult times during the thesis and have stood as support for me to fall back in hard times. It wouldn't have been this smooth or enjoyable without them. I loved and cherish all the amazing moments I spent during my Masters degree and for this I would again like to thank every one of my friends who has been there.

Finally, I would like to thank my parents and my sister who have been the most important people in my life. They have been of constant support throughout and if not for their love and care, I wouldn't have been the person I am today. They are my heroes and my constant inspiration and I cannot thank them enough in words for how much they mean to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Networks and Graphs

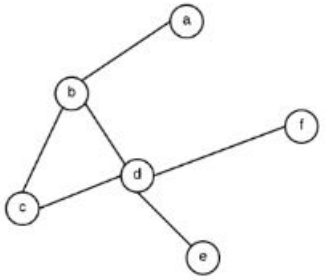
A Network is a group of nodes that interact with each other. Any system with distinct entities and interaction between these entities can be represented as a network. Complex systems can be modeled as networks where the nodes represent the entities in the network while links represent the interaction between them. Many of the real world systems can be represented as networks. Different systems such as transportation systems, biological systems, communication networks, social networks, world wide web, a network of author-citations power grids are few real-world systems that can have network representations. Graphical representation of a complex system allows mathematical analysis in the network making it easier to understand systems and find interesting properties.

Network representation of a system allows easy application of analytical methods like computing centrality of nodes, betweenness among nodes, the diameter of the network, degree distribution of network, density of interactions and several others on real-world networks.

Different type of network representations can exist for real-world systems. A scenario where all the nodes may not be of the same type leads to a class of networks called heterogeneous networks. Other class of networks could be where the links connect more than two nodes. These edges are called hyperedges and the resultant networks are called hypergraph networks. Another case could be networks where the interaction between nodes is directed which means only one direction of interaction holds true in contrast to bi-directional interaction which forms a new category called directed graph networks. Also, these interactions could be weighted instead of being just a boolean present/absent interaction which makes the networks weighted graph networks. In this thesis, we only consider the class of networks with the same type of nodes, no weight on interactions and the interactions to be undirected making the networks unweighted, undirected and homogeneous networks.

1.2 Mathematical Representation

A graph G corresponding to a network is represented as $G = (N, E)$ where N is the set of all nodes in the network while E is the set of all the edges or interactions between nodes in the network. We consider $n = |N|$ to be the number of nodes in the graph. This is a very simple and abstracted representation of a network but is quite informative to analyze. Since it has a mathematical form, all of the graph theory concepts could be applied for analysis.



Edges	Degree
a,b	a 1
b,d	b 3
d,f	c 2
b,c	d 4
d,e	e 1
d,f	f 1

Figure 1.1: Example of a small graph with six nodes. Degrees of different nodes are listed.

For this thesis, we only consider simple graphs where the nodes are all of the same type and the edges are all simple undirected, unweighted connections between nodes. The Adjacency matrix is a matrix representation of graph such that it encompasses all the information in the graph giving a complete representation. Adjacency matrix A is defined as

$$a_{ij} = 1 \quad if \quad \{i, j\} \in E \tag{1.1}$$

$$= 0 \quad if \quad \{i, j\} \in E \tag{1.2}$$

Here, we have that if a link exists between node i and node j, then a_{ij} becomes 1 else it is 0. We can also assume that since we consider simple graphs with no self-loops, we have $a_{ii} = 0 \quad \forall \quad i \in \{1, 2, ..., n\}$. Also, we only consider case of undirected graphs. So, if $\{i, j\} \in E$ then we have $a_{ij} = a_{ji} = 1$. This makes the adjacency matrix A symmetric with diagonal elements being zero.

In a graph, the degree of a node represents the number of links between that node and all other nodes in the graph. The degree of a node can be easily calculated from adjacency matrix A. Since, degree indicates the sum of all the edges from that node, row sum of that particular node in adjacency matrix gives the degree of that node. So, we have degree of node i given by

$$d_i = \sum_{j=1}^{n} a_{ij} \tag{1.3}$$

1.2.1 Graph models

Graphs are mathematical representations of real-world networks. But we can also generate our graphs based on a model thus creating synthetic graphs which are not based on real-world networks. Generally, graphs generated from real-world networks have degree distributions of nodes following the power law.

Generating a graph can either be backed by a model or can be random [23], [19], [12]. Consider a scenario, where we have n nodes for the graph and without following any distribution we randomly select e number of links among all the possible $\frac{n(n-1)}{2}$ links. We get a graph with n nodes and e edges which has been generated randomly without basis of any model.

One such graph generation method is called $Erdös - Rényi$ random graph which generates a random graph [6]. The way it works is that we initially consider n nodes for the graph. Then we choose a random probability p that lies in [0, 1] such that an edge between each pair of vertices is chosen with probability p where for nodes i and j we have adjacency matrix entry $A_{i,j} \sim bernoulli(p_{i,j})$. This model used to generate graphs is called $G_{n,p}$ where this represents the ensemble of all the graphs generated by this model. A graph from this model with m edges occurs with probability $\binom{M}{m}p^m(1-p)^{M-m}$ where M is total number of possible edges which is equal to $\frac{n(n-1)}{2}$. There also exists another alternate model $G_{n,m}$ where all the graphs having n nodes and m edges are equally likely to occur.

Considering the $G_{n,p}$ model, we have certain interesting properties that occur because of the way model is chosen. We have a property that nodes in the graph have degrees following Poisson distribution and hence this model is also called Poisson random graph. Hence, the average degree of a node in the graph turns out to be mean of Poisson distribution which is approximately pn. There are also other properties like network size, graph density, connectivity which can be estimated because of the model used to generate the graph. There are several other models like generalized random graphs, bipartite graphs and power-law degree based graphs that can be used to generate graphs.

1.3 Communities and Community Detection

Given a graph of n nodes, there exists some inherent community structure within the nodes on the basis of similarity or higher connectivity among the nodes [7], [10], [15]. A community is defined as a group of nodes that exhibit similarity or strong relation among themselves than with other nodes in the graph. It is essentially a set of nodes grouped into a community if they have certain common characteristics compared to other nodes in the graph.

The community detection problem is the problem of assigning community labels to every node [8], [11], [14], [16]. So, given a graph, we need to assign a community label to every node and this information for all the nodes together forms a community assignment. We have that

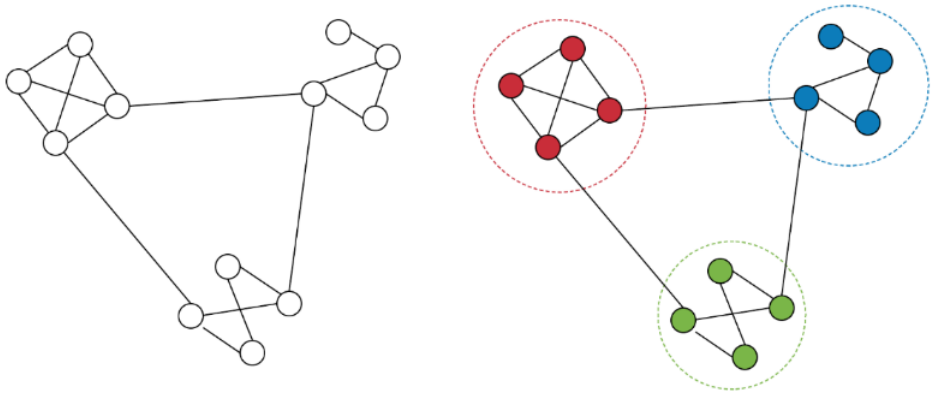


Figure 1.2: Small network showing community information of nodes with same color nodes representing one community.

nodes with same community assignment belong to same community indicating that they are more related or connected to each other than to other nodes outside that community.

This is a form of unsupervised labeling method as there is no classification of nodes into different categories but rather this is a concept of grouping "like" nodes together. So, for a given graph, a complete community assignment C indicates community assigned to every node in the graph. So, for community assignment, there is no supervision as to which nodes need to be alloted to which community. The only inputs on which community assignment depends on is the input graph data and the number of communities that the nodes need to be grouped into.

We also need to have an evaluation method to determine how good the solution of the community detection problem is and what represents the goodness of a solution. We could consider a community assignment to be good as it is close to the ground truth community assignment. However, we cannot directly compare the community assignments because of the label matching problem. So, we perform label matching and label switching before evaluating the community assignment. Once matched, we can define the error metric as the hamming distance between the transformed community assignment and the ground truth community assignment. This measures the number of mismatches in node community assignments thus giving an overall sense of mismatching in a grouping of nodes. As this metric increases, it indicates that the community assignment is far and very different from the ground truth. If this metric is zero, it indicates that all the node community assignments are exactly same as ground truth indicating a perfect zero error scenario.

For the distance or comparison error measure, we could also use other metrics like cosine similarity, Normalized mutual information (NMI), Euclidean distance and several others. In this thesis, we will use hamming distance as a metric.

1.3.1 Label Matching

Given community assignment vector for all the nodes in the graph, it could be possible that few of the community assignments indicate the same grouping of nodes but are represented differently.

Consider a scenario where we have five nodes n_1, n_2, n_3, n_4, n_5 in a graph and we need to find two communities in the graph. Consider one of the community assignment that assigns n_1, n_3, n_5 to community 1 and assigns n_2, n_4 to community 2, which implies that the community assignment vector would be {1, 2, 1, 2, 1}. Consider another community assignment that assigns n_1, n_3, n_5 to community 2 and n_2, n_4 to community 1 which implies that the community assignment vector would be {2, 1, 2, 1, 2}. From the nature of the community assignment, a community label only indicates which community the nodes belong to. Rather, the community assignment only provides information indicating that nodes with same community label belong to the same community. In the above example, both the community assignments mean the same thing indicating that n_1, n_3, n_5 belong to the same community while n_2, n_4 belong to another community. As the actual community labels do not mean anything, the information is only encoded in the fact that same community labeled nodes fall into the same community. Hence, the community assignments shown above are equivalent. This issue is called label switching where the labels might just be interchanged still representing the same membership information for nodes.

In a general community detection problem, it could be possible that two community assignments mean the same thing about node memberships but the community labels could be permuted. So, to solve the problem of label matching, we permute community labels and check to see if the community assignments become same for some community label permutation. If such permutation exists, we just need to choose that permutation and thus consider equivalence of community assignment.

One of the methods to find label matching is by brute force. We compute all possible permutations of community labels and for every permutation of labels, we perform label switching over the community assignment vector. We use a distance or error metric for comparing community assignments and the permutation that results in the least value for the error metric indicates the closest equivalent label matching that can be obtained. If it goes to zero, it indicates that the community assignments have perfectly matched implying that the community assignment vectors are actually same but only representing it in a different manner.

1.3.2 Label Switching

Label switching is a follow-up step for Label matching. When we try to match two community assignments, we try out all the possible label permutations that can reduce the error metric between the community assignments. The resulting label permutation needs to be applied to

the community assignments to remove the difference in representations and bring equivalence while comparing community assignments. This is called label switching where labels are switched without losing any information just to ensure equivalence.

This is very useful while comparing different community assignment vectors. Consider a scenario where we have different community assignments with some common nodes and few nodes which are not common in both of them. Now, if we want to use both the community assignments together, we need to first make community assignments equivalent by matching the common nodes. Now, we perform label switching to ensure that the non-common nodes are also brought to a mutually agreed equivalent representation between both the community assignments. Then the information from both the community assignments can be used together to obtain much more information about the combined set of all the nodes. This is the basis for the stitching process that is outlined in the next sections.

1.3.3 Community based graph models

In the earlier subsection, a random model has been used to generate graphs. We have seen what communities mean and how community detection problem is formulated. There exist graph generation models that do not consider community information to generate a graph. Few models exist which consider community-based information while generating graphs. These are called block models as they consider blocks in the network and ensure that nodes within a block have a special property compared to nodes from other blocks so that nodes from the same block are related and hence would have certain block-wise community structure. Three different community-based graph models existing are:

1. Stochastic Block Model (SBM)
2. Degree Corrected Block Model (DCBM)
3. Popularity Adjusted Block Model (PABM)

Stochastic Block model (SBM)

This is a very basic graph model which considers k blocks in the model where every community is considered to be a block. This model considers that all nodes in a block as stochastically equivalent. Further, the model has k^2 parameters. The parameters of the model are given by ω a k-by-K matrix. This matrix indicates probabilities for edges based on communities the nodes belong to. For an edge between nodes i and j, consider that i belongs to community r and node j belongs to community s then the probability of an edge is given by ω_{rs}. So, the ω matrix gives probabilities for edges based on communities the nodes belong to and then based on this probabilities edges are generated and thus a graph is formed.

Interaction for nodes within same community is given by diagonal elements and since community interaction is symmetric, we have the ω matrix to be symmetric. Consider that we have a n-by-K membership matrix M, where every row in this matrix has a one in particular position indicating which community that particular row corresponding node belongs to.

Now, using this we can write probability matrix for the model as,

$$P = \mathbf{M}\omega\mathbf{M}' \tag{1.4}$$

From this, we can also deduce that the rank of the probability matrix is k as k is the least rank for matrices in the product. This is a very basic model as equivalence is assumed among all the nodes belonging to the same community. Hence, we have the mean degree for nodes in the same community to be same and also the distribution of edges for a node among different communities is also same for nodes belonging to the same community. [24] explains more about spectral clustering in SBM.

Degree Corrected Block Model (DCBM)

This is a slightly advanced model compared to SBM where nodes within the same block are not treated exactly equally but they are given freedom in terms of the degree of the nodes. Even here for k communities, k blocks are considered. Further, the model has k^2 and additionally n more parameters where n is the number of nodes. The parameters of the model are given by ω a k-by-K matrix and $\mathbf{\Theta}$ a n-by-n diagonal matrix. ω matrix indicates probabilities for edges based on communities the nodes belong to and $\mathbf{\Theta}$ matrix indicates the weight for each node to form an edge which is given by the diagonal elements. So, these matrices determine the probability of an edge and then the edge is generated with that probability. For an edge between nodes i and j, consider that i belongs to community r and node j belongs to community s then the probability of edge is given by $\omega_{rs}\mathbf{\Theta_{ii}\Theta jj}$. So, the ω matrix gives probabilities for edges based on communities of their nodes and $\mathbf{\Theta}$ matrix adds a factor to increase or decrease the probability of edge and thus influencing the degree of the node.

Interaction for nodes within same community is given by diagonal elements and since community interaction is symmetric, we have the ω matrix to be symmetric. Moreover, since every node has a factor associated with it, $\mathbf{\Theta}$ matrix will be a diagonal matrix with weights along the diagonal. Consider that we have a n-by-K membership matrix M, where every row in this matrix has a one in particular position indicating which community that particular row corresponding node belongs to.

Now, using this we can write probability matrix for the model as,

$$P = \mathbf{\Theta}\mathbf{M}\omega\mathbf{M}'\mathbf{\Theta} \tag{1.5}$$

From this, we can also deduce that rank of the probability matrix is k as k is the least rank for matrices in the product. This is a slightly advanced model as flexibility is allowed

among nodes belonging to the same community. However, at a higher level, we still have the behavior of same community nodes with other communities only varying by a constant. [28] describes more about Spectral clustering in DCBM model.

Popularity Adjusted Block Model (PABM)

This is an advanced model compared to DCBM where nodes within the same block are not treated equally, but they are given freedom in terms of the degree of the nodes and also in terms of their interaction with other communities. Node popularity of a specific node for a particular community is defined as the number of edges from this particular node to nodes in that specific community. For DCBM, node popularity does not have flexibility and thus PABM is advanced in a way that it lets node have flexible popularities. [25] explains more on PABM model.

For a network with n nodes and K communities, we have nK parameters for the model represented as λ where $\lambda_{ir}, 1 \leq i \leq n, 1 \leq r \leq K$, are the popularity parameters for node i in community r.

For an edge between node i and node j for $i < j$, we have probability of an edge given by

$$p_{ij} = \lambda_{ic_j}\lambda_{jc_i}, \tag{1.6}$$

where $\lambda_{ir}, 1 \leq i \leq n, 1 \leq r \leq K$, are the popularity parameters and $0 \leq p_{ij} \leq 1$ for all $i < j$.

The probability matrix for the model when written in matrix form is given by,

$$P = (\lambda M) \circ (\lambda M')' \tag{1.7}$$

where $A \circ B$ represents the Hadamard product of A and B and M be the n-by-K membership matrix.

Note that λM is n-by-n with rank K, and for Hadamard products, $rank(A \circ B) \leq rank(A) \times rank(B)$. Thus, we have that probability matrix has a rank less than or equal to K^2.

More details and analysis of these models are dealt with in later sections.

1.4 Clustering

As discussed in earlier sections, community detection problem is finding the best or most appropriate community assignment for nodes in the graph. But since this is an unsupervised learning problem, where we do not have a notion to measure if the community detection result is good, we need some function based on which we perform optimization.

In cases with known ground truth community assignments, we can use hamming distance as metric after performing label matching and label switching. However, we cannot have ground

truth available for evaluation in case of unsupervised learning. So, we come up with different optimization functions to evaluate and find the best community assignment. Depending on the optimization function, the method for community detection varies. Clustering in an exact sense is a grouping of nodes into different clusters. The method we use to group these varies with the evaluation function or the error metric we choose for clustering.

1.4.1 General Clustering

General clustering methods exist where the entire data is taken as such without any dimensional reduction and a distance-based grouping is done. One such method is k-means clustering.

In k-means clustering, the optimization function we use is the mean squared distance. This ensures that all the points to the extent possible are assigned to the closest cluster center. And for this particular mean squared optimization, a gradient descent based iterative algorithm exists which runs iteratively improving the assignment each time and finally converges to a cluster assignment that optimizes the mean squared distance function.

Other types of clustering algorithms like Hierarchical clustering exist which work differently compared to k-means algorithm [27]. Hierarchical clustering has two different approaches. One is called agglomerative clustering where initially lots of tiny clusters are formed based on some distance measure and later clusters are merged into one if they are within a threshold distance. Different threshold values result in a different number of clusters and finally, we choose a cutoff threshold to get the required number of clusters. The other approach is a divisive approach where initially we consider the entire set of points and then slowly break the group into pieces depending on the distance metric cutoff. As the threshold changes, a varied number of clusters can be obtained. Finally, we choose a cutoff threshold to obtain the final number of clusters.

Another clustering method called Expectation Maximization based on soft assignments exists. In this, we try to fit Gaussian models where the number of models would be equal to the number of communities required. This is a soft clustering method where probabilities are assigned to every node indicating the probability of it being in a cluster.

1.4.2 Spectral Clustering

This is also a distance-based clustering method but it deals with data in a lower dimensional Eigenspace. In this method, we consider the adjacency matrix representation of the graph. Eigendecomposition is done over this matrix to find Eigenvectors. Since we are trying to cluster into k communities, we only consider the top k Eigenvectors as they would contain significant information. The rows of this Eigen matrix represent the nodes in Eigenspace. These k dimensional subspace representations of the nodes are then used

to perform a distance-based clustering like k-means in the Eigenspace. Based on this mean squared distance optimization function in Eigenspace, communities are assigned to the nodes in the graph.

An expensive step in this method is Eigen decomposition which lets us project nodes into Eigenspace and hence making it computationally better for performing k-means. However, this only tackles mean squared distance based optimization in Eigenspace and would not give best community assignment with respect to some other optimization function. [24] explains more on Spectral Clustering.

1.4.3 Generalized Optimization

All of the existing clustering algorithms work in cases where the optimization function is fixed and do not work for any general optimization function. There exists a brute force method that can be applied in case of a general optimization function where all the possible community assignments are exhaustively computed and plugged in the optimization function to find the best community assignment. But this method is computationally very expensive.

There are a few optimization functions that do not have any existing algorithms designed but the optimization functions are quite critical giving out useful community assignments and parameters for mode fitting. Few of them are modularity functions for specific block models discussed above. Further details of this are mentioned in Appendix C. SBM, DCBM and PABM block models have a modularity function associated with them which tells the goodness of the fit for that particular model and also evaluates the goodness of community assignment. [21], [20] explain more about the ideas of extreme points.

Convex Optimization-Extreme points

A specific set of general optimization functions that satisfy certain conditions become eligible for extreme point based optimization algorithm [3]. This extreme point algorithm for community detection circumvents the problem of spanning all community assignments in a brute force manner. Further cases of two community and multiple community extreme point algorithm are discussed in further sections. This algorithm works on the principle of convex optimization where the only boundary of feasible region achieves optimum and not the interior points. Rather, the extreme points of the feasible region would contain the optimal solution for the network. But this is only applicable for optimization functions which are linear in the space where extreme points are computed [3].

To start with, we perform Eigen decomposition and compute the Eigenspace into which we want to project. Now, we compute the exhaustive set of all the community assignments possible for the graph and project them into the lower dimensional space. All the projected points in this space form the set of feasible points. We compute the extreme points of the

convex hull of these points where only these extreme points are the candidate community assignments for optimizing the generalized optimization function. An incremental method also exists for extreme point computation. Extreme point algorithm is applicable to two community detection case. A slightly different algorithm is designed for multiple community case and also its incremental version is discussed in the further chapters.

Chapter 2

Spectral Clustering via sub-sampling

2.1 Motivation

Spectral Clustering is a popular and well-studied community detection method based on Eigen decomposition of the Graph Laplacian. The top k Eigenvectors of the graph are considered and the point representations in this space are used as data for a spatial clustering technique to find community assignments for all the nodes in the network. The computationally expensive steps in this method are Eigen decomposition and the Laplacian matrix computation. As the size of the network grows, the number of nodes in the graph increases resulting in an increase in computation time. We propose a method based on network sub-sampling to make the spectral clustering method computationally efficient for large networks.

2.2 Spectral Clustering

Consider a network containing a set of nodes $\{N\}$ and a set of edges $\{E\}$ where every edge is an undirected link between two nodes in the network. Let us consider a mathematical representation of the network in the form of adjacency matrix A. Considering that the network has n nodes which is equal to the cardinality of the set of nodes $\{N\}$, we will have the adjacency matrix to be of size n-by-n. An element in i^{th} row and j^{th} column indicates if there exists an edge from i^{th} node and j^{th} node. An undirected graph has all the edges between nodes to be undirected and therefore in the adjacency matrix, we have the same value for an element in i^{th} row, j^{th} column and element in j^{th} row, i^{th} column. This makes the adjacency matrix A symmetric.

All the traditional clustering algorithms including spectral clustering assume that along with the network graph information we also have the number of communities/clusters as input. To estimate this, typically a heuristic or elbow method based on some distance metric.

However, for our problem, we assume that we know the number of clusters we need to find in the network.

2.2.1 Algorithm

Spectral Clustering for a graph

Input: Adjacency matrix A, number of clusters k *Output:* cluster assignment $\{C\}$.

1. Diagonal matrix D with node degrees is constructed using adjacency matrix A.
2. Compute Laplacian L of the adjacency matrix.
3. Eigen decomposition of the Laplacian L is done resulting in Eigen matrix E.
4. Considering the highest k Eigen vectors, we obtain matrix E' from E.
5. Row normalization is performed over rows of matrix E'.
6. Rows in E' are k dimensional representations of network nodes. K-means clustering is performed to group the network nodes into k communities. Every node is assigned to a community denoted by C_i for node i. Thus we have the output community assignment $\{C\}$.

We have graph adjacency matrix A and the number of communities k as input. Initially, instead of using adjacency matrix, we use the Laplacian form of the adjacency matrix to perform spectral clustering. In cases of sparse matrices, we could also add small perturbation [5] to the matrix to ensure smoother and faster computation of Eigenvectors and also to improve accuracy by accounting for randomness in the adjacency graph. For this, we compute the degree diagonal matrix D where every element in diagonal represents the degree of the node which is the sum of a row in adjacency matrix A. Then, we compute Laplacian L as, $L = D^{-1/2}AD^{-1/2}$.

Now, we apply Eigen decomposition on the graph Laplacian L to get the Eigen matrix E where we have $LE = E\Lambda$ with the columns of E indicating the Eigenvectors. Now, we use this Eigen information to project nodes into a lower dimensional space. For the k community detection problem, we consider only the top k Eigenvectors from Eigen matrix E which form a matrix E'. The rows in this matrix E' would be of size k indicating points in k dimensional space. These n rows are projections of network nodes into k dimensional Eigenspace.

Now, we have a representation of all the network nodes in a lower dimensional Eigenspace. We perform a normalization step to make all the rows unit norm so that they have their L_2

norm equal to 1. Every normalized row is a normalized unit vector representation of network node in Eigenspace. In this representation, points belonging to the same cluster/community lie close to each other. So, we perform a simple distance-based (Euclidean) k-means clustering method which converges to a final assignment where every node would be assigned to a certain cluster. Thus, we have an assignment where all the nodes are assigned cluster ids indicating the cluster they belong to. This results in a community detection problem solution where we have that all the nodes having the same cluster id belong to the same cluster.

2.3 Problem Statement

Many of the real-world systems can be represented as network-graphs. In cases like social networks with a very large number of users or in cases where we have a network of all the products in a warehouse which is a tremendously huge graph, we have large graphs or networks to deal with. So, handling large data is a case that needs to be considered and we need to come up with some solution to apply this Spectral Clustering algorithm in an efficient manner for large networks.

So, this new proposed algorithm needs to have two main objectives. One objective would be that it should be able to handle large-scale networks, which in case of slightly smaller networks could be reflected as improved performance. The other objective would be to maintain the accuracy as close to the accuracy obtained by the actual spectral clustering method. This is critical as an improved performance with the cost paid in terms of accuracy is not acceptable. So, accuracy needs to be within a certain limit from actual Spectral clustering algorithm.

2.4 Design

For large scale networks, some of the steps in spectral clustering method get expensive, making the algorithm slow. We analyze different steps in Spectral clustering algorithm to find out the bottlenecks so that performance can be improved. The first module of computing Laplacian is slightly expensive as it is dependent on network size. Other module involving Eigen Decomposition also depends on the size of the network, hence is the critical step taking large time in case of huge networks. The last module involving k-means clustering doesn't scale a lot with network size as the number of dimensions is fixed based on the number of communities. Increase in the number of points does not cost a lot in this module. So, we have the main bottleneck as the Eigen decomposition step that depends on network size making the algorithm very slow for large-scale networks.

So, the main idea behind this new algorithm is reducing the network size. We need to somehow come up with a smaller network to work with. This idea directly leads to a point

where we consider a sub-network of the entire graph. But this introduces many things to be handled such as the number of sub-graph to be considered, how do we relate all of these sub-graphs to get back to the original graph. So, the entire algorithm is split into two parts. The first part is sampling and the second part is stitching.

The first part of the algorithm deals with dividing the whole network into subgraphs and dealing with each one of them separately. The second part of the algorithm is about how we stitch results computed for each of these subgraphs to get the result for the whole graph.

2.5 Parameters

This algorithm is based on sampling subgraphs from the whole network. The algorithm has few parameters that could be varied to make algorithm give different results.

The following are the parameters that this algorithm depends on:

1. Size of subgraph (m) : Proportion of nodes in the subgraph.
2. Number of subgraphs (s) : Number of splits of the original graph.
3. Overlap size (o) : Proportion of nodes in the overlap region.
4. Sampling technique (Ω) : Selection strategy for overlap and subgraph nodes.

The algorithm depends on how we sample the network. So, the sampling technique Ω that is used becomes a critical parameter that could be varied. There also exists some overlap between different subgraphs so that certain coherence exists between subgraphs which aids the stitching step to collectively compute the result for the entire graph. The sampling technique parameter Ω decides how the overlap happens, the number and kind of nodes present in the overlap. A Further effect of varying sampling technique is discussed later.

The next important parameter becomes the relative size of subgraph m. This dictates the amount of computational gain we obtain. This essentially decides the size of subgraph and the size of subgraph directly influences the computational complexity of clustering on subgraph. So, this is an important parameter for the algorithm.

Another parameter for the algorithm would be the number of subgraphs s. This dictates the amount of computational gain we get. The number of subgraphs influences the size of the subgraph. This critically influences stitching step.

Overlap size o is another critical parameter. This ensures that different subgraphs have certain nodes in common which help for maintaining coherence between subgraphs so that this common information can be used to stitch these subgraph results to get results for the whole graph.

These parameters depend on one another. Since all of these parameters are relative to the total number of nodes n, we can have that

$$o + sm = 1 \tag{2.1}$$

This is supported by the argument that the total number of nodes is nothing but a combination of overlap nodes and the combined set of all nodes distributed among different subgraphs. As we have s subgraphs and every subgraph has m relative nodes in it, we have sm nodes other than overlap nodes. Finally, we have relative number of overlap nodes o which when added to the rest of nodes will result in a complete set of all nodes.

We further discuss in detail about each of the parameters and provide an overview of the effect of choosing each parameter on the algorithm.

2.5.1 Number of subgraphs (s)

Number of subgraphs s denotes the number of subgraphs that the entire graph is split into. Every subgraph has a certain number of overlap nodes, the rest on nodes are split among all of these subgraphs into a disjoint set of nodes.

In relation with other parameters, We have number of subgraphs s as

$$s = \frac{1 - o}{m} \tag{2.2}$$

This parameter decides the number of subgraphs we divide the graph into in the first step of the algorithm. It also effects on the stitching step of the algorithm as it depends on the number of subgraphs to be stitched.

It can take integer values in the range one to the difference of one and overlap nodes thus having a range between 1 to $1 - o$.

The case where number of subgraphs s is equal to one is equivalent to the scenario where no sub-sampling happens and all the nodes are considered as a single subgraph.

As the number of subgraphs increases, smaller and smaller subgraphs are considered. The following are the effects on both accuracy and time complexity of the algorithm as the number of subgraphs increases:

1. Size of subgraph decreases and hence the computational time of the algorithm decreases in the subgraph clustering step.

2. Size of subgraph decreases and hence accuracy of clustering on this subgraph decreases in the subgraph clustering step.

3. Stitching needs to be done over more number of graphs and hence computational time increases in the stitching step.

4. Voting would be done on more number of subgraphs and hence accuracy could improve in stitching step.

These effects are proved experimentally in the results section.

2.5.2 Subgraph size (m)

Subgraph size m denotes the relative number of nodes in each subgraph leaving out the overlap nodes. These denote the distinct nodes in a subgraph which differ between different subgraphs. It is directly affected by choice of the number of subgraphs. This is a number relative total number of nodes in the graph. If we have M as actual nodes in the subgraph other than overlap nodes and if the graph has a total number of n nodes, then we have

$$m = \frac{M}{n} \tag{2.3}$$

Also in relation to other parameters, We have subgraph size as

$$m = \frac{1 - o}{s} \tag{2.4}$$

This parameter influences the total size of subgraph as it is equal to the number of nodes in subgraph leaving out the overlap nodes. Hence, this affects the subgraph clustering step. This parameter also affects the number of subgraphs in the algorithm thus influencing the stitching step.

This parameter can take values in the range from one to the total number of nodes leaving out overlap nodes thus has a range between $\frac{1}{n}$ to $1 - o$.

The case where m takes a value of $\frac{1}{n}$ indicates that only one extra node belongs to subgraph other than overlap nodes. The other extreme case where m takes a value of $1-o$ indicates that the subgraph contains all the nodes other than overlap thus resulting in only one subgraph.

As the subgraph size m increases, the total size of subgraph keeps increasing. The following are the effects of increasing the subgraph size parameter m:

1. Total size of subgraph increases and hence the computational time of the algorithm increases in the subgraph clustering step.

2. Total size of subgraph increases and hence accuracy of clustering on this subgraph increases.

3. Stitching needs to be done over a lesser number of graphs and hence computational time decreases in the stitching step.

4. Voting would be done on a lesser number of subgraphs and hence accuracy doesn't improve in stitching step.

These effects are experimentally shown in the results section.

2.5.3 Overlap size (o)

Overlap size parameter o denotes the relative number of overlap nodes in each subgraph. These denote the set of nodes that remain the same in all the subgraphs. It determines the number of nodes that would be left to be distributed among all the subgraphs. If we have O as actual number of overlap nodes and if graph has a total number of n nodes, then we have

$$o = \frac{O}{n} \tag{2.5}$$

Thus o overlap size is a relative parameter indicating the proportion of overlap nodes. Also in relation to other parameters, We have overlap size as

$$o = 1 - ms \tag{2.6}$$

This parameter influences the total size of subgraph as it is equal to the number of overlap nodes in every subgraph. Hence, this has an effect affects the subgraph clustering step. This parameter also affects the size of overlap in the subgraph thus influencing the stitching step.

This parameter can take values in the range from zero to the total number of nodes thus has a range between 0 to 1.

The case where o takes a value of 0 indicates that no overlap nodes exist. This makes the algorithm non-functional. So, we have a lower limit for the value of parameter o. From the stitching algorithm, we have that all the communities need to be represented in the overlap so that label matching part of the stitching algorithm works. For that, we consider a mathematical bound as proven in appendix A. It says that given n as the number of nodes and k as the number of communities, we have that the minimum value of o needs to be greater than certain bound. Thus when choosing overlap size parameter o, we need to start from this bound value.

The other extreme case where o takes a value of 1 indicates that the subgraph contains all the nodes of the graph and only one subgraph is obtained.

As the overlap size o increases, the total size of subgraph keeps increasing. The following are the effects of increasing o the overlap size parameter:

1. Total size of subgraph increases and hence the computational time of the algorithm increases in the subgraph clustering step.

2. Total size of subgraph increases and hence accuracy of clustering on this subgraph increases.

3. Stitching needs to be done over more number of overlap nodes and hence computational time increases in the stitching step.

4. Label matching would be done on more number of overlap nodes and hence accuracy increases in stitching step.

These effects are experimentally shown in the results section.

2.5.4 Sampling technique

Sampling technique is a parameter which decides the way sampling is done and subgraphs are generated. In the subgraph clustering step, we first sub-sample the graph into subgraphs, then perform spectral clustering on them. Again, during stitching step we use the overlap nodes which are common to all the subgraphs to stitch results from different subgraphs together. So, sampling method plays a vital role on how to choose overlap nodes and how to partition into subgraphs via sub-sampling.

We could choose any of the existing sampling methods. It could be a specific method where a certain type of nodes or nodes with certain special characteristics are given importance. However, we need to ensure that the overlap nodes contain representative nodes from all communities. Also, it should be approximately true that the rest of subgraph needs to be a good representative of communities in the graph.

Here, we choose a random sampling technique where nodes are chosen randomly from the whole graph. This ensures fairness in the node selection and since the sampling is random, the subsample would be a good representative of all the communities in the whole graph.

As the sampling method gets closer to being perfectly random, the fairness in sub-sampling keeps increasing. The following are the effects of choosing a random sampling technique:

1. The nodes in overlap are good representatives of all the communities in the graph and hence will lead to perfect label matching and hence accuracy in stitching step.

2. Since the remaining nodes distributed among the subgraphs are perfectly random, every subgraph forms a good representative of the entire graph and hence increases accuracy in subgraph clustering step.

2.6 Algorithm

The algorithm deals with spectral clustering but on smaller scale networks thus improving performance. The algorithm works in two steps where the first step is to split the whole graph into subgraphs by sub-sampling. In this step, a normal spectral clustering algorithm is applied over the subgraphs. The next step would be Stitching where the overlap nodes in different subgraphs are used to stitch the cluster assignments from different subgraphs together to form a single cluster assignment for the entire graph. Finally, we would have a complete community assignment for all the nodes of the original graph.

The following are the two steps in the algorithm:

1. Clustering on subgraph
2. Stitching

2.6.1 Clustering on subgraphs

To begin with, we consider the entire graph which consists of n nodes. We select certain parameter value for overlap size o such that it is greater than the bound. To choose these overlap nodes, we use random sampling technique place them in all subgraphs. We consider the remaining nodes which would be the total number of nodes reduced by overlap size number of nodes. We select a certain value for the number of subgraphs parameter s indicating the number of subgraphs we plan on forming. With a number of subgraphs and overlap size chosen, we can determine the resulting subgraph size. Further, from the set of remaining nodes, random sampling is done and set of subgraph size nodes is placed in every subgraph.

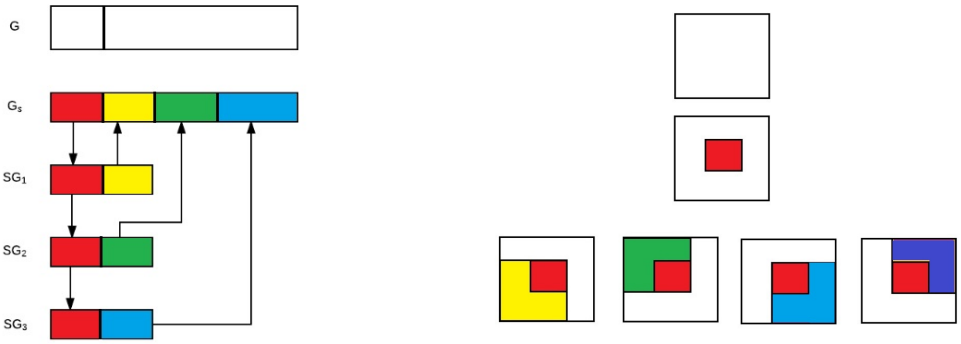
With this, we have the entire set of subgraphs where there exists a certain number of overlap nodes common in all the subgraphs and then the remaining nodes are divided into disjoint sets and distributed to form several subgraphs.

Now, spectral clustering is applied on each of this subgraphs. Eigendecomposition is performed over these smaller graphs and using the representations of nodes in Eigenvector space, k-means clustering is performed. We obtain the best cluster assignment for every subgraph as a result of performing spectral clustering.

Now, these discrete cluster assignments are stitched together as part of stitching step to form the complete community assignment.

2.6.2 Stitching

The second step of the algorithm is stitching where we stitch the cluster assignments from different subgraphs to form a single complete cluster assignment. For stitching, we use the



(a) Node based representation of subgraph splitting.

(b) Graph based representation of subgraph splitting.

Figure 2.1: Figure illustrates how subgraphs are formed via sub-sampling. G represents the whole graph while SG_1, SG_2, SG_3 and SG_4 represent the subgraphs with red color indicating the overlap region.

coherence between different subgraphs established by overlap nodes. These overlap nodes act as a bridge to compare information between subgraphs, relate them and stitch their assignments.

Consider two of the subgraphs as a pair. As the overlap nodes are common in both of them, both need to have the same community assignments for the overlap nodes. However, even same nodes can have a label matching problem as discussed above. So, we perform label matching and form a correspondence between cluster assignments of two subgraphs. We use this for the remaining nodes in the second subgraph and perform a label switching for second subgraph nodes. Once the label switching is done, we will have coherent cluster assignments for both subgraphs. Since the non-overlap nodes are distinct for different subgraphs, we directly use the switched results for them to be considered as the final cluster assignment. In case of overlap nodes, we consider their cluster assignments from all the subgraphs and perform a voting procedure where the majority community into which the overlapping node is voted by the subgraphs becomes the final community assignment for that overlap node. With this, we end up with final community assignment for the entire graph.

2.6.3 Implementation

The exact algorithmic steps are as follows:

Spectral Clustering via sub-sampling

Input: Adjacency matrix A, number of clusters k *Output:* cluster assignment $\{C\}$.

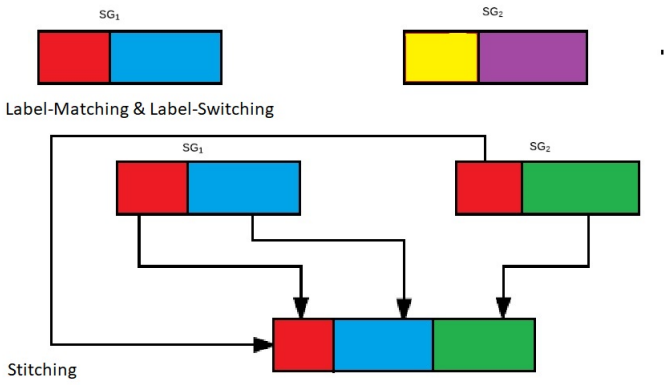


Figure 2.2: Figure illustrates how stitching of subgraphs occurs.SG_1 and SG_2 are subgraphs while after stitching we get community assignment for the entire graph as shown in the last step.

1. Certain values for parameters number of subgraphs s and overlap size o is chosen and random sampling technique is considered.
2. Random node set of overlap size is chosen and assigned to every subgraph. The remaining nodes are distributed into subgraphs as disjoint set of nodes.
3. Spectral clustering is performed at subgraph level.
4. Label matching is performed on overlap nodes of different subgraphs and accordingly label switching is done for subgraph community assignments.
5. Voting is done for choosing community assignment for overlap nodes. Rest of the nodes get community assignment directly from the label switched community assignment of subgraph.
6. We finally have a cluster assignment $\{C\}$ for the entire graph.

2.7 Computational Complexity

The computational complexity is determined to compare and estimate the run-time for the actual clustering and spectral clustering via sub-sampling. The detailed analysis for this is given in B.1.

The computational complexity for the case of the actual spectral clustering algorithm is given by $\mathcal{O}(N^3)$.

The computational complexity for the case of spectral clustering via sub-sampling is expressed as two parts.

For the subgraph part, we have the complexity given by

$$\mathcal{O}(\frac{(1+o(s-1))^3N^3}{s^2})$$

For the stitching part, we have the complexity given by

$$\mathcal{O}(soNk^k)$$

This indicates that for actual spectral clustering, we have the complexity increasing drastically with an increase in the number of nodes. And for the sub-sampling based spectral clustering, we can see that computational complexity is quite low compared to the actual method where a gain of order N is obtained thus giving huge computational gain.

Also within the sub-sampling based method, we can see that with an increase in the number of subgraphs (s), the computational complexity of subgraph step decreases hugely while for a very large value of s, the computational complexity of stitching step increases. On the whole, we have that as the number of subgraphs (s) increases overall computational complexity decreases but if s becomes large, computational complexity starts increasing.

On the same lines, we can see that with an increase in overlap size (o), the computational complexity of subgraph step increases as the subgraph size increases. Also, the complexity of stitching step increases. Hence, with an increase in the overlap size (o), computational complexity increases.

2.8 Experiments and results

In the earlier sections, we have only given a sense of the effect of parameters on the algorithm and also the betterment of the sub-sampling based algorithm compared to the actual clustering algorithm. To support the argument, we have performed extensive simulations under different conditions and the results for those are shown in this section.

In all of the experiments, we have studied the effect of overlap size o by considering values from the set {0.01, 0.02, 0.03, 0.05, 0.1}. Similarly, for studying the effect of the number of subgraphs s, we consider values from the set {5, 10, 25, 50}. For every combination of o and s from these set of values, we compute the runtime and error for sub-sampling based spectral clustering. And for all the simulations, we have used a simple Intel i5 processor on a simple laptop (CPU based) without GPU.

2.8.1 SBM model generated graphs

Here, we consider graphs generated using SBM model. For the SBM model, we have intra-cluster edge probability to be a random number in the range [0.1, 0.2] and inter-cluster edge probability to be a random number in the range [0, 0.05]. For a given graph, we add a little perturbation which is a very small constant multiplied by the total number of edges just to ensure that sampled subgraphs are not disconnected. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average value of time and error. We also compute the standard deviation in the error value. Plots are shown with error versus run-time keeping number of subgraphs constant and overlap size constant.

Consider a graph with 3600 nodes and three communities. The figure 2.3 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 2.4 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual spectral clustering corresponding point has a high runtime and is shown on the graph towards the extreme right. The broken plot with a break in the x-axis(that indicates runtime) is used to show the difference in scale of runtime for actual spectral clustering and spectral clustering in a case where sub-sampling is done.

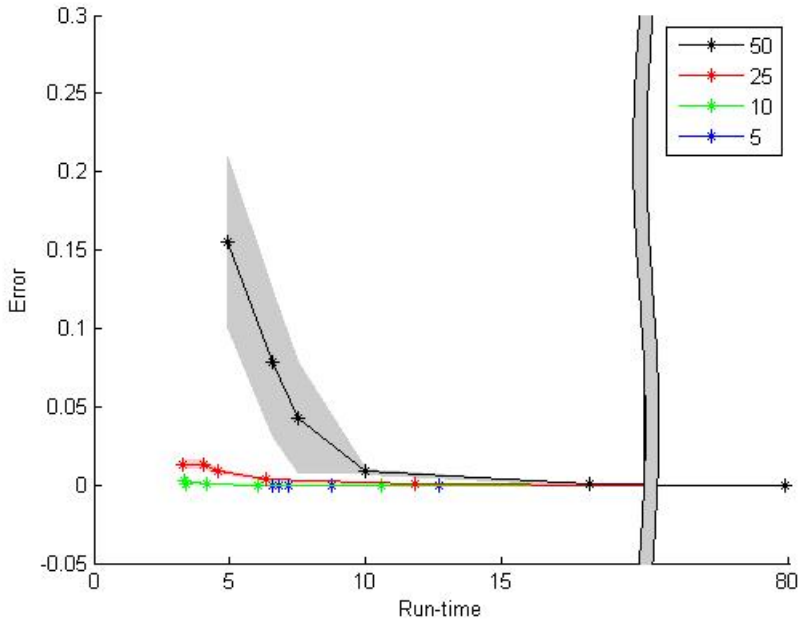


Figure 2.3: Error vs run-time(seconds) plot for SBM model graph with 3600 nodes, three communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

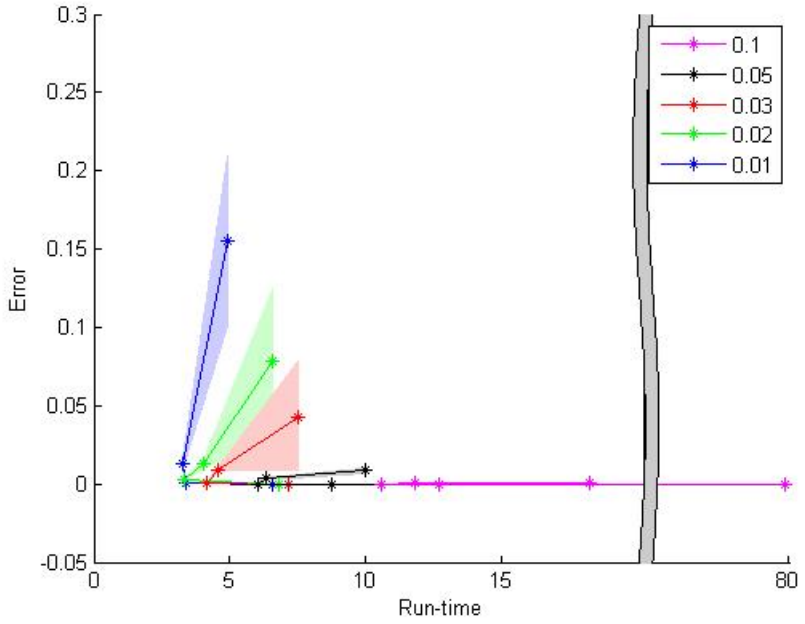


Figure 2.4: Error vs run-time(seconds) plot for SBM model graph with 3600 nodes, three communities having overlap size o constant. Each colored line shows different overlap size.

Consider a graph with 2400 nodes and four communities. The figure 2.5 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 2.6 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual spectral clustering corresponding point has a high runtime and is shown on the graph towards the extreme right. The broken plot with a break in the x-axis(that indicates runtime) is used to show the difference in scale of runtime for actual spectral clustering and spectral clustering in a case where sub-sampling is done.

2.8.2 DCBM model generated graphs

We consider graphs generated using DCBM model. For the DCBM model, we have intra-cluster edge probability to be a random number in the range [0.1, 0.2] and inter-cluster edge probability to be a random number in the range [0, 0.05]. Also, for node degrees we consider random values sampled from a generalized Pareto distribution. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average value of time and error. We also compute the standard deviation in the error value. Plots are shown with error versus runtime keeping number of subgraphs constant and overlap size constant.

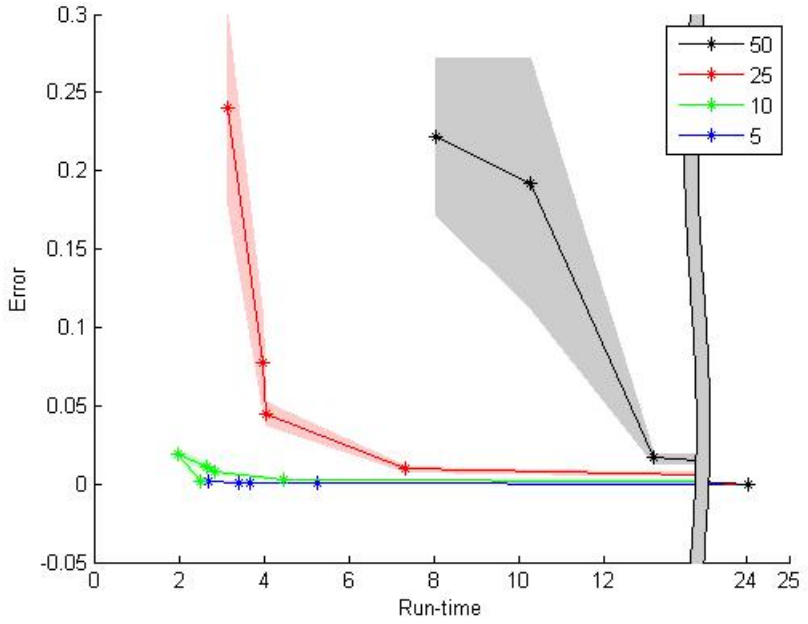


Figure 2.5: Error vs run-time(seconds) plot for SBM model graph with 2400 nodes, four communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

Consider a graph with 3600 nodes and three communities. The figure 2.7 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 2.8 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual spectral clustering corresponding point has a high runtime and is shown on the graph towards the extreme right. The broken plot with a break in the x-axis(that indicates runtime) is used to show the difference in scale of runtime for actual spectral clustering and spectral clustering in a case where sub-sampling is done.

Consider a graph with 2400 nodes and four communities. The figure 2.9 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 2.10 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual spectral clustering corresponding point has a high runtime and is shown on the graph towards the extreme right. The broken plot with a break in the x-axis(that indicates runtime) is used to show the difference in scale of runtime for actual spectral clustering and spectral clustering in a case where sub-sampling is done.

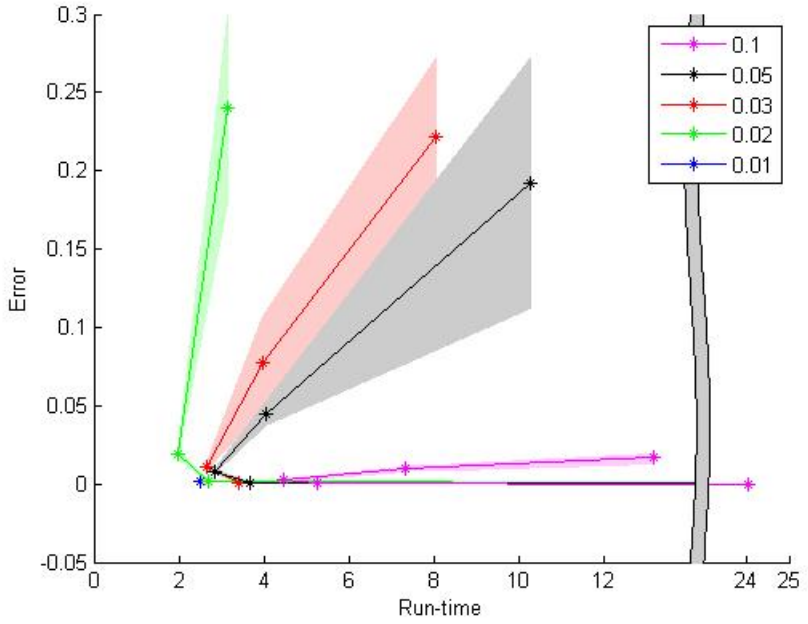


Figure 2.6: Error vs run-time(seconds) plot for SBM model graph with 2400 nodes, four communities having overlap size o constant. Each colored line shows different overlap size.

2.8.3 Real graph

We consider graphs generated from real networks. DBLP (Digital Bibliography & Library Project) is the authoritative website, listing over two million articles. A Connected subset of DBLP data containing authors from four research areas has been extracted [10], [15]. The clustering problem becomes identifying research areas for authors. For this DBLP dataset, we consider all the authors to be nodes resulting in 4057 nodes and the conference participation data as links. We have authors broadly divided into four research areas namely Database, Data Mining, AI and Information Retrieval thus forming four communities. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average value of time and error. We also compute the standard deviation in the error value. Plots are shown with error versus run-time keeping number of subgraphs constant and overlap size constant.

The figure 2.11 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 2.12 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual spectral clustering corresponding point has a high runtime and is shown on the graph towards the extreme right. The broken plot with a break in the x-axis(that indicates runtime) is used to show the difference in scale of runtime for actual spectral clustering and spectral clustering in a case where sub-sampling is done.

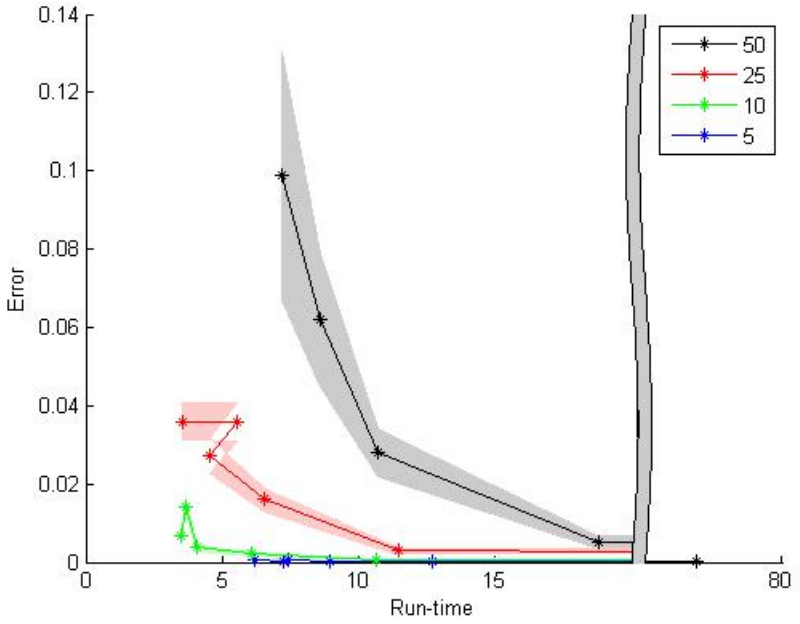


Figure 2.7: Error vs run-time(seconds) plot for DCBM model graph with 3600 nodes, three communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

2.9 Observations

The following are the observations made form the experiments and results above.

Considering the SBM and DCBM model graph data, the plot containing Error versus runtime for a fixed s shows that for a given value of s, the error decreases with increase in o but simultaneously runtime increases with an increase in o. The runtime increase with an increase in o is justified from the complexity expression where the subgraph size increases and hence run time increases. The decrease in error can also be explained by the reason that with an increase in o subgraph size increases making the accuracy in each subgraph higher and also since o increases, more comparisons are possible and the stitching step gives more accurate result by comparing overlap regions.

The plots containing error versus runtime for a fixed o but different s values show that as s increases, error increases and also runtime initially decreases but increases for large s values. The runtime behavior can be justified by the complexity expression where for smaller s values, as s increases, the computational cost of subgraph decreases as the size of subgraph decreases. However, for large s, runtime again increases as the increase in the computational cost of the stitching step overshadows the decrease in computational cost of subgraph step.

Considering the real data example, we still have the same trend where with an increase in

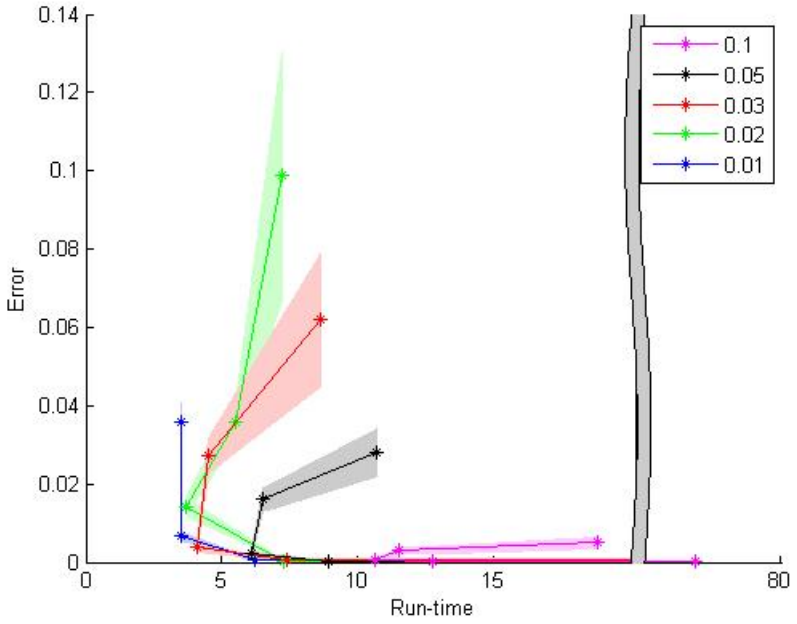


Figure 2.8: Error vs run-time(seconds) plot for DCBM model graph with 3600 nodes, three communities having overlap size o constant. Each colored line shows different overlap size.

o runtime increases while the error variation is not so smooth. In general, error initially decreases with increase in o but later increases. The trend in runtime can be explained by the computational complexity where runtime increases with o while for error, to start with in this case sub-sampled method works better than actual method indicating that sampling is doing good by splitting network and decreasing error. So, error initially decreases with increase in o but later increases with o as smaller graphs seem to perform better and since we are increasing o, error increases to reach to the actual spectral clustering level.

For same o values and an increase in s values, we have runtime initially decreasing but later increasing for huge values of s. This trend can be explained by the computational complexity expression. With an increase of s, the error value is decreasing. Because for this particular case, smaller samples give better results and hence increasing s gives a lesser error.

2.10 Concluding remarks

In conclusion, we can state that sub-sampling based spectral clustering method is useful and performs better compared to the actual spectral clustering method. For any set of parameters in our algorithm, we have that the runtime in the sub-sampling method is far-less than the runtime in case of actual clustering. Also, several choices of parameters ensure that error stays very close to the actual error and in some cases also less than actual error thus giving

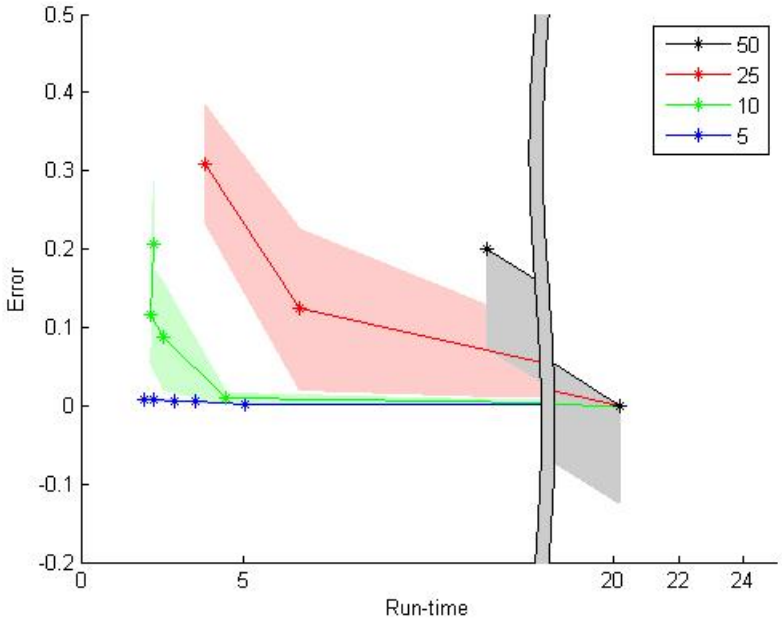


Figure 2.9: Error vs run-time(seconds) plot for DCBM model graph with 2400 nodes, four communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

us a huge gain in computational time but for a very small cost in terms of higher error. The optimal choice of parameters would be a higher value of s but not a very large value thus leaving it around 25. For o, we want to make sure that the o chosen is greater than the bound value and also larger o can give excellent results in terms of deviation from actual error but will lead to a rise in runtime. So, the optimal choice would be a value which is slightly higher than the bound of o.

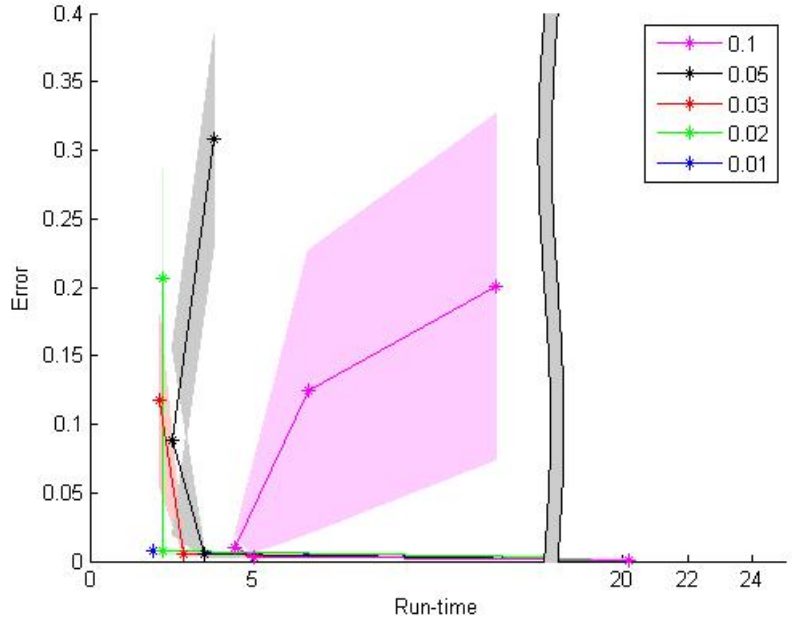


Figure 2.10: Error vs run-time(seconds) plot for DCBM model graph with 2400 nodes, four communities having overlap size o constant. Each colored line shows different overlap size.

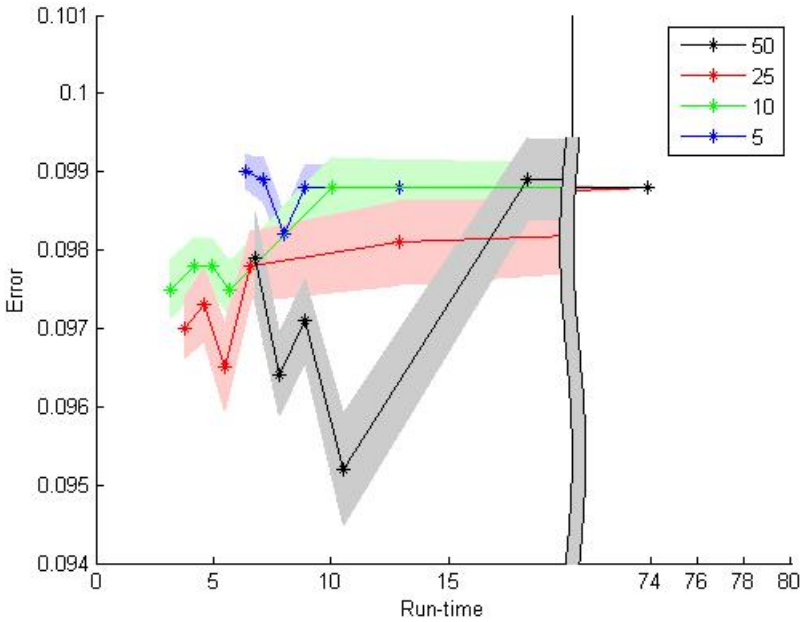


Figure 2.11: Error vs run-time(seconds) plot for DBLP data with 4057 nodes, four communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

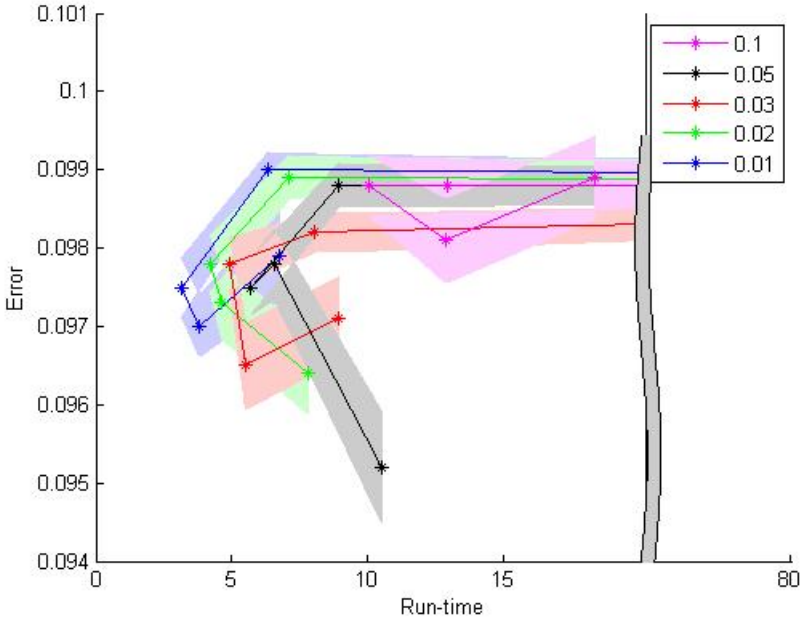


Figure 2.12: Error vs run-time(seconds) plot for DBLP data with 4057 nodes, four communities having overlap size o constant. Each colored line shows different overlap size.

Chapter 3

Extreme point algorithm-Speed and Scale

3.1 Motivation

The conventional two community extreme point algorithm is designed to give a set of candidate community assignments out of which one would be a solution to optimize generalized optimization criteria [21], [20]. This holds true for a broad class of optimization functions among which the popular ones are SBM modularity, DCBM modularity and PABM modularity functions. Details of modularities are discussed in Appendix C. The set of candidate community assignments are obtained by computing extreme points of the convex hull. This convex hull is formed by all the exhaustive community assignments projected into a lower dimensional space. Convex hull extreme point computation is an expensive step in this algorithm [26], [13], [9]. The number of extreme points increases as the number of nodes increase thus increasing the computational complexity of the problem. Here, we propose a method based on sub-sampling and using subgraphs to improve upon the extreme point algorithm by heavily decreasing the computational complexity while still maintaining error within a small bound from actual error.

Consider the working of extreme point algorithm as shown above. The figure above shows a case where there are many points in the projected space but we do not need to consider all the points. Considering the extreme points is enough when looking for the point that is the best for given optimization condition.

Also, the extreme point algorithm does not work for more than three communities problem. So, we modify the two community extreme point algorithm to make it applicable to more than two community problems.

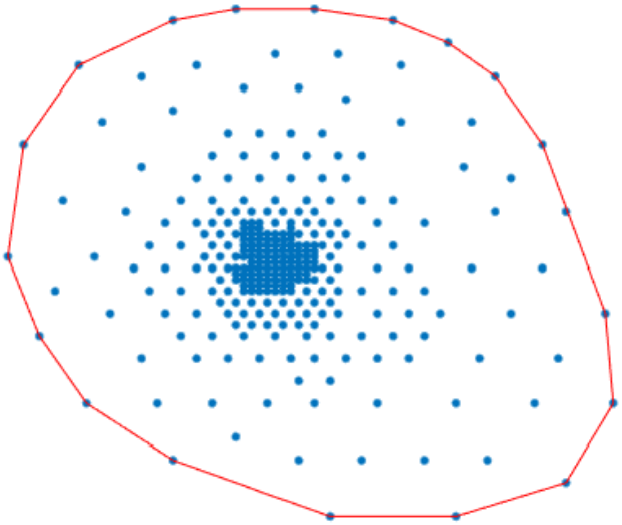


Figure 3.1: Illustration of convex hull in EP algorithm.

3.2 Problem

Extreme point algorithm for community detection deals with optimizing generalized function but it involves projecting all the possible communities into a lower dimensional space and then computing convex hull for the projected community assignments. The extreme points of this convex hull become candidate community assignments for optimizing the general objective function. Here, convex hull computation is an expensive step and it directly scales with the size of the graph. So, we try to speed-up the process by implementing a sub-sampling based extreme point algorithm where we get to deal with smaller subgraphs making the algorithm faster and resulting in computational gains.

Another problem is to make the algorithm work for more than two communities. For multiple community algorithm, the problem is that the algorithm needs to deal with the issue of capturing more than two community information and work in a higher dimensional space to compute appropriate extreme points. Consider an experiment where for a graph with three communities in it, two community based extreme point algorithm is used. We get the three-dimensional convex hull as shown in figure 3.2.

The red dots marked in the convex hull are the extreme points. But the green point shown is the actual correct community assignment. But we see that green point is stuck on an edge in the convex hull. Since we compute the competing candidates by considering extreme points, we will end up not choosing this as a competitive candidate and hence never giving a chance to get the right assignment. Rather, all the extreme points computed in this scenario are assignments where only two communities are used making the number of nodes in the third community to be zero.

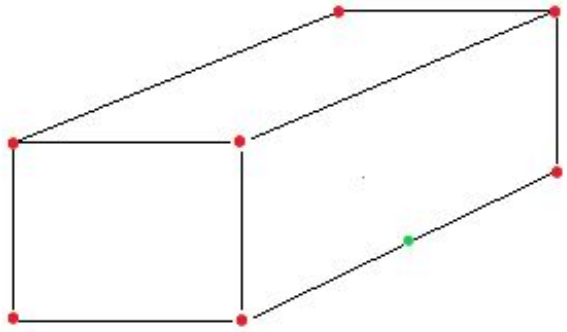


Figure 3.2: Convex hull for a graph with three communities using two community extreme point method.

So, the problem we notice here is that the number of dimensions used is not enough to capture the three community information. So, we need to move a higher dimensional space where much more information can be represented and hence the actual correct assignments get pushed into the corners of the convex hull making them the extreme points. We would ideally want a convex hull as shown in figure 3.3.

We can notice that still some of the red dots indicating assignments with only two communities are extreme points but we also see that the green dots indicating assignments with all the three communities are pushed to the corners of the convex hull thus making them extreme points. Now, we have a good set of candidate assignments as extreme points of which even the correct assignment which has all three communities is likely present. As we end up getting such a good set of extreme points, we can end up finding correct one among these by maximizing modularity.

So, we can reason that a higher dimensional space is needed to capture all the communities information for a multiple community detection problem. We will further discuss the design and choice of such a higher dimensional space.

3.3 Two community extreme point algorithm

The two community extreme point algorithm works trying to find the two community assignment. The algorithm broadly goes through the following steps. Initially, Eigenspace of the graph is determined and all the community assignments are projected into this low dimensional Eigenspace. Then, the convex hull of all these points is considered and the extreme points of the hull form the set of candidate community assignments that can maximize a

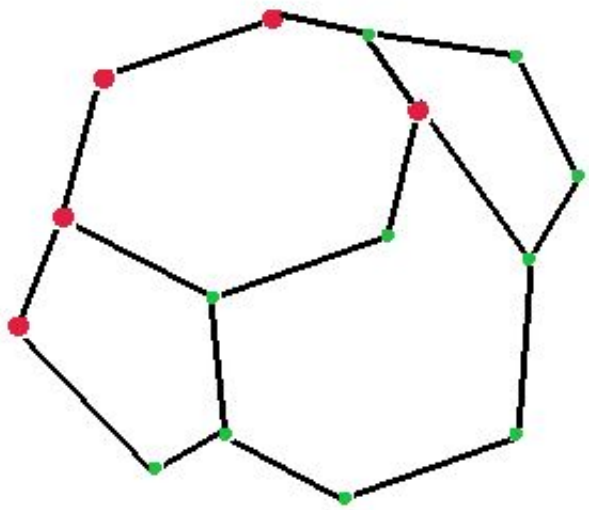


Figure 3.3: Convex hull for a graph with three communities using multiple community extreme point method.

specific class of optimization functions.

3.3.1 Algorithm

The algorithm for the two community extreme point is outlined below:

Exhaustive version of extreme point algorithm for two communities detection

---

Input: Adjacency matrix A, number of clusters $k = 2$ *Output:* Candidate assignments $\{C\}$.

1. Compute Eigen decomposition of the Adjacency matrix A to give Eigen matrix E.
2. Select the highest k Eigen vectors to obtain matrix E'.
3. Construct the set of all possible community assignments C.
4. Project community assignments into k dimensional space represented by rows of E'.
5. Convex hull of the projected community assignments is constructed whose extreme points are set of candidate community assignments C.

Initially, Eigen decomposition of the Adjacency matrix A is done resulting in Eigen matrix E and diagonal Eigenvalue matrix Λ with the decomposition equation as $AE = E\Lambda$ where the columns of E are the Eigenvectors. This is the exhaustive version of the algorithm where all the nodes are considered. From the obtained E Eigen matrix, we only consider the most important k Eigenvectors. We obtain matrix E' from E by taking only k columns corresponding to the highest absolute Eigenvalues. The rows in E' are projections of network nodes in Eigenspace. Further, we consider the set of all possible community assignments C where each assignment has every node assigned to some community. All possible such assignments would be a total of k^n.

Rows in E' are k dimensional representations of network nodes. Now, the exhaustive set of all possible community assignments is projected into the k dimensional space. For a given assignment, a projected point is the sum over all the nodes where if a node belongs to the first community, the corresponding k dimensional Eigenvector of the node is added to the k dimensional vector. We will have the entire exhaustive set of assignments projected into the k dimensional space. A Convex hull is computed in this space over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C.

Here, all the community assignments have been considered together making the algorithm exhaustive. We will look at an iterative version of the extreme point algorithm for two communities.

3.3.2 Iterative Algorithm

The iterative version of two community extreme point algorithm is outlined below:

Incremental version of extreme point algorithm for two communities detection

---

Input: Adjacency matrix A, number of clusters $k = 2$ *Output:* Candidate assignments $\{C\}$.

1. Compute Eigen decomposition over adjacency matrix A resulting in Eigen matrix E.
2. Choose the highest k Eigen vectors to obtain matrix E'.
3. Compute exhaustive set of all possible community assignments for "starting nodes" to obtain C_{st}.
4. Project the set of all possible community assignments for "starting nodes" into the k dimensional space and compute convex hull to obtain C_s.
5. Span all points by incrementally adding one node each time, expanding the community assignments and then recomputing convex hull on them.

6. Stop when all the nodes are added and when we finally have the set of candidate cluster assignments C.

To start with, Eigen decomposition of the Adjacency matrix A is done resulting in Eigen matrix E and diagonal Eigenvalue matrix Λ with the decomposition equation as $AE = E\Lambda$ where the columns of E are Eigenvectors. We obtain matrix E' from E by taking only k columns corresponding to the highest absolute Eigenvalues. The rows in E' are projections of network nodes in Eigenspace.

Consider the set of all possible community assignments C_s for certain set of nodes called "Starting nodes" where each assignment has every node among "starting nodes" assigned to some community. All possible such assignments would be a total of k^{n_s}. Rows in E' are k dimensional representations of network nodes. Now, the set of all possible community assignments for "starting nodes" is projected into the k dimensional space. For a given assignment, the projected point is the sum over all the nodes where if a node belongs to the first community, the corresponding k dimensional Eigenvector of the node is added to the k dimensional vector. We will have the entire set of assignments projected into the k dimensional space. A Convex hull is computed in this space over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C_s.

Now, incrementally an extra node is considered each time. For every extreme point from the set of candidate assignments, we add all possible community assignments for this point and expand the possible community assignments by k times. Extreme points are again computed over convex hull formed by all of these points. This process is repeated by adding one node each time till all nodes are spanned and we finally have the set of candidate cluster assignments C.

This incremental version works only by updating one node each time thus pruning away lots of extreme points and thus giving huge computational efficiency.

3.4 Higher dimensional space

We realized in the earlier section that using a simple k dimensional space to project community assignments, computing convex hull and finding extreme points will not be sufficient and will lead to incorrect results.

We will choose a higher dimensional space such as $k(k-1)$ which we will justify further.

Let's consider the case where we have a graph and we need to detect k communities. Since this is an extension of the two community problem, we would have $k > 2$. Since, this is a k community detection problem, from the spectral nature of the graph, we will have most of

the information encoded in the top k Eigenvectors. So, to start with we would still need to use only the top k Eigenvectors but representing in a higher dimensional space.

From the two community problem, we notice that k dimensions are needed to find community assignment. This can be seen as a zero-one problem where we get the information of whether a node belongs to the first community automatically knowing its presence in the second community. This indicates that we need k dimensions to find if a node belongs to a particular community or not. As we have to determine assignments for k communities, in this case, considering k dimensions to find presence or absence of a node in a community, doing this for all the communities will lead us to consider $k \times k = k^2$ dimensions. But we have a small independence condition which we can use here.

For a given node, its presence/absence information in $k-1$ clusters automatically indicates its presence/absence in the k^{th} cluster. This can be split into the following cases:

1. If a node belongs to any one of the $k-1$ communities, it implies that it is absent in the k^{th} community.
2. If a node doesn't belong to any one of the $k-1$ communities, it implies that it is present in the k^{th} community.

From this, we can deduce that we only need to deal with the problem of determining presence/absence information of a node in $k-1$ independent communities. Hence, we only need $k(k-1)$ dimensions. But here, all these information of presence/absence of a node in $k-1$ communities are related. They cannot be solved independently. The idea is that the following constraint holds for any node.

For any given node, it can only belong to a single cluster.

To hold this constraint, we need to solve for presence/absence problem in a coupled manner rather than a decoupled manner. This means that the constraint holds us from doing absence/presence problem in k dimensions $k-1$ times in a decoupled manner. So, we need to have all of the $k-1$ number of k dimensional spaces together thus forming a huge $k(k-1)$ dimensional space. The illustration of the vector in higher dimensional space is shown below:

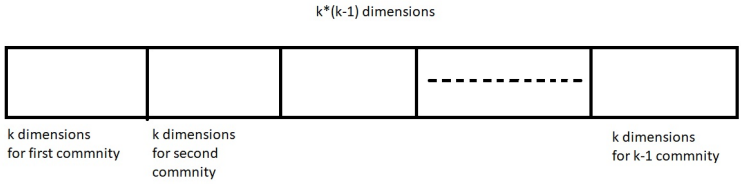


Figure 3.4: Higher dimensional vector split-up and representation.

Now, we have that we need to use $k(k-1)$ dimensional space for multiple communities

detection. We will see the exact algorithm of how multiple communities detection in this higher dimensional space is done.

3.5 Multiple Community extreme point algorithm

The multiple community extreme point algorithm works trying to find the multiple community assignments. The algorithm broadly goes through the following steps. Initially, Eigenspace of the graph is determined and all the community assignments are projected into this lower (higher than two community case) dimensional Eigenspace. Then, the convex hull of all these points is considered and the extreme points of the hull form the set of candidate community assignments that can maximize a specific class of optimization functions.

3.5.1 Algorithm

We have the exhaustive multiple community detection extreme point algorithm as below:

Extreme point algorithm for multiple community detection

Input: Adjacency matrix A, number of clusters k *Output:* Candidate assignments $\{C\}$.

1. Perform Eigen decomposition of the Adjacency matrix A to obtain Eigen matrix E.
2. Select the highest k Eigen vectors to obtain matrix E'.
3. Compute the set of all possible community assignments C which would be a total of k^n.
4. Exhaustive set of all possible community assignments is projected into the $k(k-1)$ nodes.
5. Convex hull is computed in this space over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C.

To start with, Eigen decomposition of the Adjacency matrix A is computed resulting in Eigen matrix E and diagonal Eigenvalue matrix Λ with the decomposition equation as $AE = E\Lambda$ where the columns of E are Eigenvectors. Considering the highest k Eigenvectors, we obtain matrix E' from E by taking only k columns corresponding to the highest absolute Eigenvalues. The rows in E' are projections of network nodes in Eigenspace. Consider the

set of all possible community assignments C where each assignment has every node assigned to some community. All possible such assignments would be a total of k^n.

Rows in E' are k dimensional representations of network nodes. Now, the exhaustive set of all possible community assignments is projected into the $k(k-1)$ nodes. For a given assignment, a projected point is the sum over all the nodes where if a node belongs to any of the $k-1$ communities, the corresponding k dimensional Eigenvector is added to the corresponding place in $k(k-1)$ dimensional vector. We will have the entire exhaustive set of assignments projected into the $k(k-1)$ dimensional space. A Convex hull is computed in this space over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C.

Over this resultant list of cluster assignments, we could apply any modularity function for optimization to find the best cluster assignment for that particular optimization function. This function needs to satisfy certain constraints for the extreme point concept to be applicable.

3.6 Iterative algorithm

The above described exhaustive algorithm for extreme points in multiple community case is quite computationally expensive when performed on all of the nodes at once. The total number of all the exhaustive cluster assignments will be k^n which grows exponentially with the number of nodes n. Hence, this algorithm becomes infeasible to be done on all of the nodes at once.

An incremental approach is proposed to the multiple community extreme point algorithm which doesn't consider all of the nodes at once but rather incrementally considers one extra node every time thus making sure that the computational complexity doesn't shoot up at once but gradually rises.

The basis for this incremental update is that we only start with a very small set of nodes called "starting nodes". We initially only consider exhaustive assignments for these starting nodes. Upon computation of convex hull for these starting nodes, we consider one extra node which can belong to any community thus increasing the number of candidates in higher dimensional space by three. Now, again convex hull is computed over these. And further one extra node is considered. There is a recursion that occurs, where we compute convex hull, considering an extra node every time. Once this is done over all nodes, the algorithm ends.

Incremental extreme point algorithm for multiple communities detection

---

Input: Adjacency matrix A, number of clusters k *Output:* Candidate assignments $\{C\}$.

1. Perform Eigen decomposition of the Adjacency matrix A to get Eigen matrix E.
2. select the highest k Eigen vectors to obtain matrix E'.
3. Compute the set of all possible community assignments C for "starting nodes" to have a total of $k^{n_{st}}$.
4. Project the exhaustive set of all possible community assignments is into the $k(k-1)$ dimensional space.
5. Convex hull is computed over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C_s for "starting nodes".
6. Every node of the graph is considered incrementally by considering for community assignments. Once all nodes are done, we end up with a set of extreme points which is the set of candidate assignments.

To start with, Eigen decomposition of the Adjacency matrix A is performed resulting in Eigen matrix E and diagonal Eigenvalue matrix Λ with the decomposition equation as $AE = E\Lambda$ where the columns of E are Eigenvectors. Considering the highest k Eigenvectors, we obtain matrix E' from E by taking only k columns corresponding to the highest absolute Eigenvalues. The rows in E' are projections of network nodes in Eigenspace.

Consider the set of all possible community assignments C for "starting nodes" where each assignment has every node assigned to some community. All possible assignments would count to a total of k^{n_s}. Rows in E' are k dimensional representations of network nodes. Now, the exhaustive set of all possible community assignments is projected into the $k(k-1)$ dimensional space. For a given assignment, a projected point is the sum over all the nodes where if a node belongs to any of the $k-1$ communities, the corresponding k dimensional Eigenvector is added to the corresponding place in $k(k-1)$ dimensional vector.

We will have the entire exhaustive set of assignments projected into the $k(k-1)$ dimensional space for all of the "starting nodes". A Convex hull is computed in this space over the entire set of points. The extreme points of this convex hull become the resultant candidate assignment set C_s for "starting nodes". Now, every node of the graph is considered incrementally. Each time an extra node is considered, the set of all extreme points adds an extra node. Listing all possible community assignments for this node increases set of candidates by k times. The Convex hull of this set is then computed and repeated for all the nodes in the graph. Finally, we end up with a set of extreme points which is the set of candidate assignments.

Over this resultant list of cluster assignments, we could apply any modularity function for optimization to find the best cluster assignment for that particular optimization function. This function needs to satisfy certain constraints for the extreme point concept to be applicable.

The improvement achieved by the incremental algorithm is based on the idea that if all the nodes are considered together, we would have k^n points on which convex hull would be computed. But rather by only considering "starting nodes" to begin with, computing convex hull at every node addition prunes bad points and doesn't carry them as extreme points further thus reducing the overall count of contestant assignments trying to become extreme points. It is assumed that the points which are inside the convex hull when a certain number of nodes are considered cannot become extreme points later. An early pruning is done over such assignments making the algorithm faster and reducing the computational overhead.

3.7 Two communities as a special case

We have extreme point algorithm for multiple community detection in place where we consider community assignment representations in a higher dimensional space. This boils down to the existing extreme point algorithm for two community detection in a case where $k = 2$.

For $k = 2$ in case of multiple community detection, $k(k-1)$ dimensional space turns out to be two-dimensional space where the rows from the column reduced Eigen matrix can be directly used. Here, the only question would be if a node belongs to a community or not. Hence, we only need k dimensions which would thus reduce down to using two-dimensional space.

Thus algorithm for extreme points in two community detection problem is only a special case of extreme point algorithm for multiple community detection.

3.8 Computational Complexity

The computational complexity is determined to compare and estimate the run-time for the actual clustering and spectral clustering via sub-sampling. The detailed analysis for this is given in B.2.

The computational complexity for the case of actual multiple communities extreme point algorithm is given by

$$\mathcal{O}(N^{k(k-1)})$$

The computational complexity for the case of multiple communities extreme point via sub-sampling is given by

$$\mathcal{O}\left(s\left(\frac{(1+o(s-1))N}{s}\right)^{k(k-1)}\right)$$

.

This slightly changes for two community extreme point algorithm which is explained in B.2. The above complexity expression indicates that for actual spectral clustering, we have the complexity increasing drastically with increase in the number of nodes. And for the sub-sampling based spectral clustering, we can see that computational complexity is quite low compared to the actual method where a gain of order N is obtained thus giving huge computational gain.

Also within the sub-sampling based method, we can see that with an increase in the number of subgraphs (s), we can see that computational complexity of subgraph step decreases hugely. On the same lines, we can see that with an increase in overlap size (o), the computational complexity of subgraph step increases as the subgraph size increases.

3.9 Experiments and results

In the earlier sections, we only have deduced an intuitive sense of the effect of parameters on the algorithm and also the betterment of the sub-sampling based algorithm compared to actual extreme point algorithm. To support the argument, we have performed extensive simulations under different conditions and the results for those are shown in this section. And for all the simulations, we have used a simple Intel i5 processor on a simple laptop (CPU based) without GPU.

3.9.1 Two communities

In all of the experiments, we have studied the effect of overlap size o by considering values from the set $\{0.01, 0.02, 0.03, 0.05, 0.1\}$. Similarly, for studying the effect of the number of subgraphs s, we considered values from the set $\{5, 10, 25, 50\}$. For every combination of o and s from these set of values, we compute the run-time and error for sub-sampling based extreme point algorithm.

SBM model generated graphs

Here, we consider graphs generated using SBM model. For the SBM model, we consider intra-cluster edge probability to be a random number in the range [0.1, 0.2] and for inter-cluster edge probability to be a random number in the range [0, 0.05]. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average case time and error. We also compute the standard deviation in the error value. Plots are shown with error versus run-time keeping number of subgraphs constant and overlap size constant.

Consider a graph with 1000 nodes and two communities. The figure 3.5 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used

and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 3.6 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual extreme point corresponding point has a high runtime and is shown on the graph towards the extreme right but a broken plot with a break in the x-axis(that indicates runtime). This is used to show the difference in scale of runtime for actual extreme point and extreme point algorithm in the case where sub-sampling is done.

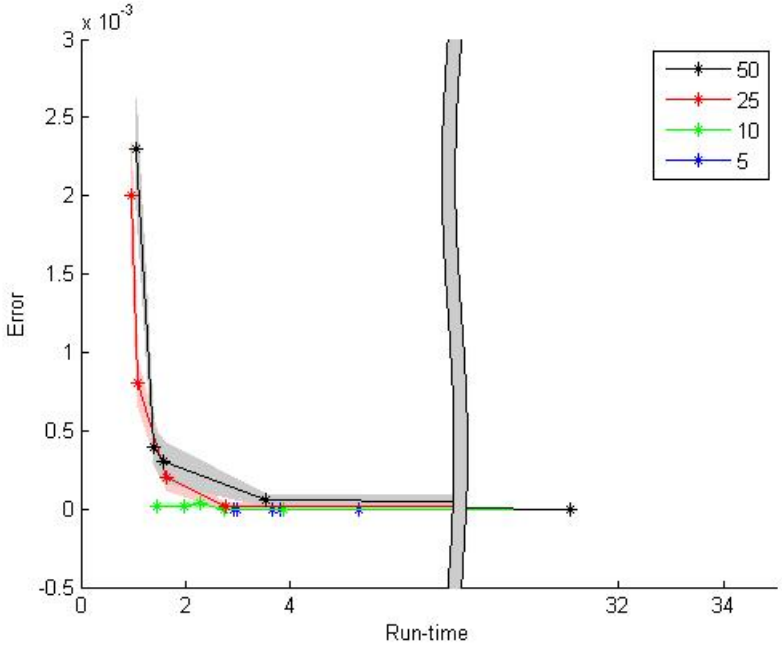


Figure 3.5: Error vs run-time(seconds) plot for SBM model graph with 1000 nodes, 2 communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

DCBM model generated graphs

We consider graphs generated using DCBM model. For the DCBM model, we consider intra-cluster edge probability to be a random number in the range [0.1, 0.2] and for inter-cluster edge probability to be a random number in the range [0, 0.05]. Also, for node degrees we consider random values sampled from a generalized Pareto distribution. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average value of time and error. We also compute the standard deviation in the error value. Plots are shown with error versus run-time keeping number of subgraphs constant and overlap size constant.

Consider a graph with 1000 nodes and two communities. The figure 3.7 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used

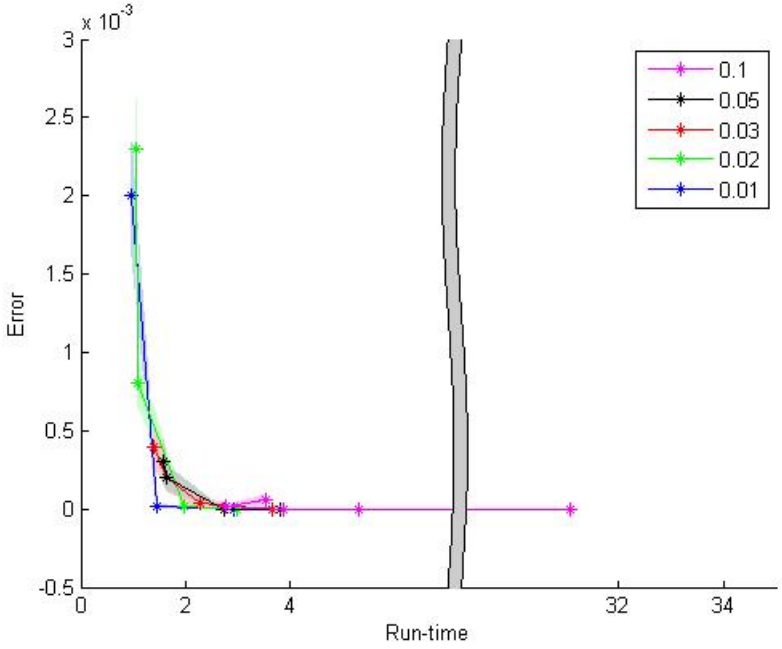


Figure 3.6: Error vs run-time(seconds) plot for SBM model graph with 1000 nodes, 2 communities having overlap size o constant. Each colored line shows different overlap size.

and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 3.8 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual extreme point algorithm corresponding point has a high runtime and is shown on the graph towards the extreme right but a broken plot with a break in the x-axis(that indicates runtime). This is used to show the difference in scale of runtime for actual extreme point algorithm and extreme point algorithm in the case where sub-sampling is done.

Real graph

We consider graphs generated from a real network. The political blogs network has been well studied in the networks literature [1] in general and particularly in connection with the DCBM starting from the original DCBM paper [18], [28], [4], [2]. Following the usual practice, we extract the largest connected component and treat it as a simple graph with 1222 nodes, 16,714 edges and K = 2 communities. For a set of parameters, we run the sub-sampling based algorithm 50 times to find the average value of time and error. We also compute the standard deviation in the error value. Plots are shown with error versus run-time keeping number of subgraphs constant and overlap size constant.

Consider this real political blogs data, the graph has 1222 nodes and two communities. The

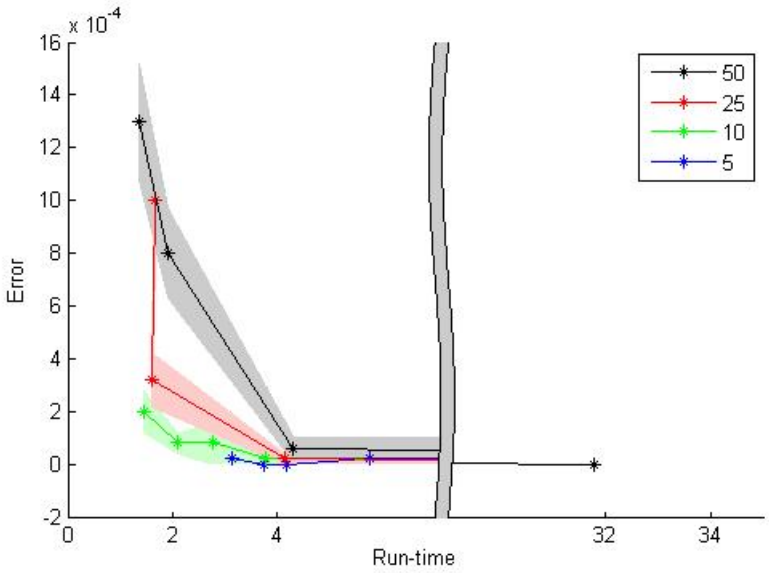


Figure 3.7: Error vs run-time(seconds) plot for DCBM model graph with 1000 nodes, 2 communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

figure 3.9 shows the graph of error versus runtime where for every line we have a fixed s but different s values are used and plotted as different lines. The shaded region shows the possible deviation in error value. Figure 3.10 shows graph of error plotted versus runtime but for a fixed o and also different lines for different o are plotted. In both the plots, the actual extreme point algorithm corresponding point has a high runtime and is shown on the graph towards the extreme right but a broken plot with a break in the x-axis(that indicates runtime). This is used to show the difference in scale of runtime for actual extreme point algorithm and extreme point algorithm in the case where sub-sampling is done.

3.9.2 Multiple communities

Here, we run experiments on small graphs to show the extremely high cost and expensiveness of multiple communities extreme point method.

We consider a small SBM model based graph with 300 nodes and three communities. An incremental algorithm for multiple communities extreme point is run. The plot shows extreme point count against the time taken to run. The figure 3.11 has a blue line indicating a quick rise of extreme points showing both the actual points and the extreme points for every node added while the red line shows the extreme points for every node addition.

Further experiments are run on very small sized graphs to compare multiple community extreme point method with sub-sampling based extreme point method. The results are

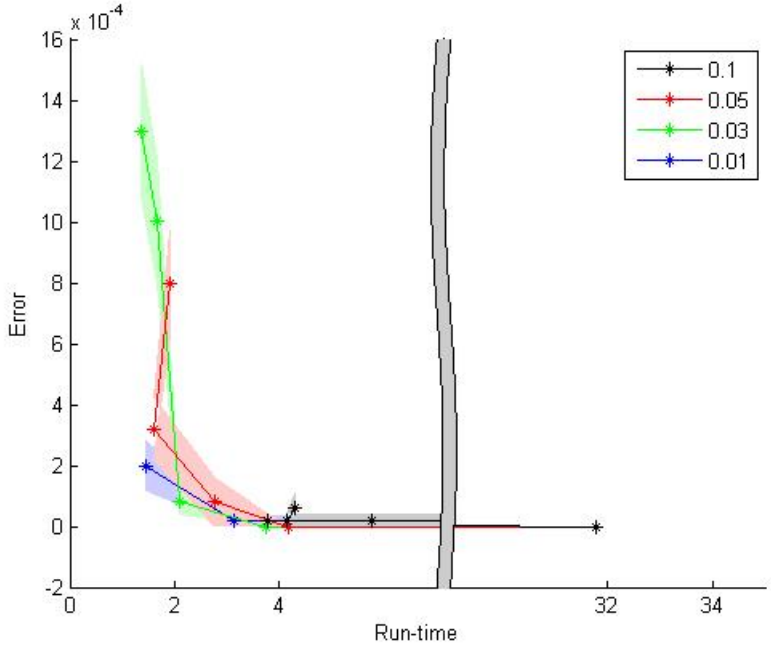


Figure 3.8: Error vs run-time(seconds) plot for DCBM model graph with 1000 nodes, 2 communities having overlap size o constant. Each colored line shows different overlap size.

shown in table 3.1. The results indicate huge improvement in runtime while not much change in error rate. All the graphs used are generated from SBM model for three communities with intra-block interaction probability in the range [0.6, 0.7] and inter block interaction probability lying in [0, 0.05].

Table 3.1: Comparison of multiple community extreme point versus sub-sampled case for three community SBM model graphs.

	Run-Time (sec)		Error	
Nodes	Actual	Sub-sampled	Actual	Sub-sampled
12	62.72	9.72	0.25	0.25
15	136.02	23.19	0.33	0.33
18	Infeasible	99	Infeasible	0.1111

3.10 Observations

From the above experiments and results, we can analyze the effect of parameters and also comment on multiple communities extreme point method.

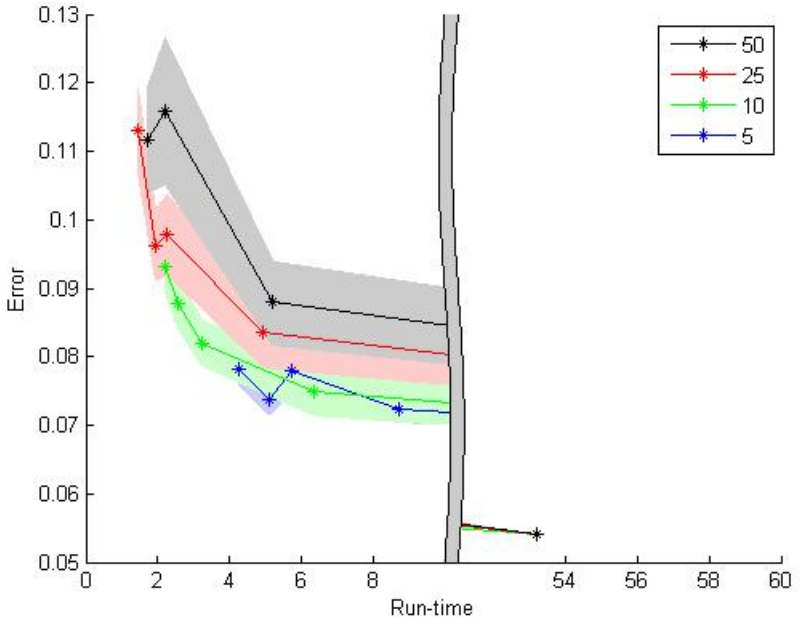


Figure 3.9: Error vs run-time(seconds) plot for Political blogs data with 1222 nodes, two communities having number of subgraphs s constant. Each colored line shows different number of subgraphs.

Considering the SBM and DCBM model graph data and real network graph data, the plot containing Error versus runtime for a fixed s shows that for a given value of s error decreases with increase in o but simultaneously runtime increases with increase in o. The runtime increases with an increase in o is justified from the complexity expression where with an increase in o, the subgraph size increases and hence run time increases. The decrease in error can also be explained by the reason that with an increase in o subgraph size increases making the accuracy in each subgraph higher and also since o increases, more comparisons are possible and the stitching step gives more accurate result by comparing overlap regions.

The plots containing error versus runtime for a fixed o but different s values shows that as s increases, error increases and also runtime decreases. The runtime behavior can be justified by the complexity expression where, as s increases, the computational cost of subgraph decreases as the size of subgraph decreases.

And regarding the multiple communities extreme point method, since this method can now run for multiple community detection problem, we can claim that scalability has been achieved. But for multiple communities, the method takes too much time to run and is nearly infeasible. This is due to the very expensive convex hull step which takes $O(N^6)$ to compute and takes $O(N^5)$ to store the extreme points. Hence, the method becomes infeasible. The graph showing the growth of a number of extreme points with time indicates that the method is infeasible because even for a very small number of nodes, the number of

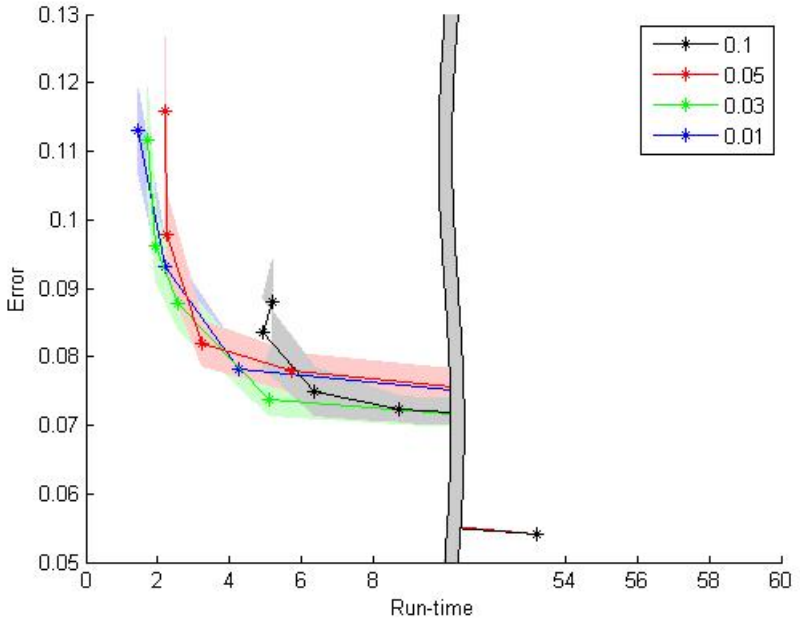


Figure 3.10: Error vs run-time(seconds) plot for Political blogs data with 1222 nodes, two communities having overlap size o constant. Each colored line shows different overlap size.

extreme points and the time taken to compute them increases.

Sub-sampling has been tried in multiple communities extreme point algorithm. But even this is not feasible because even for a small number of nodes, multiple community extreme point method becomes infeasible and hence division should be done into very very small subgraphs. Few examples have been shown in the table where for scenarios in which extreme point algorithm could run, sub-sampling gave better run-time value while error remained same. However, in cases where the multiple communities extreme point is infeasible, sub-sampling based extreme point algorithm could run yielding results.

3.11 Concluding remarks

Thus a more generalized version of the algorithm for extreme point computation for multiple communities is implemented. Two community extreme point algorithm is only a special case of this. This involves using the top k Eigenvectors and a $k(k-1)$ higher dimensional space to capture all the community information and compute extreme points. Also, an incremental version of this algorithm has been proposed where instead of all the nodes being considered together, each node is iteratively taken into consideration thus giving computational gains.

Sub-sampling based extreme point algorithm is proposed where the sub-sampling brings

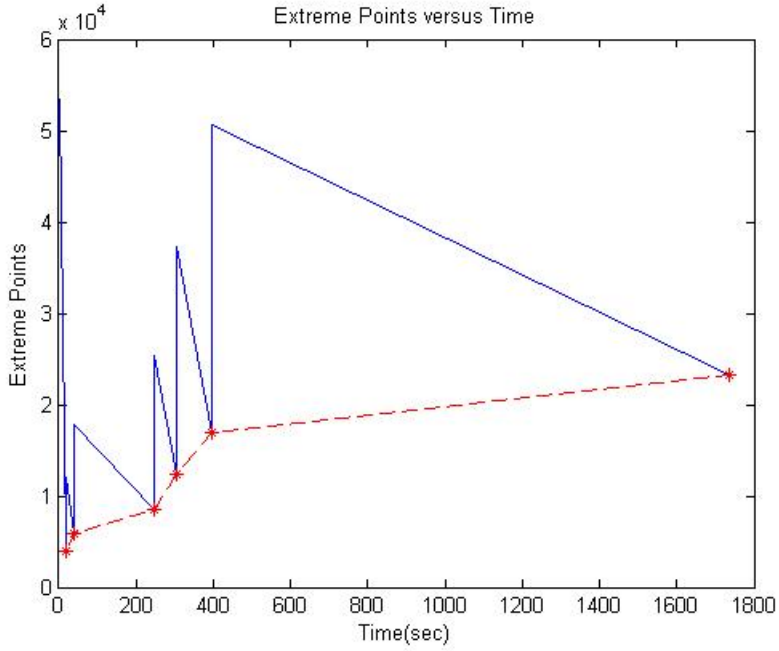


Figure 3.11: Graph showing increase in the number of extreme points versus time taken to compute them.

huge computational gains. This works perfectly well for two communities where the run-time is hugely impacted without much loss in terms of error. It is also applicable for multiple community cases where if the actual method is feasible, sub-sampling brings computational gain and if the actual method is not feasible, then sub-sampling provides a feasible method of running extreme point algorithm for multiple communities.

Chapter 4

Spectral clustering for PABM model

4.1 Motivation

Spectral clustering method exists for SBM model and a more general normalized spectral clustering algorithm exists for DCBM which work very well and provide accurate results. PABM is a more advanced model compared to SBM and DCBM. Most of the real-world networks fit well to a PABM model. But the existing normalized spectral clustering method doesn't yield good results in case of PABM model. So, we try to find a spectral clustering algorithm that works better for PABM model and as a result can perform well on real-world networks too.

4.2 Problem

PABM stands for Popularity Adjusted block model [25]. It is a block structure based model which can be used to generate graphs. In this case, the problem we try to solve is to make spectral clustering algorithm exact (theoretically exact while practically highly accurate) for graphs generated using PABM model. Spectral clustering is an existing algorithm for finding communities in a graph. Applying the existing spectral clustering method gives bad results for PABM model graphs. So, the idea is to add an extra processing step to the existing spectral clustering method and change few things in the algorithm to make it more generalized and highly accurate for PABM model.

The superiority of PABM model is essentially in the fact that node popularities could be modeled which is not possible in SBM and DCBM. For modeling node popularity in networks, we have PABM model where for $i < j$,

$$p_{ij} = \lambda_{ic_j}\lambda_{jc_i}, \tag{4.1}$$

where $\lambda_{ir}, 1 \leq i \leq n, 1 \leq r \leq K$, are the popularity parameters and $0 \leq p_{ij} \leq 1$ for all $i < j$. Here, c_j and c_i indicate the communities to which node j and node i belong to. The interaction between nodes depends on which community the other node belongs to.

So, for a network with n nodes and K communities, we have nK parameters for the model in contrast to k^2 for SBM and $k^2 + N$ for DCBM.

For spectral clustering, we first need to write down the probability matrix for the model. Let M be the n-by-K membership matrix. We have

$$P = (\lambda M) \circ (\lambda M')' \tag{4.2}$$

where $A \circ B$ represents the Hadamard product of A and B. Note that λM is n-by-n with rank K, and for Hadamard products, $rank(A \circ B) \leq rank(A) \times rank(B)$. Hence, we can deduce that the rank of the matrix for PABM model is less than or equal to k^2.

We compute the Laplacian of P to be $\mathcal{L}$.

$$\mathcal{L} = D^{-1/2} P D^{-1/2} \tag{4.3}$$

This Laplacian form is used as a starting point for spectral clustering. But since this is just a matrix multiplication, rank remains the minimum rank among all the product terms. So, the rank of Laplacian can also be considered to be less than or equal to K^2.

Spectral clustering is based on a special property of the spectral structure of Laplacian $\mathcal{L}$ when the random graph model is an SBM or a DCBM. For both models, the Laplacian has exactly K non-zero eigenvalues, corresponding to the K communities. Let $\mathcal{X}$ be the N-by-K matrix of the orthonormal eigenvectors corresponding to the top K eigenvectors of $\mathcal{L}$. Under the SBM, rows of $\mathcal{X}$ have exactly K distinct rows, and under the DCBM, the rows of $\mathcal{X}$ point to at most K different directions. The actual graph corresponding eigenvector matrix $\mathbf{X}$ is a noisy version of $\mathcal{X}$ and therefore retains these structural properties, up to some obfuscation due to variability. But for the case of PABM, every node has parameters influencing the interaction of the node with different communities. As there is too much variation, it is not easy to null the variation among nodes and bring out the community assignment information.

4.3 Existing Spectral clustering methods

4.3.1 Spectral clustering over SBM

In Stochastic block model (SBM) [24], [22], for K community detection problem, we have the rank of probability matrix as K. So, perform spectral decomposition and pick the K most significant Eigenvalues and corresponding Eigenvectors. This results in representation

for every one of all the n points in a K dimensional space. For pure model probability matrix, we have all the points belonging to a community corresponding to a single point in K dimensional space. Hence, we could separate these K points and therefore have all the points binned into K communities.

Lower dimensional spectral representations for 2 and 3 community problem for original probability matrix of SBM model are shown in figures 4.1, 4.2.

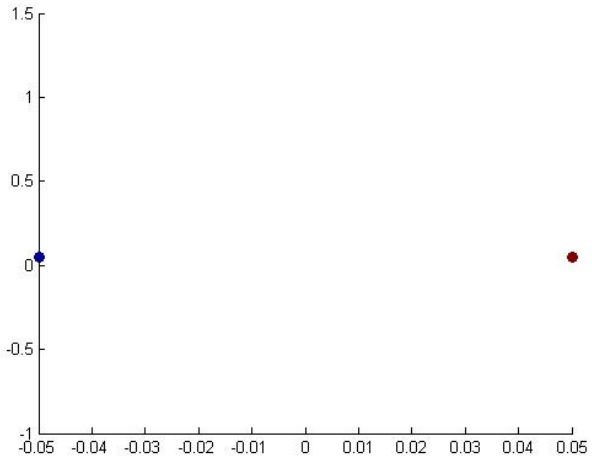


Figure 4.1: 2 dimensional representations of all points for 2 community detection problem given original probability matrix for SBM model.

Further, if we consider actual graph generated from the original probability matrix that we earlier used, we would get a clear separation and binning of points in a lower dimensional space under spectral representation. Though all the points exactly don't fall at the same location, they lie around the same location due to the noise that comes in while the graph is generated. This property lets us do a very simple distance-based clustering (k-means) thus allowing us group nodes into communities.

4.3.2 Spectral clustering over DCBM

Degree Corrected stochastic model DCBM) [28], [4], [3], [17] is a slightly more complex model than SBM allowing flexibility in degrees for nodes within the same cluster. In DCBM, for K community problem, we have that the rank of probability matrix as K. So, for spectral clustering, we perform a spectral decomposition and pick the K most significant Eigenvalues and corresponding Eigenvectors. Here, if we have K columns for the K chosen Eigenvectors, we will have n rows, each denoting a point in K dimensional Eigenspace. Unlike SBM model, these n points do not directly map to K points in Eigenspace but rather we have all the n points mapping to K distinct directions where all the points belonging to a community

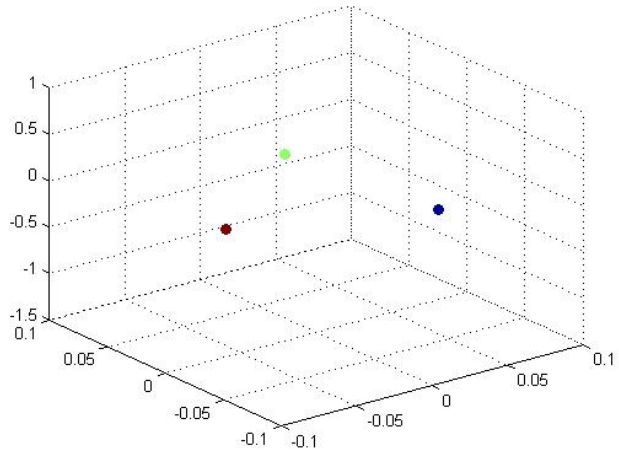


Figure 4.2: 3 dimensional representations of all points for 3 community detection problem given original probability matrix for SBM model.

lie in the same direction. So, if we nullify the magnitude difference, by row normalizing i.e. projecting all points onto a unit sphere in K dimensional space, we will again have a case similar to SBM where all the n normalized rows corresponding to n nodes in the graph converge to K points. It holds that all the points in a cluster correspond to a single point in K dimensional space. This would let us easily group the graph into K communities.

Lower dimensional spectral representations and normalized representations for 2 and 3 community problem for original probability matrix of DCBM model are shown in figures 4.3, 4.4 where figures 4.3a, 4.4a indicate simple representations of points in Eigen space while 4.3b, 4.4b indicate normalized representations of points in Eigen space.

In DCBM, normalizing each point which is equivalent to projecting each point onto a unit sphere in K dimensions will nullify the degree variation related noise and thus will result in same Eigenspace point for all nodes belonging to the same community.

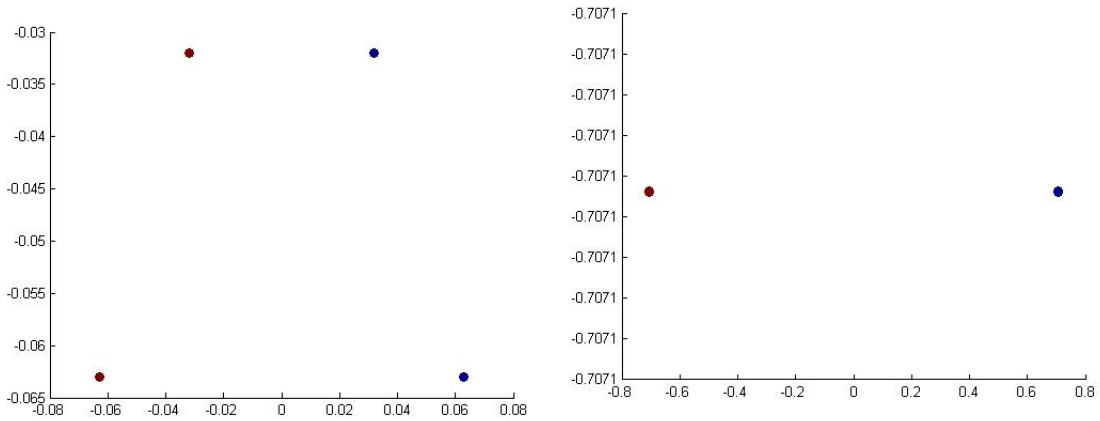
4.4 PABM properties

For popularity-adjusted block model (PABM), we have probability of an edge $\{i, j\}$ for $i < j$,

$$p_{ij} = \lambda_{ic_j}\lambda_{jc_i}, \tag{4.4}$$

where $\lambda_{ir}, 1 \leq i \leq n, 1 \leq r \leq K$, are the popularity parameters and $0 \leq p_{ij} \leq 1$ for all $i < j$. Here c_i and c_j indicate the clusters to which node i and node j belong to.

For spectral clustering, we first need to write down the model probabilities in a matrix form. For the PABM, there is no straightforward way to represent the probability matrix as a



(a) Simple representation in Spectral domain

(b) Normalized representation in Spectral domain

Figure 4.3: 2 dimensional representations of all points for 2 community detection problem given original probability matrix for DCBM model.

matrix product. We have

$$P = (\lambda M) \circ (\lambda M')' \tag{4.5}$$

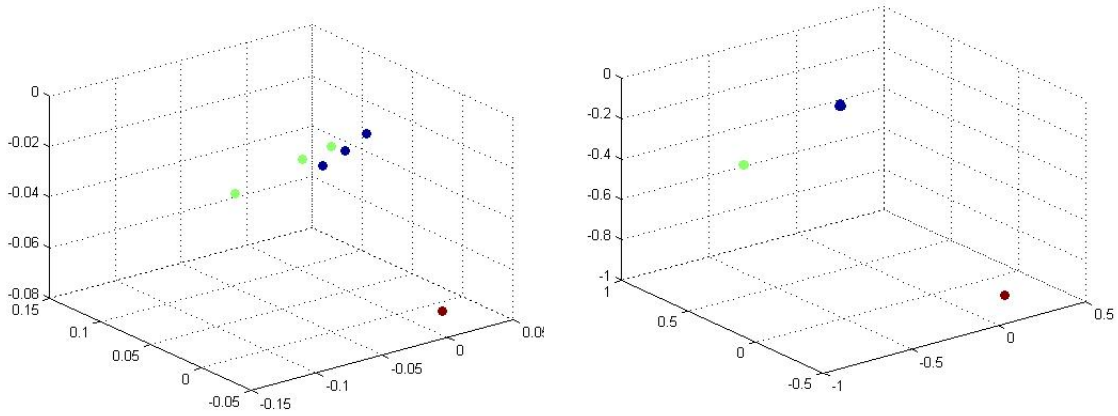
where $A \circ B$ represents the Hadamard product of A and B. Note that λM is n-by-n with rank K, and for Hadamard products, $rank(A \circ B) \leq rank(A) \times rank(B)$ and hence we have the rank to be less than or equal to k^2.

Under model (4.4), nodes can have varying popularity in different communities, which adds greater complexity to the spectral structure. To start with, we have that the graph Laplacian has

$$p = \min(n, K^2) \tag{4.6}$$

non-zero eigenvalues under this model, in contrast to K non-zero eigenvalues for the SBM and the DCBM. The spectral decomposition still has a structural connection with the community structure, albeit in a manner more subtle than under the SBM or the DCBM. By carefully analyzing the spectral decomposition under model (4.4), a connection between spectral structure and community structure can be established thereby developing a spectral clustering algorithm that will work under varying node popularities (PABM) model.

Since PABM probability matrix has rank less than or equal to k^2, we need to consider as many columns as the rank of the matrix, which in the worst case is k^2. Since all of them have information about communities, we need to use them to find community assignment by spectral clustering.



(a) Simple representation in Spectral domain

(b) Normalized representation in Spectral domain

Figure 4.4: 3 dimensional representations of all points for 3 community detection problem given original probability matrix for DCBM model.

4.5 Sub-communities

For the spectral clustering algorithm, we start out with the Eigen decomposition of $\mathcal{L}$. The Laplacian has a rank r less than or equal to K^2. Let $\mathcal{X}$ be the n-by-r matrix of orthonormal eigenvectors corresponding to to the top r eigenvectors of $\mathcal{L}$. Practically, we will consider $\mathbf{X}$ which is a noisy version of population version $\mathcal{X}$, the graph generated from the model, but it still retains the structural properties up to certain level.

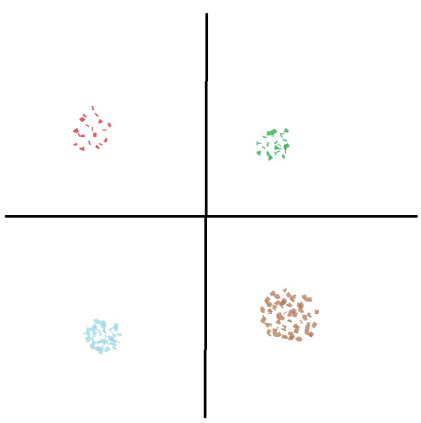
But in this case for algorithm design, we consider population version $\mathcal{X}$ which is obtained from the graph generated by the model. Every row of this matrix corresponds to a node in the network. A first level distance-based clustering on rows of $\mathcal{X}$ in r dimensional space results in K^2 clusters. Every community has different types of nodes within which form K different clusters thus resulting in K^2 clusters.

So, we have that a simple distance-based clustering in k^2 dimensional space results in k^2 sub-communities because of the rank of the matrix. It could be a slightly lower number of communities too based on the rank of the matrix. But even if we group into k^2 sub-communities we have the property that these sub-communities together belong to each community.

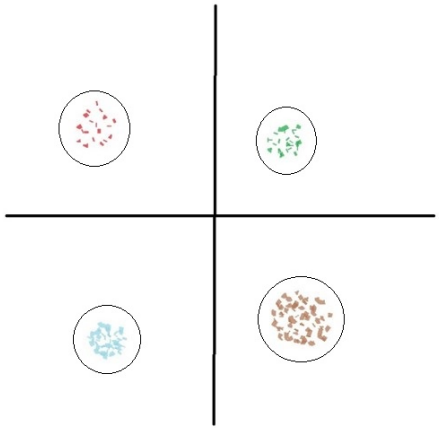
Now, we need to warp these K^2 sub-communities into K clusters where each indicates a community. So, we will only consider representatives (cluster centers) of these K^2 clusters and design some transformation/warping function which when applied on these K^2 points warps them into K points thus showing us the K communities in the network.

4.6 Spatial warping for two communities

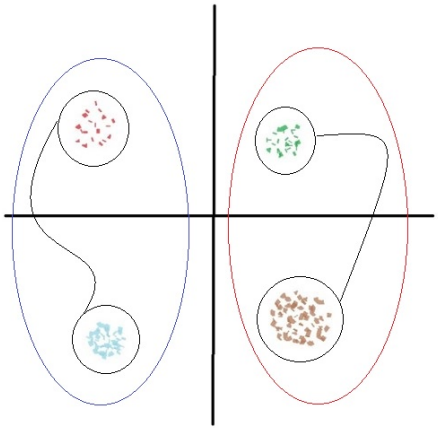
From the above section, we deduce that some warping function is needed to convert the k^2 sub-communities into k communities. We consider a specific case of $K = 2$ where we only have two communities. Here, we generate each value in λ matrix randomly and form the model. Even in this case, we get just $K^2 = 4$ representative cluster centers each corresponding to a small set of nodes. When these are warped to 2 clusters, we have the community assignment problem solved.



(a) Points in lower dimensional space



(b) Sub-communities by local clustering



(c) Warping to map four sub-communities to two communities.

Figure 4.5: Illustration of PABM spectral clustering method.

Figure 4.5 shows a rough illustration of how spectral clustering for PABM happens for two community problem. Assume the plots shown to be rough representations of 4-dimensional

space. Initially, in this 4-dimensional space all the rows are scattered as shown in figure 4.5a. But these points inherently group into four sub-communities in this 4-dimensional space as shown in figure 4.5b. Finally, we apply a warping/ spatial transformation which is shown in figure 4.5c to merge two sub-communities into a single cluster thus resulting in only two clusters which indicate true community assignments.

We consider a very specific case of a network with four nodes and two communities. Each node in the community behaves differently. Thus we have the Eigen matrix E which satisfies,

$$\mathcal{L}E = E\Lambda \tag{4.7}$$

where Λ is a diagonal matrix with Eigenvalues along the diagonal. Considering columns of Eigen matrix E, we have that they are Eigen vectors satisfying property that

$$\mathcal{L}y_i = \lambda_i y_i \tag{4.8}$$

The rows of Eigen matrix indicate nodes in the network in Eigenspace. Here, since we have only four nodes, we will have just four rows in Eigen matrix indicating the $K^2 = 4$ clusters in 4-dimensional Eigen space. We need to find a relationship between these rows so that two rows are grouped thus resulting in two communities.

We will consider a numerical example to understand the way warping function can be found.

Consider a network with 4 nodes and Laplacian matrix to be:

$$\mathcal{L} = \begin{bmatrix} 0 & 0.5819 & 0.0476 & 0.1455 \\ 0.5819 & 0 & 0.1455 & 0.4444 \\ 0.0476 & 0.1455 & 0 & 0.5819 \\ 0.1455 & 0.4444 & 0.5819 & 0 \end{bmatrix}$$

This Laplacian has Eigen values to be $0.2334, -0.5079, -0.7255, 1.0000$.

Now, we have the Eigen matrix of the Laplacian to be,

$$E = \begin{bmatrix} -0.5948 & 0.5618 & 0.3829 & 0.4289 \\ -0.3829 & -0.4289 & -0.5948 & 0.5618 \\ 0.5948 & 0.5618 & -0.3829 & 0.4289 \\ 0.3829 & -0.4289 & 0.5948 & 0.5618 \end{bmatrix}$$

The columns of this matrix are Eigenvectors.

The rows of this matrix are representatives of each node in 4-dimensional Eigenspace.

We notice that warping function, in this case, could be a rotational matrix. Here, we have rotation matrix to be

$$\hat{O} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix}$$

We have a relation between first two rows that,

$$\begin{bmatrix}-0.5948 & 0.5618 & 0.3829 & 0.4289\end{bmatrix}\begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ -1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\end{bmatrix} = \begin{bmatrix}-0.3829 & -0.4289 & -0.5948 & 0.5618\end{bmatrix}$$

$$\begin{bmatrix}-0.3829 & -0.4289 & -0.5948 & 0.5618\end{bmatrix}\begin{bmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ -1 & 0 & 0 & 0\\ 0 & -1 & 0 & 0\end{bmatrix} = -\begin{bmatrix}-0.5948 & 0.5618 & 0.3829 & 0.4289\end{bmatrix}$$

Similarly, we have the same relation to third and fourth rows. This indicates that once we discover this transformation $\hat{O}$, we can relate first two rows and next two rows thus indicating that first two points belong to one community and the next two points belong to another community.

But this warping function which is a specific rotational matrix, in this case, varies with different two community PABM models.

For different other networks with n nodes and 2 communities we have the transformation matrices to be,

$$\hat{O}_1 = \begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\\ 0 & -1 & 0 & 0\\ -1 & 0 & 0 & 0\end{bmatrix}$$

$$\hat{O}_2 = \begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 0 & -1 & 0\\ 0 & 1 & 0 & 0\\ -1 & 0 & 0 & 0\end{bmatrix}$$

So, generalizing a bit for network of n nodes and two communities, we have Eigen matrix to be E where columns of this matrix are Eigenvectors and satisfies

$$\mathcal{L}E = E\Lambda \tag{4.9}$$

We have 4 rows that represent the $K^2 = 4$ clusters. Let these rows be $x_1, x_2, x_3 and x_4$. Consider x_1 *and* x_2 to be of same community and x_3 *and* x_4 to be of other community. And let transformation matrix be $\hat{O}$, then we have

$$x_1\hat{O} = x_2 \tag{4.10}$$

$$x_2\hat{O} = -x_1 \tag{4.11}$$

Similarly, we have relations for x_3 *and* x_4.

So, we have this transformation matrix $\hat{O}$ as a function of the network Laplacian so that we get a different $\hat{O}$ for various networks.

4.6.1 Some Interesting properties of warping function

Warping function for two community PABM model obeys certain interesting characteristics.

- Rotational Matrix $\hat{O}$ is orthogonal i.e. all the columns of $\hat{O}$ are orthonormal which gives that $\hat{O}\hat{O}' = I$.
- We also have $\hat{O}^2 + I = 0$ and $\hat{O}' = -\hat{O}$. This implies that the rotational matrix is askew symmetric matrix.

4.6.2 Semi-supervised Approach

In this case, since we do not have a generalization where we can find transformation matrix (specifically the rotational matrix) as a closed form expression in terms of the graph, we could use the nodes information to compute warping which brings the problem under a semi-supervised learning framework.

We can consider that we know extra information about nodes from same community but within different sub-communities. Once this information is known, we can compute the rotation matrix and as a result, can complete the spectral clustering algorithm. So, we have a semi-supervised approach to be a method that can give correct community assignments even for popularity considered PABM model. Typically in a real-world network, we usually know the popular or highest degree nodes and their community information. So, if we have information about few high degree nodes within the same community, the algorithm can be completed. It is further shown in the experiments section below indicating how many points are needed to be known to perform spectral clustering for PABM model under semi-supervised learning framework.

4.7 Experiments and Results

In this section, we consider a PABM model and perform experiments over it for evaluating the theoretical claims made in the above sections. For this, we consider a graph generated by a PABM model, perform community detection by spectral clustering over it. For

multiple communities detection, we show that all the nodes in the graph get split into k^2 sub-communities where k is the number of communities in the PABM model. Considering the PABM edge probability matrix, we have pure model information without any noise, and hence the nodes cluster into k^2 communities exactly. Rather, all the nodes have one of the k^2 values in the k^2 dimensional space and hence converge to exactly k^2 points (which can be considered as sub-communities) on clustering. But when adjacency matrix of the graph obtained from PABM model is considered, there is a slight error while clustering into k^2 sub-communities where few of the nodes are mis-clustered into other sub-communities among the k^2 sub-communities. But once we warp the k^2 sub-communities to form k communities typically by applying a transformation, the error obtained from clustering, in this case, is compared to error obtained in the case where traditional normalized spectral clustering is performed. And for all the simulations, we have used a simple Intel i5 processor on a simple laptop (CPU based) without GPU.

4.7.1 Multiple communities

For multiple communities PABM model based graph, the idea is that since the rank of the matrix is k^2, we will have k^2 sub-communities when clustering is done in the k^2 dimensional space. Since we do not have a warping mechanism in place yet, we do a manual warping to bring the k^2 communities down to k communities. And then the error in this k community assignment is compared to the k community assignment done by normalized spectral clustering method.

Here, we consider the case of three community PABM model. We consider all the parameters of the PABM model to be random numbers in the range [0, 1]. So, if the graph has n nodes and k communities, nk random numbers are generated as parameters and used for the PABM model. In this case we consider $k = 3$. We run 100 simulations on PABM generated graphs with 150 nodes, 300 nodes and 450 nodes having an equal number of nodes in each of the three communities. We consider both the cases of probability matrix (which is the noiseless pure information matrix) and the graph generated from probability matrix. Errors are computed for both the PABM spectral clustering and normalized spectral clustering cases. The results comparison is shown in the table. Here, the average error is measured as the ratio of the number of mis-clustered nodes to the total number of nodes.

In this case, three communities are considered for different node graphs and both probability matrix and adjacency matrix are tested for error which is measured as the number of mis-clustered nodes after performing label matching.

Table 4.1: Comparison of PABM spectral clustering versus normalized spectral clustering for 3 community case.

	Probability Matrix		Generated Graph	
Nodes	PABM	Normalized	PABM	Normalized
150	0	0.16	0.1267	0.2467
300	0	0.233	0.0667	0.24
450	0	0.3178	0.0044	0.3578

4.7.2 Two communities

For two communities PABM model based graph, we try to perform experiments and see that $k^2 = 4$ sub-communities are formed. We also experiment and see that in a semi-supervised kind of setting where two or more nodes from same community are known, the warping can be determined and hence PABM spectral clustering can be solved. In comparison, normalized spectral clustering is also run to show the difference in error. Here, the error is computed as the hamming difference between community assignments after label matching is done.

Here, we consider the case of two community PABM model. We consider all the parameters of the PABM model to be random numbers in the range [0, 1]. So, if the graph has n nodes, $2n$ random numbers are generated as parameters and used for the PABM model.

Sub-communities

Here, we consider a PABM model with 400 nodes. In this network, if we perform clustering in $k^2 = 4$ dimensional space, we have all the nodes divided into four sub-communities.

The centers of the sub-communities are given in the table.

Table 4.2: Cluster centers for 400 nodes two community PABM model graph

0.5370	-0.6643	0.1975	0.4359
-0.3933	-0.1413	-0.6887	0.5826
0.5370	0.6643	-0.1975	0.4359
-0.3933	0.1413	0.6887	0.5826

With the cluster centers shown in table 4.2, all the nodes of the graph fall into four sub-communities where we have all the nodes from a community distributed among two sub-communities and all the nodes from a different community are distributed into two other sub-communities. The cluster centers shown indicate that there exists a simple rotational matrix that can warp the four sub-communities into two sub-communities.

In this case, the warping function can be a skew-symmetric orthogonal matrix which is as follows:

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}$$

This when applied, the four sub-communities warp into two giving the exact community assignments and thus perfectly detecting communities for PABM model.

Semi-supervised Communities

Here, we consider that we already have $k^2 = 4$ sub-communities. But we need to warp these into two communities to solve the assignment problem. So, we consider a semi-supervised approach where if we are given prior information about few nodes in one community, we can use that to find the warping needed. Typically we need these nodes to be falling into different sub-communities to be able to find the warping pattern. Here, we consider the method of choosing few of the highest degree nodes from a community as known information and find warping to arrive at actual community assignment. This number of high degree nodes that have been used is also noted in the table.

We run 100 simulations on graphs with 100 nodes, 200 nodes and 400 nodes having an equal number of nodes in both communities. We consider both the cases of probability matrix (which is the noiseless pure information matrix) and the graph generated from probability matrix. Errors are computed for both the PABM spectral clustering and normalized spectral clustering cases. The result comparison is shown in a table. Here, the error is measured as the ratio of the number of mis-clustered nodes to the total number of nodes.

Table 4.3: Comparison of PABM spectral clustering versus normalized spectral clustering for 2 community case.

	Probability Matrix		Generated Graph		Number of nodes for
Nodes	PABM	Normalized	PABM	Normalized	semi-supervised warping
100	0	0.2	0.03	0.23	2
200	0	0.19	0.02	0.205	2
400	0	0.1925	0	0.1825	6

In this case, two communities are considered for different node graphs and both probability matrix and adjacency matrix are tested for error which is measured as the number of

mis-clustered nodes after performing label matching. And the number of nodes for semi-supervised warping indicates the minimum number of high degree nodes information needed from a community to find the warping. This establishes our method as a kind of semi-supervised learning method where information about few nodes from a community is needed.

4.8 Observations

From the results, we observe that PABM spectral clustering works very well for PABM model networks. In case of the probability matrix, as it does not have any noise and is pure information, we get the community detection error to be exactly zero and hence a perfect community assignment. This is true for both two and three community cases and also in general holds for PABM model with any number of communities. But even using the probability matrix, performing normalized spectral clustering yields bad results.

In case of an actual graph from the PABM model, which is the noisy version of the pure probability matrix, we have some error from the PABM spectral clustering method but this is very small and can almost be considered as zero compared to performing a normalized spectral clustering method.

In both the two community and three community cases, we do not have a warping method to relate k^2 sub-communities to the actual k communities they belong to. So, for the experiments over three community graphs, we manually warp the k^2 sub-communities to the k actual communities before computing error.

For two community case, we see that there exists a relation between sub-communities centers belong to the same actual community. There exists a skew-symmetric orthogonal rotation matrix that can warp these sub-communities to actual communities. And as part of a semi-supervised approach, we consider information about few nodes as known to find the rotational matrix that can act like warping function. Ideally, we would need information about nodes from same community but different sub-communities to compute the rotational warping relation. We have used a method where we select a certain number of high degree nodes and hope that we have nodes from both sub-communities. In the results, we also have the minimal number of nodes that were required tabulated. We can see that around 2%-3% of high degree nodes from a community is needed to determine the rotational warping function. This isn't a large number of nodes. Typically even in real networks we usually have at-least few nodes about which we are sure that they belong to the same community. These would be few well-connected prominent nodes in the network. This information would be enough to perform the PABM spectral clustering method on the whole network.

Here, we have considered a higher dimensional space k^2 in contrast to the earlier methods that considered k dimensional space. This also explains that due to the usage of data in a higher dimensional space, more information is represented and hence a better performance is shown in clustering results when compared to other spectral clustering methods.

4.9 Conclusion

PABM Spectral clustering proves to be a very superior and exact method for PABM model graphs. And PABM is an advanced model and better fits for most of the real-world networks, we can deduce that this method works well as an improvement to the existing method even for real-world networks. But there needs to a generalized expression for warping function to making this applicable for all cases.

Chapter 5

Conclusions

In this thesis, we have proposed methods for improving community detection algorithms in different scenarios. One method considered is the popular spectral clustering algorithm for which a sub-sampling based method is introduced so that it out-performs the existing spectral clustering. Another method considered is the extreme point method which finds the best community assignment for a generalized optimization function. Extreme point algorithm is also extended and experimented for multiple communities detection problem. We also analyze spectral properties of PABM model to design a spectral clustering algorithm which is exact for PABM model based graph.

Existing Spectral clustering algorithm is made to perform better by using the sub-sampling method. In the sub-sampling method, we split the original graph into subgraphs by maintaining a certain number of nodes as common in all the subgraphs. After performing spectral clustering over each of these subgraphs, community assignments are stitched together to obtain the community assignment for the entire graph. Experiments are conducted on SBM, DCBM model graphs and also on real graphs. In all of the simulations, we get the runtime in sub-sampling method to be much smaller than the actual spectral clustering method and hence giving computational gains. And even within our sub-sampling method, we have the run-times and error rates varying for a different set of parameters. It is deduced that smaller number of subgraphs implies larger subgraphs and hence smaller error while more runtime. Also, higher overlap size implies larger subgraphs and therefore smaller error while also increasing runtime because of the larger size of subgraphs.

Extreme point algorithm works well for two communities scenario. It works for any general optimization functions such as SBM, DCBM and PABM modularities which try to fit the graph to the respective model. But extreme point computation gets expensive for large networks as complexity increases with network size. So, extreme point algorithm via sub-sampling is proposed to obtain computational gains. In the sub-sampling method, the original graph is divided into subgraphs and extreme point algorithm is applied on subgraphs. Using the modularity or the optimization function, we compute the best community

assignment for the subgraphs. Further, these subgraph community assignments are stitched to form the community assignment for the entire graph. The runtime for the sub-sampling based algorithm is much smaller compared to the actual method for any given set of parameters. And within the sub-sampling based algorithm, with an increase in overlap size, runtime increases while the error is low and with an increase in the number of subgraphs, runtime decreases while error increases due to smaller subgraph size.

Existing extreme point algorithm works for two community problem while for multiple communities, the lower dimensional space is not enough to encode the multiple community information. So, a slightly higher dimensional space is designed and assignments are projected into this space. Extreme points detected in this space are good candidates for multiple community detection. An iterative approach is also proposed for this extreme point computation in case of multiple communities. It works for pure information matrix and approximately for model generated graphs. The issue with this method is that it is computationally very expensive for communities greater than two and even for a small number of nodes, the algorithm is infeasible. Sub-sampling based approach is also applied in this case which resulted in bringing down the computation time but even this approach saturates for a small number of nodes as multiple community extreme points algorithm takes exceptionally high time to run even on subgraphs.

Further, an analysis is done and patterns are discovered for PABM model spectral decomposition. We make the traditional spectral clustering applicable to PABM model graph by moving to a slightly higher dimensional space where k^2 dimensional space is considered. In this space, we see that k^2 sub-communities are formed by performing a simple distance-based clustering. These k^2 columns need to be transformed to form k communities indicating the communities in PABM model. For higher number of communities, we prove the existence of theses k^2 sub-communities and also a manual transformation of these k^2 sub-communities shows that PABM spectral clustering is accurate provided we have the general function to transform the k^2 sub-communities into k communities. Also, for the case of two community PABM model, distance-based clustering in 4-dimensional space leads to 4 sub-communities but we do not have a function to transform these into two communities. We consider a semi-supervised approach where information about a certain number of nodes from a community is assumed to be known. Using this information, the sub-communities belonging to each community are identified and further community assignment is obtained for the whole graph. This approach has given very high accuracies for PABM model graph compared to the actual normalized spectral clustering method.

For the future work, we could work on the following problems as a follow up to the existing work done in this thesis. In the extreme point method for multiple communities detection, convex hull computation is a very expensive step making the method infeasible. So, if the convex hull computation method could be improved for this particular case where the set of points are the projections of community assignments into a lower-dimensional space, the entire extreme point method for multiple communities speeds up making its implementation feasible. Also, in case of spectral clustering for PABM we still do not have a general case

warping or normalization function to convert the k^2 sub-communities to k communities. If this transformation is identified, the problem can be modeled as an unsupervised learning problem rather than semi-supervised learning problem.

Chapter 6

Summary

Most of the complex real-world systems can be represented as a network graph as it can represent the entities and their interactions mathematically. This mathematical abstraction of the network helps us to perform analysis and observe interesting properties in a network. Community structure is one such interesting phenomena where an inherent grouping of nodes exists in the graph and the problem of finding these community assignments for all the nodes is called community detection. In all of the above work for thesis, we have worked on improving these community detection algorithms making it much more useful for practical implementations. We have worked on both aspects of making the algorithm perform better in terms of accuracy and also improving the performance of algorithm by lowering the runtime as a result of computational gains. Both spectral clustering and extreme point algorithm have been improved in terms of runtime following the proposed sub-sampling based approach. Also, extreme point algorithm is extended for multiple community detection making it useful for a more general set of graphs. An improvement in terms of error rate has been brought for Spectral clustering method specifically for PABM model graphs thus making the algorithm more accurate on a whole.

Bibliography

[1] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 us election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[2] Arash A Amini, Aiyou Chen, Peter J Bickel, Elizaveta Levina, et al. Pseudo-likelihood methods for community detection in large sparse networks. *The Annals of Statistics*, 41(4):2097–2122, 2013.

[3] Peter J Bickel and Aiyou Chen. A nonparametric view of network models and newman–girvan and other modularities. *Proceedings of the National Academy of Sciences*, 106 (50):21068–21073, 2009.

[4] Peter J Bickel and Purnamrita Sarkar. Hypothesis testing for automated community detection in networks. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 78(1):253–273, 2016.

[5] Fan Chung and Linyuan Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

[6] Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

[7] Stephen E Fienberg, Michael M Meyer, and Stanley S Wasserman. Statistical analysis of multiple sociometric relations. *Journal of the american Statistical association*, 80 (389):51–67, 1985.

[8] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[9] Komei Fukuda. From the zonotope construction to the minkowski addition of convex polytopes. *Journal of Symbolic Computation*, 38(4):1261–1272, 2004.

[10] Jing Gao, Feng Liang, Wei Fan, Yizhou Sun, and Jiawei Han. Graph-based consensus maximization among multiple supervised and unsupervised models. In *Advances in Neural Information Processing Systems*, pages 585–593, 2009.

[11] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[12] Anna Goldenberg, Alice X Zheng, Stephen E Fienberg, Edoardo M Airoldi, et al. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2(2):129–233, 2010.

[13] Peter Gritzmann and Bernd Sturmfels. Minkowski addition of polytopes: computational complexity and applications to gröbner bases. *SIAM Journal on Discrete Mathematics*, 6(2):246–269, 1993.

[14] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[15] Ming Ji, Yizhou Sun, Marina Danilevsky, Jiawei Han, and Jing Gao. Graph regularized transductive classification on heterogeneous information networks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 570–586. Springer, 2010.

[16] Pall F Jonsson, Tamara Cavanna, Daniel Zicha, and Paul A Bates. Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. *BMC bioinformatics*, 7(1):2, 2006.

[17] Antony Joseph and Bin Yu. Impact of regularization on spectral clustering. *arXiv preprint arXiv:1312.1733*, 2013.

[18] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical Review E*, 83(1):016107, 2011.

[19] Eric D Kolaczyk. Models for network graphs. *Statistical Analysis of Network Data*, pages 1–44, 2009.

[20] Can M Le, Elizaveta Levina, and Roman Vershynin. Optimization via low-rank approximation for community detection in networks. *arXiv preprint arXiv:1406.0067*, 2014.

[21] Can M Le, Elizaveta Levina, Roman Vershynin, et al. Optimization via low-rank approximation for community detection in networks. *The Annals of Statistics*, 44(1): 373–400, 2016.

[22] Jing Lei, Alessandro Rinaldo, et al. Consistency of spectral clustering in stochastic block models. *The Annals of Statistics*, 43(1):215–237, 2015.

[23] Mark Newman. *Networks: an introduction*. Oxford university press, 2010.

[24] Karl Rohe, Sourav Chatterjee, Bin Yu, et al. Spectral clustering and the high-dimensional stochastic blockmodel. *The Annals of Statistics*, 39(4):1878–1915, 2011.

[25] Srijan Sengupta and Yuguo Chen. A block model for node popularity in networks with community structure. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 2017.

[26] Christophe Weibel. Implementation and parallelization of a reverse-search algorithm for minkowski sums. In *2010 Proceedings of the Twelfth Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 34–42. SIAM, 2010.

[27] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[28] Yunpeng Zhao, Elizaveta Levina, Ji Zhu, et al. Consistency of community detection in networks under degree-corrected stochastic block models. *The Annals of Statistics*, 40 (4):2266–2292, 2012.

Appendices

Appendix A

Lower bound on overlap size

Consider a graph $G = \{V, E\}$ where $\{V\}$ denotes the set of all the nodes and $\{E\}$ denotes the set of all the edges between nodes. Let there be k communities in the graph and the number of nodes in a community i is denoted by n_i where we have $\sum_{i=1}^{k} n_i = n$. Now, we need to select few nodes from this entire set $\{V\}$ to form the overlap region. Here, we assume that the overlap region needs to be a good representative of the graph. So, we consider the worst case scenario where nodes from one of the community are completely missing from the overlap region. We want to choose the size for overlap region such that at least one node from each community is present in the overlap region with very high probability.

Let x_i denotes the number of nodes from community i that are present in the overlap region. Actually, we choose nodes in overlap region by selecting nodes without replacement. Consider the selection of $\{x_1, x_2, ...x_k\}$ to be following a multinomial distribution with total number of nodes in overlap being $o \times n$ and since nodes in the overlap region needs to be good representatives of nodes in the graph, we have the probability of choosing node from community i as $\frac{n_i}{n}$ where this fraction denotes the proportion of nodes in community i compared to total nodes in actual graph.

$$\{x_1, x_2, ...x_k\} \sim multinomial(on, \frac{n_1}{n}, \frac{n_2}{n}, ..., \frac{n_k}{n}) \tag{A.1}$$

We can assume independence that all the nodes from different community are chosen independently and hence we have for each community i,

$$x_i \sim binomial(on, \frac{n_i}{n}) \tag{A.2}$$

From this, we have the probability of having zero nodes from community i in overlap as

$$P(x_i = 0) = \left(1 - \frac{n_i}{n}\right)^{o \times n} \tag{A.3}$$

So, for the worst case scenario we considered, where one of the community does not have any of its nodes in the overlap region, we have probability of that happening to be a union of all the cases where any of the community nodes can be absent giving,

$$P_0 = P(x_1 = 0) \cup P(x_2 = 0)..... \cup P(x_k = 0)$$

Probability of union of events is less than or equal to sum of probabilities of each event

$$\leq P(x_1 = 0) + P(x_2 = 0)..... + P(x_k = 0)$$

which is less than or equal to the worst case probability for all communities

$$\leq k \max_{1 \leq i \leq k} P(x_i = 0)$$

considering that number of nodes in the worst case community are very few compared to average number of nodes in a community by an order of 0.05

$$= k \left(1 - \frac{1}{20k}\right)^{o \times n}$$

Now, we have the worst case probability P_0 denoting the case where nodes from at-least one community need to be absent. So, we need to set some tolerance on level of P_0 which in ideal case needs to be zero. Hence, let the tolerance we choose be ϵ. Then,

$$P_0 \leq \epsilon$$

Applying log on both sides, we get

$$log(P_0) \leq log(\epsilon)$$

Substituting P_0, we get

$$log(k) + (on) log \left(1 - \frac{1}{20k}\right) \leq log(\epsilon)$$

$$(on) log \left(1 - \frac{1}{20k}\right) \leq log(\epsilon) - log(k)$$

Since we divide by a negative number

$$o \geq \frac{log(\epsilon) - log(k)}{n log \left(1 - \frac{1}{20k}\right)}$$

So, we have a bound on o indicating that the overlap we choose for our algorithm needs to depend on the size of the graph and also on the number of communities. We maintain our overlap size choice to be greater than this, which is given by

$$o \geq \frac{log(\epsilon) - log(k)}{n \times log \left(1 - \frac{1}{20k}\right)} \tag{A.4}$$

where ϵ is the tolerance level for probability that the worst scenario of an entire community missing from overlap region happens, k is the number of communities and n is the number of nodes in the graph.

Appendix B

Computational Complexity

Computational complexities for sub-sampling technique in case of Spectral clustering and extreme point algorithm are analyzed below and also compared with complexities of the original algorithms that worked without sub-sampling. We consider a graph of N nodes and k communities.

B.1 Spectral Clustering

In the case of spectral clustering, for the actual algorithm, we have several steps like Laplacian computation, Eigen decomposition, row normalization and applying k-means. Laplacian computation has a complexity of $\mathcal{O}(N^2)$, Eigen decomposition has a complexity of $\mathcal{O}(N^3)$, Row normalization has around $\mathcal{O}(Nk)$ and k-means has complexity $\mathcal{O}(Nk^2)$. Out of all of these, we have Eigen decomposition to be the most expensive step which dominates the complexity hence making the complexity of actual spectral clustering to be $\mathcal{O}(N^3)$.

When the sub-sampling technique is used, we compute complexity for both subgraph clustering step and stitching step separately. For the subgraph clustering step, since the size of subgraph decreases and is less than the original graph, complexity reduces. Since, we have s number of subgraphs to deal with each of size r where r contains overlap size o and also the distinct set of points $m = \frac{1-o}{s}$ resulting in r to be

$$r = o + \frac{1-o}{s}$$

And hence, we have the complexity of the subgraph clustering step to be,

$$\mathcal{O}\left(s\left(\frac{(1+o(s-1))N}{s}\right)^3\right)$$

For the stitching step, a label matching and a label switching step are involved. For every label match, we perform switching and this happens for all the subgraphs and an extra overall label assignment step happens, hence leading to a complexity of

$$\mathcal{O}((sk!o) + N)$$

which is approximately equal to

$$\mathcal{O}((sk^k o) + N)$$

B.2 Extreme point

In case of extreme point algorithm, there are several steps like Eigen decomposition, the projection of assignments into Eigen space, Convex hull computation and modularity computation. Out of these, Eigen decomposition and Convex hull computation are two most expensive steps having complexities $\mathcal{O}(N^3)$ and $\mathcal{O}(N^{k(k-1)}k^{2d-1})$ respectively. Depending on the value of k, different step dominates thus influencing the complexity.

For two community detection (k=2), we have complexity of Eigen decomposition step to be $\mathcal{O}(N^3)$ while complexity of Convex hull computation becomes $\mathcal{O}(N^2)$. So, the complexity of overall extreme point algorithm becomes $\mathcal{O}(N^3)$. And if we use sub-sampling technique, similar to the above shown spectral clustering case, complexity of subgraph step becomes

$$\mathcal{O}\left(s\left(\frac{(1+o(s-1))N}{s}\right)^3\right)$$

and complexity of stitching step becomes

$$\mathcal{O}((sk^k o) + N)$$

For multiple community detection ($k > 2$), complexity of Convex hull computation dominates and hence the overall complexity becomes $\mathcal{O}(N^{k(k-1)})$. And when sub-sampling technique is used, the complexity of subgraph step dominates compared to the stitching step thus making the overall complexity

$$\mathcal{O}\left(s\left(\frac{(1+o(s-1))N}{s}\right)^{k(k-1)}\right)$$

Appendix C

Modularities

For a given graph adjacency matrix A and a community assignment vector e, a specific form of the optimization function is needed to be able to apply the extreme point algorithm [21]. The general form of optimization function is given as,

$$f_A(e) = \sum_{j=1}^{q} g_j(h_{A,j}(e))$$

where g_j are scalar functions on R and $h_{A,j}(e)$ are quadratic functions of A and e given as,

$$h_{A,j}(e) = (e + s_{j1})^T A(e + s_{j2})$$

Here, q is a fixed number while s_{j1} and s_{j2} are constant vectors.

Since, we only have adjacency matrix A as input data and community assignment e as the required community assignment, we only have optimization function f to be a function of A and e. For this particular optimization function $f_A(e)$, we can define the best community assignment as the one that maximizes the optimization function.

We have the best community assignment e^* as,

$$e^* = argmax\{f_A(e), e \in \epsilon_A\}$$

where ϵ_A indicates the set of all the extreme points of the convex hull.

For specific models like SBM, DCBM and PABM, we have modularity functions that satisfy the above general form and which when optimized try to fit the graph into the respective model. So, optimizing those modularities gives the community assignment that best fits that model.

Consider that the graph has n nodes and hence e will an n-dimensional vector. Let there be K communities. We have number of nodes in community k given as, $n_k(e) = \sum_{i=1}^{n} Ie_i = k$.

From this, we will have number of edges between communities k and l given by

$$O_{kl}(e) = \sum_{i,j=1}^{n} A_{ij} I\{e_i = k, e_j = l\}$$

where $k \neq l$.

We also have sum of node degrees in community k given by, $O_k = \sum_{l=1}^{K} O_{kl}$. Degree of node i in the graph is given by $d_i = \sum_{j=1}^{n} A_{ij}$ and hence, total number of edges in graph is given by $m = \sum_{i=1}^{n} d_i$ where m is actually twice the number of edges in the graph. We also have popularity of node i in community r defined as $M_{ir} = \sum_{j \in N_r} A_{ij}$ where N_r denotes the set of all the nodes in community r.

Now, according to Chung-Lu model [5] for the entire graph, Newman-Girvan modularity is defined as

$$Q_{NG}(e) = \frac{1}{2m} \sum_{k} \left(O_{kk} - \frac{{O^2}_k}{m} \right)$$

For Stochastic block model [18], we have the modularity function defined as

$$Q_{SBM}(e) = \sum_{k,l=1}^{K} O_{kl} log \frac{O_{kl}}{n_k n_l}$$

For Degree Corrected block model [20], we have the modularity function defined as

$$Q_{DCBM}(e) = \sum_{k,l=1}^{K} O_{kl} log \frac{O_{kl}}{O_k O_l}$$

For Popularity adjusted block model [25], we have the modularity defined as

$$Q_{PABM}(e) = 2 \sum_{i} \sum_{r} M_{ir} log(M_{ir}) - \sum_{rs} O_{rs} log(O_{rs})$$

We can prove that all of the above modularities can be expressed in the general optimization function form as mentioned above and therefore for all of these modularities, extreme point algorithm works.