I, Divya Sardana, hereby submit this original work as part of the requirements for the degree of Doctor of Philosophy in Computer Science & Engineering.

It is entitled:
Analysis of Meso-scale Structures in Weighted Graphs

Student's name: Divya Sardana

This work and its defense approved by:

Committee chair: Raj Bhatnagar, Ph.D.
Committee member: Yizong Cheng, Ph.D.
Committee member: Anil Jegga, B.V.Sc & A.H.
Committee member: Ali Minal, Ph.D.
Committee member: Tomasz Stepinski, Ph.D.
Analysis of Meso-scale Structures in Weighted Graphs

A dissertation submitted to the

Graduate School
of the University of Cincinnati

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in the Department of
Electrical Engineering and Computer Science
of the College of Engineering and Applied Science

July 2017

by

Divya Sardana
MS (CS), Univ. Cincinnati

Dissertation Advisor and Committee Chair:
Dr. Raj Bhatnagar

Dissertation Co-Advisor and Committee Co-Chair:
Dr. Anil Jegga
Abstract

Many real world systems across multiple disciplines, like social, biological and information networks can be described as complex networks, i.e., assemblies of nodes and edges having nontrivial topological properties. Along with network topology, most real world networks provide useful connectivity strength information between nodes in the form of edge weights. For example, in a co-authorship network, edge weights could represent the number of publications shared by two authors. This information of edge weights can be leveraged along with the network topological properties to uncover many hidden and interesting properties inherent in these networks. Meso-scale structures can help in analysing network properties which are not very evident at either the global level (eg. network diameter) or at the local level (eg. node degree). In this dissertation, we develop novel methodologies for the analysis of weighted networks or graphs at the meso-scale. Specifically, we develop algorithms to find two types of meso-scale structures in graphs, namely, community structure and core periphery structure.

The first problem that we solve involves finding density-based, disjoint clusters or communities in a weighted graph. We develop a novel graph clustering algorithm called G-MKNN based upon a node affinity measure called Mutual K-nearest neighbors. Our algorithm is based upon a new
definition of density called SE-density which tries to achieve a balance be-
tween structural density and edge-weight homogeneity in the final clus-
tering. We compare our algorithm with other state-of-the-art weighted
and un-weighted graph clustering algorithms using both synthetic and real
world protein-protein interaction (PPI) datasets.

The second problem that we solve involves extracting core periphery
structures in weighted graphs. In this work, first, we formalize the defi-
nition of core periphery structures for weighted networks. Next, we build
two algorithms to extract core-periphery structures in a weighted graph.
We also develop a methodology to categorize and score the peripheries sur-
rounding a core. Using synthetic and real world datasets, we demonstrate
the importance of studying core-periphery structures in weighted graphs.
We further provide a case study using a dataset of crimes taking place in
San Francisco to demonstrate the usefulness of core periphery structures
found by the developed algorithms in studying temporal and spatial net-
works.

An inherent drawback in most clustering algorithms is that they do not
find any overlap among the clusters, which can be a source of crucial infor-
mation in many areas. Moreover, overlapping communities are naturally
present in many domains such as social networks. In the third problem, we
extend our graph clustering algorithm developed in problem one to find
overlapping clusters (OG-MKNN). We use both a synthetic dataset as well
as a real world weighted social network to illustrate the effectiveness of our
The fourth problem that we solve involves developing a semi-supervised clustering algorithm (SG-MKNN) with the help of biological domain knowledge to find clusters in protein-protein interaction networks. We embed gene ontology based functional information into an MKNN based clustering algorithm for making better informed clustering decisions. We demonstrate the effectiveness of the algorithms developed using real world PPI datasets.
Dedicated to my Late Grandmother Mrs. Kaushalya Devi Sardana
Acknowledgements

I heartily thank my advisor, Dr. Raj Bhatnagar and my co-advisor, Dr. Anil Jegga for their support and guidance throughout my dissertation work. I consider it a privilege that I got an opportunity to work with them.

I thank my dissertation committee members Dr. Yizong Cheng, Dr. Ali Minai and Dr. Tomasz Stepinski for their valuable guidance and feedback on my work. I want to thank Dr. Bruce Aronow, Ms. Diana Boone, Dr. Jon Iverson, and Ms. Josette Reip for providing me employment during the course of my PhD. Their financial support has been indispensable in my studies.

I thank my parents who have supported me throughout my studies and constantly provide me motivation to work hard through their example. A special thanks to my husband Vineet, who has been a pillar of strength during my dissertation. I thank my sisters, Monika and Tanu, and my brother, Keshav, for their help and support during my dissertation.

I want to thank my friends for their constant support and understanding throughout my dissertation work. I want to thank Junu, Subasri, Mukta, Viji, Karen, Kirana, Shruti, Carrie, Dushman, Vaibhav, Dippy and Jagadeesh for their camaraderie.
# Contents

Abstract ................................................................. i

1 Introduction ......................................................... 1
   1.1 Graph-Mutual K-Nearest Neighbor (G-MKNN) Clustering ... 6
   1.2 Discovery of core-periphery structures using MKNN ....... 8
   1.3 Overlapping Graph-Mutual K-Nearest Neighbor (OG-MKNN)
       Clustering ......................................................... 10
   1.4 Functional information guided G-MKNN clustering for PPI
       networks ......................................................... 11
   1.5 Conclusion ...................................................... 13
   1.6 Dissertation Outline ......................................... 14

2 Graph-Mutual K-Nearest Neighbor Clustering (G-MKNN) .... 15
   2.1 Background/Motivation ........................................ 16
   2.2 Related Work .................................................. 19
      2.2.1 Molecular COMplex DEtection (MCODE) .............. 20
      2.2.2 Markov Clustering Algorithm (MCL) ................. 23
3.1 Background .............................................. 60
3.2 Related Work ........................................... 63
  3.2.1 Clustering with Overlapping Neighborhood Expansion (ClusterONE) ................. 65
3.3 Our Approach ............................................ 66
  3.3.1 Formal Definitions .................................... 67
  3.3.2 CP-Greedy: A Greedy Core Periphery Algorithm ........................................ 71
    3.3.2.1 Initialization Phase ............................... 71
    3.3.2.2 Greedy Growth Phase ............................. 72
    3.3.2.3 Overlap Phase .................................... 74
    3.3.2.4 Extraction of Core Periphery associations ..... 77
  3.3.3 CP-MKNN: An MKNN based Core Periphery Algorithm ................................. 78
    3.3.3.1 Initialization Phase ............................... 80
    3.3.3.2 CP-MKNN Phase 1 ................................. 80
    3.3.3.3 CP-MKNN Phase 2 (Merge and Boundary Formation) .......................... 80
    3.3.3.4 CP Phase: Core Periphery Structure Formation .................................. 85
  3.3.4 Complexity Analysis .................................. 87
3.4 Experimental Evaluation ................................. 90
  3.4.1 Results for Synthetic Datasets .......................... 90
    3.4.1.1 Evaluation Measures ............................. 90
    3.4.1.2 Results ........................................ 93
3.4.2 Results for Real Datasets . . . . . . . . . . . . . . . . 95
  3.4.2.1 Evaluation Measures . . . . . . . . . . . . . . . 95
  3.4.2.2 Results . . . . . . . . . . . . . . . . . . . . . 96
3.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 114

4 Overlapping Graph-Mutual K-Nearest Neighbor Clustering (OG-MKNN) 115
  4.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . 115
  4.2 Related Work . . . . . . . . . . . . . . . . . . . . . . . . . . 116
    4.2.1 Clique Percolation Method (CFinder) . . . . . . . . . 117
  4.3 Our Approach . . . . . . . . . . . . . . . . . . . . . . . . . . 118
    4.3.1 OG-MKNN Phase 1 . . . . . . . . . . . . . . . . . . . . . 118
    4.3.2 OG-MKNN Phase 2: Merge and Boundary Formation 119
    4.3.3 OG-MKNN Phase 3: Overlap Phase . . . . . . . . . . . 119
    4.3.4 Computational Complexity . . . . . . . . . . . . . . . 120
  4.4 Experimental Evaluation . . . . . . . . . . . . . . . . . . . 120
    4.4.1 Synthetic Dataset . . . . . . . . . . . . . . . . . . . . . 120
    4.4.2 Real Dataset Results . . . . . . . . . . . . . . . . . . . 124
    4.4.2.1 Dataset Description: A Social Network . . . . . . . . 124
    4.4.2.2 Evaluation Measure used . . . . . . . . . . . . . . . 125
    4.4.2.3 Results . . . . . . . . . . . . . . . . . . . . . 127
  4.5 Conclusion . . . . . . . . . . . . . . . . . . . . . . . . . . . 130

5 Functional Information Guided Semi-supervised Graph Cluster-
5.1 Background .......................................................... 132
5.2 Related Work ......................................................... 133
  5.2.1 Semantic Similarity Measures between GO Terms .................. 136
  5.2.2 Gene Ontology based Clustering Methods for PPI networks .......... 138
    5.2.2.1 Using Gene Ontology (GO) to filter out clusters ................. 138
    5.2.2.2 Using Gene Ontology (GO) in the clustering process ............... 139
5.3 Our Approach ....................................................... 140
  5.3.1 Pre-processing based SG-MKNN .................................. 141
  5.3.2 Post-processing based SG-MKNN .................................. 141
  5.3.3 Complexity Analysis ............................................. 143
5.4 Experimental Evaluation ............................................ 144
5.5 Conclusion .......................................................... 144

6 Conclusions and Future Directions ..................................... 146
  6.1 Future Work ....................................................... 147
    6.1.1 G-MKNN Clustering with Node Attributes ......................... 148
    6.1.2 G-MKNN clustering on the Cloud ................................. 148
    6.1.3 Using MKNN for other meso-scale structures in graph ............. 149
## List of Tables

2.1 Results for synthetic dataset 2 ........................................ 52  
2.2 Real PPI datasets ....................................................... 53  
2.3 Real Dataset Results using SGD complexes as a gold standard 56  
2.4 Average Variance and Structural Density (\( \rho_s \)) comparison of  
G-MKNN vs MCL for real datasets ...................................... 56  

3.1 Comparison of clusters by CP-MKNN vs CP-Greedy on synthetic datasets .......................................................... 94  
3.2 Core Periphery structures by CP-MKNN vs CP-Greedy for Synthetic Datasets 2 and 3 ......................................................... 94  
3.3 Comparison of CP-MKNN (K=4) vs CP-Greedy for PPI datasets 97  
3.4 Core Periphery structures by CP-MKNN and CP-Greedy for PPI datasets ................................................................. 98  
3.5 Comparison of relationships of different types obtained for  
CP-MKNN (K=4) for Gavin Dataset .................................... 100  

4.1 ClusterONE vs ClusterONE-CP on synthetic datasets ........... 124
4.2 Comparison of overlapping clustering algorithms on Zachary's Karate club dataset ........................................ 128

5.1 Comparison of post-SGMKNN (K=4) and pre-SGMKNN (K=4) using MIPS complexes as a gold standard ........... 143
List of Figures

2.1 Clustering on a weighted graph using different notions of density ........................................ 18
2.2 Synthetic Graph 2 ................................................................. 21
2.3 Clusters obtained by MCODE on Synthetic dataset 2 ....... 23
2.4 Clusters obtained by MCL on Synthetic dataset 2 ........... 24
2.5 Clusters obtained by Louvain’s Algorithm on Synthetic dataset 2 .................................. 26
2.6 Description of the concept of Mutual K-nearest neighbors (MKNN) ........................................ 29
2.7 Flowchart to find MKNNs in a graph ................................. 31
2.8 Preliminary Clusters obtained by G-MKNN on Synthetic dataset 2 ........................................ 35
2.9 Clusters obtained by G-MKNN on Synthetic dataset 2 ....... 46
2.10 Variation of Average Variance with K for synthetic dataset 2 ............................................. 57
2.11 Variation of Average $\rho_s$ with K for synthetic dataset 2 ............................................. 57
2.12 Variation of Num. clusters with K for synthetic dataset 2 ............................................. 57
3.14 A dense core by CP-MKNN (K=2) (blue nodes) and CP-Greedy (blue and red nodes) on *Les Miserables*, a movie co-appearance dataset .................................................. 104

3.15 Variation in SF crime dataset across time and space ........ 106

3.16 Variation in SF dataset across 39 different crime types .... 107

3.17 Crime density contour plot for two crime types on a map of SF 108

3.18 Core Periphery structures for the year 2005

**Graph Details**

No. of nodes: 37

Structural Density: 0.88

Standard Deviation: 0.27

Clustering Coefficient: 0.943 ............................... 109

3.19 Core Periphery structures for the year 2010

**Graph Details**

No. of nodes: 38

Structural Density: 0.95

Standard Deviation: 0.245

Clustering Coefficient: 0.962 ............................... 110
3.20 Core Periphery structures for the year 2015

**Graph Details**

- No. of nodes: 37
- Structural Density: 0.76
- Standard Deviation: 0.31
- Clustering Coefficient: 0.86

3.21 Robustness of CP-MKNN with changing K for SF crime dataset

4.1 Overlapping clusters obtained by CPM on Synthetic dataset 3 (K=3)

4.2 Overlapping clusters obtained by ClusterONE on Synthetic dataset 3

4.3 Overlapping clusters obtained by OG-MKNN on Synthetic dataset 3 (K=4)

4.4 Overlapping clusters obtained by CPM (K=3) on Zachary's Karate club dataset

4.5 Overlapping clusters obtained by ClusterONE on Zachary's Karate club dataset

4.6 Overlapping clusters obtained by OG-MKNN (K=4) on Zachary's Karate club dataset

5.1 Different shapes of modules in Yeast PPI network

5.2 Block Diagram for Pre-processing based SG-MKNN

5.3 Block Diagram for Post-processing based SG-MKNN
Chapter 1

Introduction

Many real-world systems can be described as complex networks, i.e., assemblies of nodes and edges having nontrivial topological properties [1] [2]. Examples include social networks like acquaintance networks and co-authorship networks of scientific collaborations, biological networks like metabolic networks, food webs and protein-protein interaction networks, and technological networks like the internet, the World Wide Web and power grids. Analysis of such complex networks can help in gaining deeper insights into the features of such networks.

Graphs serve as a very good mathematical model for representing network structures. For example, in a graph representing a social network, nodes could be people or group of people, and edges could represent some form of interaction between the people. Further, the edges could have a weight associated with them, representing the strength of interaction between nodes in the graphs. Graphs can be described using a variety of
local, global and intermediate-scale (meso-scale) structures. Local structures are the ones described at the local scale of nodes and edges (e.g. node degree) while global structures mostly define summary statistics (e.g. network diameter). Meso-scale structures can help in the discovery of network patterns which are not easily evident at the local or the global scale [3].

One widely used meso-scale structure for analyzing complex networks is community detection or the discovery of community structures in the network. In lay man terms, a community can be defined as a subgroup of nodes in the network which share similar properties. Finding communities in a network provides a means of classification of nodes into groups and is of great practical significance in different domains. For example, finding communities of protein interactions in a PPI network can provide us with a vast variety of information about the molecular processes occurring in the cell.

More formally, a community structure in a network is a subset of nodes within the graph such that connections between the nodes inside the subset are denser than the connections with the rest of the network [4]. Thus, the problem of community detection in a network is synonymous to the problem of finding densely connected sub graphs in the network. In the graph theoretic domain, this problem is called as graph clustering.

Graph Clustering is an interesting and challenging problem with a lot of active research being done in this area [5]. Given that complex networks
have non-trivial topological properties, many algorithms use this as a criterion for designing graph clustering algorithms (e.g., modularity based algorithms [4] and clustering coefficient based algorithms [6]). In many domains, complex networks are embedded with a useful information about edge weights, in the form of distance, cost, capacity, bit rate etc. For example, in coauthorship networks, edge weights may correspond to the number of papers coauthored by two authors, or in protein-protein interaction networks, edge weights can correspond to the number of times two proteins are observed together in purification experiments. Such information could be leveraged along with network topology to find clusters of different shapes. In this dissertation, we design and implement novel clustering strategies for the analysis of community structures in weighted complex networks. Next we describe these clustering strategies briefly.

The first clustering algorithm that we develop is called as Graph-Mutual K Nearest Neighbor Clustering or G-MKNN. G-MKNN aims at finding mutually exclusive density based clusters in weighted and undirected graphs. G-MKNN uses a node-affinity measure called Mutual K-nearest neighbors (MKNN) to aid the process of clustering. It makes use of both, the topological properties of the nodes in the graph as well as edge weights as its criteria for clustering. We define a new notion of density called SE-density to achieve a balance between structural density and edge-weight homogeneity. Clusters thus obtained achieve a balance between cohesive intra-cluster structure and homogeneous edge weights. This type of clustering
has a wide utility in many real world networks where edge weights are equally important to the topological network structure. A popular example is co-authorship networks, where edge weights depicted by the number of co-authored papers are important deciders of cluster membership. The concept of MKNN has been successfully used for clustering of data points [7]. Here we extend this measure to the problem of clustering graphs.

Overlapping Graph-Mutual K Nearest Neighbor Clustering or OG-MKNN is the second graph clustering algorithm that we propose to solve. OG-MKNN aims at finding overlapping density based clusters in weighted and undirected graphs. Overlapping clusters have many practical applications since clusters in many real world domains tend to overlap with each other. For example, in protein-protein interaction networks, many protein complexes overlap with each other due to the multi-functional nature of many proteins. OG-MKNN is an extension of the G-MKNN algorithm with overlapping decisions incorporated into the algorithm. These decisions include (1) identification of the nodes which may be candidates for being shared among clusters, (2) the actual decision whether to share a node between two clusters, based upon the connectivity of the node with each cluster.

Functional information guided semi-supervised clustering (SG-MKNN) is the third graph clustering problem that we propose to solve. This clustering algorithm is especially targeted towards clustering protein-protein interaction networks based upon domain knowledge of Gene to Gene Ontology (GO) term relationships. Gene Ontology is a dynamic controlled vo-
cabulary consisting of three independent ontologies: GO Molecular Function, GO Biological Process and GO Cellular Component [8]. A GO term provides a description of the molecular functions, biological processes and sub-cellular locations that can be attributed to gene products. Our approach is based upon the fact that interacting proteins in a protein complex get enriched with the same or similar biological functions and therefore, should be related to similar GO terms [9][10]. Based upon this principle, we combine the information available from GO annotations into the G-MKNN algorithm and use it as a guide towards protein module formation.

As the second type of meso-scale structure in complex network, we investigate core periphery structures in graphs. Formally, the first definition of core periphery structures was given by Borgatti and Everett as a cohesive connected dense core, loosely connected by a sparse periphery [11]. It has been demonstrated though experimental results in [12] that core periphery structures arise naturally in networks due to presence of overlapping communities. For example, in a social network, a core group of close friends could be surrounded by a sparser periphery. In this dissertation, we provide with a definition of core-periphery structures in weighted graphs. Next, We develop two algorithms to discover core-periphery structures in weighted graphs based upon the topological properties as well as edge weight density of nodes. We further develop a methodology to score and categorize the identified core periphery structures.

As the first core periphery formulation algorithm, we develop a greedy
growth based algorithm to identify cores surrounded by sparse peripheries. This algorithm is based upon a clustering algorithm called ClusterONE [13]. As the second algorithm, we build an MKNN based algorithm called CP-MKNN to discover core-periphery structures in weighted graphs. We compare and contrast the core periphery structures obtained by the two algorithms using synthetic datasets and real world networks.

Each of the four problems described above have their own background and relevant literature. We briefly describe it for each problem separately in the following sections.

1.1 Graph-Mutual K-Nearest Neighbor (G-MKNN) Clustering

Many traditional graph clustering approaches are based upon using the topological properties of the graph to make clustering decisions. Modularity based approaches such as Newman’s clustering algorithm [4] and clustering coefficient based approaches such as MCODE [6] are a classic example of such clustering methodologies. For example, MCODE algorithm is based upon using a variant of clustering coefficient to capture cliquish structures in the graph. This approach fail to discover clusters which may be connected through homogeneous edge-weights in a non-clique structure. Recently, several versions of weighted clustering coefficients have been proposed in the literature for graph clustering. Kalna et al. provide a
survey of such variants in [14].

We have developed a graph clustering algorithm called Graph-Mutual K-Nearest Neighbors (G-MKNN) which makes use of both the topological properties of the nodes in the graph as well as edge weights as criteria for clustering. The algorithm is based upon a node affinity measure called MKNN which helps to achieve the goal of discovering cohesively connected clusters with uniform edge weights. We develop an effective two phase algorithm which makes efficient decisions for combining structure and edge weights in the final clustering. In the first phase, the algorithm identifies small sized core clusters containing nodes at relatively same level of SE-density. This helps in separating out the very dense clusters of the graph from their sparser neighborhoods. These small sized dense clusters are then iteratively merged and expanded in the second phase of clustering to obtain the final clusters.

Using synthetic and real world datasets, we show that G-MKNN clearly outperforms the unweighted graph clustering algorithm MCODE. We also compare the performance of G-MKNN with two well-known weighted graph clustering algorithms, namely, MCL [15] and Louvain’s algorithm [16]. MCL proposed by Van Dongen, is a random walk based graph clustering algorithm. It involves finding dense regions in a graph by simulating the process of flow diffusion or random walks in the graph. Using different measures of evaluation, we demonstrate that the clusters obtain by G-MKNN are much more balanced in terms of structure as well as edge-label
based density than MCL and Louvain’s algorithm. Further, since G-MKNN identifies clusters with nodes at the same level of density, it assigns a cluster label to each and every node in the graph. G-MKNN is thus a complete clustering algorithm. MCODE, MCL and Louvain’s may be good at identifying high density clusters; however, the nodes in low density areas are not assigned any cluster label, thereby leading to an incomplete clustering.

1.2 Discovery of core-periphery structures using MKNN

Community structure is the most popularly studied meso-scale structure in the literature. There are however, numerous other important types of meso-scale structures that exist in networks and which deserve attention of the research community. In this problem, we seek to study a different type of meso-structure in weighted graphs, namely, core-periphery structure.

The model of core-periphery structures in a network was first formalized by Borgatti and Everett in 2000 [11]. According to their model, a network contains a core-periphery structure if it contains a core set in which members are cohesively connected to each other and a periphery set, in which members are loosely connected to the core set. They developed two genetic algorithm based approaches for detecting both discrete and continuous versions of core-periphery structures. Boyd et al. [17] proposed the use of Kernighan-Lin algorithm to partition social networks into core and
periphery sets. Their algorithm was further enhanced by Luo et al. [18] to identify k-plex cores in protein interaction networks. Core periphery structures can be very useful in many real-world networks, such as, web, social, information and protein-protein interaction networks. Further, scoring and categorizing of peripheries can benefit in studying the flow of information in a network. Cores can be seen as dense entities connected to each other through sparse peripheries. Extracting such core-core relationships can be very useful in network analysis. For example, in a social network, core groups of close friends will be connected to each other through sparsely connected acquaintances. In such a network, core periphery structures can be used to study how trends get established and spread across the network.

In our work, we develop two algorithms to find core-periphery structures in weighted graphs. First, we formalize the definition of core-periphery structures for weighted graphs. Next, we develop a greedy growth based algorithm called CP-Greedy to identify cores and peripheries in weighted graphs. Further, we develop an MKNN based algorithm, called CP-MKNN, for the discovery of core-periphery structures in weighted graphs. Both the algorithms are followed by a methodology to score and categorize the core-periphery structures. Using both synthetic and real-world PPI graphs, we demonstrate the significance of identifying these structures in graphs. We further use a spatio-temporal dataset of crimes in San Francisco to demonstrate the differences in the two approaches that we develop.
1.3 Overlapping Graph-Mutual K-Nearest Neighbor (OG-MKNN) Clustering

Most of the research in the area of graph clustering has been focused on finding a disjoint set of clusters [19], [20], [21], [22]. Even the G-MKNN graph clustering algorithm that we develop in the first problem involves finding a set of mutually exclusive clusters in a weighted network. In [23] and [24], the authors show that community overlap is a significant feature of many real world networks. For example, in a social networks, individuals could belong to more than one social group like family, friends, colleagues. In a biological networks, proteins could be responsible for multiple functionalities in the cell and thus they may be a part of more than one modules.

For other applications which show that overlapping clusters are more suitable than non-overlapping clusters, see overlapping clustering for protein-protein interaction networks [25][13], clustering for distributed computing [26][27] and inherent multi-assignment clustering [28].

We develop an overlapping graph clustering algorithm for weighted networks based upon the principle of MKNN. Similar to G-MKNN, OG-MKNN is also built in two phases. The first phase involves building preliminary clusters with homogeneous structural and edge weight density. It is to be noted that these preliminary clusters contain mutually exclusive members.
In the second phase, we define measures to quantify the level of connectivity among the preliminary clusters. Clusters having connectivity above a certain threshold are merged together. However, for the pair of clusters where connectivity is below the threshold, we identify nodes which can be shared between the clusters. This process is repeated in an iterative fashion until we exhaust all the clusters.

We evaluate the performance of OG-MKNN algorithm using a synthetic and a real world social network dataset. We perform a comparison of the results obtained against two well-known overlapping clustering algorithms for weighted networks. The first algorithm is called Clique Percolation Method (CPM) [29]. The second algorithm that we plan to use for comparison analysis is ClusterONE [13], a greedy growth based overlapping clustering algorithm.

### 1.4 Functional information guided G-MKNN clustering for PPI networks

A substantial amount of research has been done to cluster protein-protein interaction networks incorporating domain knowledge in the form of functional information along with the network topology.

King et al. [30] proposed a graph clustering algorithm for weighted PPI networks called Restricted Neighborhood Search Clustering (RNSC). It is two step algorithm designed so as to combine Gene Ontology informa-
tion along with the topological information to obtain the final clustering. In the first phase, a clustering is done on the PPI network using random partitioning and diversification moves. In the second phase, a p-value is calculated for each cluster using GO annotations to calculate a measure of functional homogeneity of clusters. Clusters above a certain p-value are then discarded to obtain the final set of predicted clusters. Li et al. [31] proposed a clustering algorithm based upon hub removal to identify dense sub-graphs in the PPI networks. Local cliques are then identified from these dense sub-graphs and highly overlapping cliques are then merged to obtain clusters. As the final step, a functional reliability score is assigned to each cluster based upon GO annotations. Clusters below a certain threshold of the reliability score are discarded. Both these approaches use functional annotation scores as a final step to filter out clusters which are not functionally homogeneous. We propose an alternate strategy to incorporate this functional information early enough in the cluster growth process to guide towards the merging of preliminary clusters.

Our approach involves using functional annotations to build a semi-supervised clustering algorithm. We propose to build a two phase algorithm for this problem. Similar to G-MKNN, the first phase will use the notion of MKNN to determine preliminary clusters in the graph. We will then perform enrichment analysis on these preliminary clusters. The enrichment results will then be used to guide the merging process of preliminary clusters in phase 2.
Using real world PPI datasets, we compare the performance of the semi-supervised clustering algorithm with our base algorithm G-MKNN. By doing so, we will demonstrate the advantage of utilizing the functional information into the clustering process.

1.5 Conclusion

As the size and complexity of real world networks keeps growing, there is a need for developing advanced graph algorithms to help uncover hidden patterns in these networks. We believe that our contributions in this dissertation will help towards a better understanding of real world complex networks. From a broader perspective, in this dissertation, we develop algorithms to find community and core-periphery structures to analyze weighted and undirected networks. G-MKNN aims at identifying mutually-exclusive clusters in weighted and undirected graphs. OG-MKNN extends G-MKNN to find overlapping clusters. We further develop two algorithms to find core-periphery structures in weighted graphs to aid to the understanding of the intricacies of the networks. The first one being a greedy-growth based algorithm, and the second one, an MKNN based algorithm, called CP-MKNN, We demonstrate the effectiveness of core-periphery structures by using several synthetic and real world network datasets. Finally, we develop a domain knowledge based semi-supervised clustering algorithm for PPI networks. This functional information guided G-MKNN is a clustering
algorithm targeted specifically towards finding clusters for weighted PPI networks using GO-term based functional information.

1.6 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 describes the Graph-Mutual K Nearest Neighbor clustering algorithm (G-MKNN) along with its comparison with other graph clustering algorithms. Chapter 3 describes our developed approaches for discovering core-periphery structures in a weighted graph (CP-MKNN and CP-Greedy). We describe our results on synthetic and real datasets for this problem. Chapter 4 presents our solution for clustering weighted graphs into overlapping clusters (OG-MKNN). In chapter 5, we describe our semi-supervised clustering approach (SG-MKNN) for protein-protein interaction networks using GO term based functional information.
Chapter 2

Graph-Mutual K-Nearest Neighbor Clustering (G-MKNN)

This chapter presents the first graph clustering algorithm that we have developed, namely Graph-Mutual K-Nearest Neighbor clustering (G-MKNN). The corresponding experimental evaluation of the algorithm and comparison with other state-of-the-art graph clustering algorithms is also included in this chapter.

We consider an undirected, weighted graph $G = (V, E)$ where $V$ is the set of vertices or nodes and $E$ is the set of edges. $G$ can be represented by a symmetric adjacency matrix $A = [a_{ij}]$. The adjacency $a_{ij}$ between nodes $i$ and $j$ represents the strength of the connection or similarity between the two nodes and is a non-negative real number between 0 and 1. The diagonal elements are set equal to 0 as to represent to the algorithm that there are no self-loops in the network.
We propose G-MKNN as a graph clustering algorithm to partition a weighted graph $G$ into $k$ disjoint sub graphs $G_i = (V_i, E_i)$, where $V = \bigcup_{i=1}^{k} V_i$ and $V_i \cap V_j = \emptyset$ for any $i \neq j$. The desired properties of clustering are to obtain clusters such that (1) Nodes within clusters have high structural similarity within them compared to their neighborhoods, and (2) edge weights within a cluster have homogeneous values. GMKNN involves two main design concerns: (1) Defining a node affinity measure to capture our notion of density based upon network structure and edge weights, and (2) Designing a clustering algorithm based upon the developed node affinity measure. We discuss these two issues in detail in the sections below.

2.1 Background/Motivation

A desired clustering of a weighted graph should achieve a good balance between the following two properties: (1) Nodes within a cluster should be cohesively connected to each other in terms of structure (network topology), while inter-cluster topological connectivity should be minimized; and (2) The weights of edges connecting nodes inside a cluster should be homogeneous, or in other words, should have a low variance. A purely modularity based clustering approach uses a notion of density based solely on the first property, for example, [4]. This leads to clique like cohesive structures as clusters. We call this measure of density as structure-based density. On the other hand, most of the traditional relational data clustering ap-
proaches measure similarity or distance between nodes using only the second property based on edge weights. In such clustering algorithms, edges are defined between all pair of data points and the whole set of data points can be considered a big clique. Thus the concept of structure based density doesn’t work in this case. Dense clusters in this case correspond to a chunk of data points with a concentration of homogeneous edge weights with a high value of average similarity. We call such a notion of density as Edge-label based density.

To illustrate the concept of two different types of densities, we consider for example a synthetic undirected and weighted graph of 25 nodes with some embedded communities shown in Figure 2.1a. We call this graph Synthetic 1. We hypothetically partition the graph into clusters using the two different concepts of density.

**Structure-based density.** Figure 2.1b shows clustering based upon the network topology/structure information, i.e., vertex connectivity and neighborhood similarity. We can see that cliques are very well captured as modules. However, the edge labels inside clusters can have varying values in some cases.

**Edge label-based density.** Figure 2.1c shows the clustering results for edge label-based density. The edge labels within clusters are quite homogeneous. However, the clusters have a rather loose intra-cluster structure.

**Structure/Edge-label based density.** A yet another notion of density strives to achieve a balance between structure based density and edge
weight based density. We call this notion of density as Structure/Edge label-based density or SE-based density in short. Figure 2.1d shows a graph clustering result based upon maximizing SE-based density. Clusters thus obtained have a balance between structural similarity and edge weight homogeneity of the nodes within the cluster.

We propose G-MKNN as an SE-density based graph clustering algorithm for weighted and undirected networks. The goal of our clustering algorithm is to partition a weighted graph into cohesively connected clusters with nearly homogeneous edge weights. The problem is quite challenging as structure and edge weight based similarities are independent of each
other. We use a node affinity measure called as Mutual K-nearest neighbor (MKNN) [7] to capture the edge weight similarities. MKNN defines a two way relationship between a set of nodes. It is because of this two way relationship, that MKNN connects nodes at relatively the same level of SE-density. MKNN has been successfully used in [7] for clustering data points. We extend an adaptation of this same concept to graphs.

Graph clustering algorithm using MKNN can be performed in essentially two phases. In the first phase, the algorithm identifies small sized preliminary clusters where each cluster contains nodes at relatively the same level of SE-density. This helps in separating out the very dense cores of the graph from their sparser neighborhoods. These dense core clusters are then iteratively merged and expanded in the second phase of clustering to obtain the final clusters. Through extensive experimental evaluation, we show that clustering performed in this manner obtains a balance between structure based density and edge label-based density in the resultant clusters. Before delving into the details of G-MKNN clustering algorithm, we present some related work in the next section.

2.2 Related Work

Before we begin our discussion of related literature on graph clustering, consider the synthetic weighted graph 2 in Figure 2.2. This graph has 39 nodes and 74 edges and has been built so as to include three dense
communities embedded in it. The width of the edges in the graph has been drawn proportional to the weight of the edges. The three communities embedded in synthetic graph 2 are as below:

- Community 1: (R, S, T, U, V);
- Community 2: (A, B, C, D, E, F, G);

As evident from the graphical representation of synthetic dataset 2, Community 1 is a 3-clique. Communities 2 and 3 are not fully connected; however, they still form a densely connected set of nodes with homogeneous edge weights and cohesive network topology. We will use this dataset to illustrate the properties of different graph clustering algorithms discussed in this section and to compare and contrast the working of different algorithms.

2.2.1 Molecular COmplex DEtection (MCODE)

MCODE, proposed by Bader and Hogue [6] is a density based graph clustering algorithm for clustering un-weighted graphs. It is one of the first graph clustering algorithms which aimed towards finding complexes in protein-protein interaction networks. The algorithm works in two phases, vertex weighing, and molecular complex prediction. In the first phase, each of the
Figure 2.2: Synthetic Graph 2
vertices in the graph is weighted based upon the density of its neighborhood. Here, the neighborhood density is measured based upon the connectivity level as measured by a special type of clustering coefficient called core-clustering coefficient. Core-clustering coefficient of a vertex \( v \) is defined to be the density of the highest \( k \)-core in the immediate neighborhood of \( v \). A highest \( k \)-core, in turn is defined to be the most densely connected subgraph of minimal degree \( k \) in the neighborhood of \( v \). In the second phase, nodes with the highest weight are recursively set as seeds and the complexes are extended outward from the seeds, including vertices with weights above a certain threshold. This finishes the cluster formation process. There is also an optional post-processing phase where vertices are filtered out or added to the complexes formed in the second phase.

MCODE has three parameters, namely, node score cutoff, fluff and haircut. These parameters help to tweak the shape and size of modules generated by the algorithm. The node score cutoff parameter controls the entry of new nodes in a growing cluster. The fluff parameter on the other hand, is used to filter out some vertices in the clusters which lie on the periphery.

Figure 2.3 shows clusters obtained by MCODE on the synthetic graph 2. As we can see, MCODE captures clique like clusters (R, S, T, U, V) and (D, E, F, G) really well. However other types of dense clusters are not identified. Moreover, some nodes in the graph are not assigned any cluster label at all.
2.2.2 Markov Clustering Algorithm (MCL)

Van Dongen [15] proposed a random walk based graph clustering algorithm called Markov Clustering algorithm (MCL). MCL is based upon simulating the process of flow diffusion in the graph to find dense regions in a graph. MCL uses two operations called Inflation and Expansion in order to manipulate the adjacency matrix. These two operations control the random walks or flow in the graph. The inflation process corresponds to taking a Hadamard power of the matrix followed by a diagonal scaling, while the expansion process coincides with taking a power of the matrix using normal matrix product. Both these mathematical operations have the effect of enhancing the intra-cluster probabilities and lowering the inter-cluster probabilities. This makes the flow thicker in the dense regions (or inside the clusters) and thinner in sparse regions (or in between the clusters).
After a few iterations of these two operations, a stable matrix is obtained which has some remarkable properties. The connected components in the graph described by this matrix correspond to non-overlapping clusters in the original graph. The MCL algorithm is very scalable and easy to implement. The code of MCL can be obtained from [32].

Figure 2.4 shows the clusters obtained by MCL on the synthetic dataset 2. MCL, like MCODE captures cliquish structures very well, like (R, S, T, U, V) and (D, E, F, G, J). However, the edge label-based density in rest of the clusters obtained is not uniform. It does assign a cluster label to more number of nodes than MCODE. However, it still leaves out some nodes without any cluster label assigned.
2.2.3 Louvain’s Method for Community Detection

Louvain’s method [16] is a greedy optimization method that is based upon optimizing the modularity of a partition of the network. Modularity is a metric that was proposed by Newman and Girvan in [4]. This metric quantifies the strength of a community assignment by comparing how dense the connections are inside the community versus they would be in a particular type of random network. Mathematically, Modularity, Q is defined as

\[ Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{k_i k_j}{2m}\right] \delta(c_i, c_j) \tag{2.1} \]

Here A is the Adjacency matrix. \( k_i = \sum_j A_{ij} \) is the total link weight penetrating node i. and \( m = \frac{1}{2} \sum_{ij} A_{ij} \) is the total link weight the network. The Kronecker delta \( \delta(c_i, c_j) \) is 1 when nodes i and j belong to the same community and 0 otherwise. The term \( A_{ij} - \frac{k_i k_j}{2m} \) measures how strongly nodes i and j are connected in the real network, as compared to how strongly we would expect them to connected in a random network of the type described above.

The Louvain’s method proceeds with looking for small sized communities which optimize modularity locally. Second, it aggregates nodes belonging to the same community to build a meta network among the community nodes. These steps are iteratively repeated until a maximum of modularity is obtained.

Figure 2.5 shows the clusters obtained by Louvain’s algorithm on the
Figure 2.5: Clusters obtained by Louvain’s Algorithm on Synthetic dataset 2

synthetic dataset 2. This algorithm assigns a cluster label to all nodes in the graph. However, some obtained clusters (like (W, X, Y, AJ, AK, AL, AH)) are not balanced in terms of edge weight homogeneity and cohesive network topology.

2.3 Preliminary Concepts

We describe some preliminary concepts in this section before moving on to the description of the G-MKNN clustering algorithm. More specifically, here we describe the properties of the node similarity matrix and the node affinity measure used in the G-MKNN algorithm.
2.3.1 Node Similarity Matrix

A graph adjacency matrix defines edge weights between vertices which are connected to each other, while non-neighbors have a zero edge weight between them. We define an augmented version of the adjacency matrix, \( A \), called as similarity matrix to define similarities between vertices of the graph which are up to four hops away. We use the concept of transitive similarity to do so. Formally, we define the similarity matrix and associated terms below.

**Definition 2.1.** (Similarity Matrix (SM) \( | V | \times | V | \)) \( SM \) defines an edge weight between all vertices which are connected by paths of up to four hops in \( G \). This is obtained using Dijkstra’s algorithm in the local four hop neighborhood of each node.

Similarity defined in the above defined manner follows the triangle inequality and is a metric.

**Definition 2.2.** [Augmented Graph \( G_a = (V, E_a) \)] We call the graph represented by the similarity matrix \( SM \) as an augmented graph \( G_a \). Such a graph will have a weighted edge between any two vertices which are up to four hops away in \( G \).

**Definition 2.3.** [Primary similarity] An edge \( e_{ij} \in E_a \) between any two vertices \( \in V_a \) is said to be a primary similarity if \( a_{ij} \neq 0 \). In other words, any edge in graph \( G_a \) is said to be primary if the same edge also exists in \( G \).
Definition 2.4. [Secondary similarity] An edge $e_{ij} \in E_a$ between two vertices is said to be a secondary similarity if $a_{ij} = 0$ and $s_m_{ij} \neq 0$. In other words, any edge in graph $G_a$ is said to be secondary if this edge does not exist in the original graph $G$. This edge came into existence only because of the graph augmentation process described above.

We use the term similarity from now on to point to either primary or secondary similarity.

2.3.2 Node Affinity Measure: Mutual K-nearest neighbors

The similarity matrix provides a concise graph representation, however, by itself, it is insufficient for clustering. We define a node affinity measure called Mutual K-nearest neighbor relationship on the augmented graph $G_a$ based upon the node similarity values. As opposed to the traditional neighborhood affinity measure KNN, MKNN defines a two way relationship between nodes. We explain the concept of MKNN using figure 2.6 for $K=4$, where $K$ defines the maximum number of neighbors possible. The four mutual nearest neighbors for the node $G$ are nodes $A$, $B$, $E$ and $D$. However, $G$ is not among the four nearest neighbors of nodes $A$, $B$, $E$ and $D$. In other words, the relationship is not reciprocated by nodes $A$, $B$, $E$ and $D$. However, nodes $G$ and $F$ become mutual K nearest neighbors (MKNN) of each other because all the nodes in the dense cluster in the middle refuse to have a MKNN relationship with either of them.

For relatively low values of $K$, MKNN based upon SM helps to connect
vertices at approximately the same level of edge-label based density. Imposing some further constraints on the clustering algorithm to take the structural similarity between nodes into account, clustering based upon MKNN can achieve a balance between edge-label based density and structure-based density. We will see further in the algorithm that MKNN relationships can also be established on a matrix which represents closeness between nodes, and not just on SM. Formally we define MKNN relationship as follows.

**Definition 2.5.** [Mutual K-nearest neighbor] Two nodes $i$ and $j$ are MK-Nearest neighbors if either (1) in the Euclidian space between them, there are fewer than $K$ other nodes or (2) there are more than $K$ nodes between them, but many of these nodes have found mutual K-NNs amongst themselves, and thus refuse to be mutual K-NNs with any other nodes, thereby leading to at most $K-1$ nodes in the intervening Euclidean space that have not found their mutual K-NNs.
MK-nearest neighbors are calculated in two steps:

1. All node pairs \((v_i, v_j)\) in \(G_a\), such that \(sm_{ij} \neq 0\) are sorted in descending order of their similarity value.

2. MKNN relationships are then set by a top-bottom scan of the table of similarities built in (1).

MKNN relationships thus formed are mutual, i.e., if \(v_i\) is an MKNN of \(v_j\), then \(v_j\) must be an MKNN of \(v_i\). Figure 2.7 shows a flowchart to calculate Mutual K-nearest neighbors for nodes in a graph.

Graph clustering based upon the idea of MKNN begins with forming relatively SE-dense clusters by merging \textit{clusterinitiators} with their mutual K-nearest neighbors together. The clusters thus formed lie approximately at the same level of SE density and are usually small in size. We call this phase of clustering as the preliminary phase since it generates preliminary clusters as its output. These preliminary clusters thus formed may be quite close to each other in terms of structural similarity or edge label similarity. Therefore, the next phase of clustering involves iteratively merging the preliminary clusters until no more merging is possible. We call this as the merge phase of clustering. In the next section, we discuss the different phases involved in the G-MKNN clustering algorithm in detail.
2.4 Clustering Algorithm

The G-MKNN clustering algorithm is based upon partitioning a weighted graph taking into account both structural as well as edge weight similarity. It uses a node affinity measure called MKNN as described in the previous section to achieve SE-dense clusters. The algorithm begins with two initialization steps:

1. Given an input graph G, it augments G to form an augmented graph $G_a$ as per definition 2.2.

2. Mutual K-nearest neighbor relationships are then defined amongst all the nodes of $G_a$ as per definition 2.5 (K being a user input).
After these two initialization steps, the algorithm essentially works in two phases. The first phase builds candidate clusters by merging MKNN neighbors together. The second phase performs merging of relatively close candidate clusters to obtain the final clusters. We describe the two phases of clustering in detail below. We will use the synthetic graph 2 as described previously in figure 2.2 to describe the different phases of the algorithm.

2.4.1 Clustering Phase 1: Preliminary Phase

The preliminary phase of clustering aims at partitioning the graph into small sized SE-dense sub graphs. This is achieved by merging MKNN neighbors (which lie at approximately the same level of edge-label density) together in the order of decreasing structural density of nodes. We describe the preliminary phase in detail after defining what we mean by an i-node.

**Definition 2.6.** [i-node] We refer to vertices or nodes in the preliminary phase of clustering as individual nodes or i-nodes. For example, in figure 2.2, each independent node A, B, C etc. corresponds to an i-node. In phase 1, i-nodes merge with each other based upon the MKNN relationships amongst each other.

All MKNN partners of an i-node lie approximately at the same edge similarity level from the i-node. Therefore, it makes sense to merge these i-nodes together to form preliminary clusters with almost similar edge weights. However, this doesn’t ensure that the cluster would also be good as per the
structural density constraint as well. In order to account for this problem, we introduce a term called radius to rank the i-nodes in a decreasing order of SE-density surrounding them.

**Definition 2.7.** \([\text{Radius}, \delta]\) Given an i-node \(i_n\) with its similarities from all its MK-nearest neighbors being \((sm_1, sm_2, \ldots sm_p)\). Let the degrees of MKNNs be \((d_1, d_2, \ldots d_q)\). The radius of \(i_n\) is defined as:

\[
\delta_{i_n} = \frac{\sum_{i=1}^{p} d_i \cdot sm_i}{p \cdot d_{avg}}
\]  

Here \(p \leq K\) is the number of MK-nearest neighbors of \(i_n\) and \(d_{avg}\) is the average degree of all i-nodes in the graph. Intuitively, radius gives a measure of SE density surrounding an i-node. An i-node which is strongly connected to its MKNN neighbors, which in turn have a high degree will have a high value of radius. Further, such an i-node should be given the first chance to merge with its MKNN neighbors. We call this merging order of i-nodes defined by the descending order of their radius values as the cluster initiator order. This order ensures that i-nodes which are densest as per SE-density get the first chance to merge their surroundings into one cluster. We formally define a cluster initiator as follows.

**Definition 2.8.** \([\text{ClusterInitiator}\] A cluster initiator is an i-node which starts the process of cluster formation by including all its MK-Nearest neighbors in to one cluster.

Thus, i-nodes, begin merging with their MKNN neighbors in the cluster
initiator order. This ensures that the i-nodes which are densest as per the SE density merge with their MKNN neighbors first to obtain a good balance between the structural and edge-label based density in the preliminary cluster thus formed. For example, in figure 2.2, i-node D has the highest value of radius and is the first chosen cluster initiator to add its MKNNs A, E, F and G into one preliminary cluster. Thus, \((A, D, E, F, G)\) becomes one preliminary cluster. Let us call this cluster as \(p1\). The next best cluster initiator in line is i-node C. Its MKNN neighbors are B, E, G and A. C adds i-node B into its cluster. However, before adding i-nodes E, G and A into its cluster, it notes that these three i-nodes were already added by the first cluster initiator D into the preliminary cluster \(p1\). For all such i-nodes, which are MKNN neighbors of two cluster initiators, we merge them with the cluster initiator where they will cause the least increase in standard deviation upon addition to the preliminary cluster. In this case, i-nodes E, F and G remain in \(p1\). Thus, the cluster initiator C forms a preliminary cluster with i-nodes \((B, C)\). We call this preliminary cluster as \(p2\). It is to be noted that i-nodes like A, E, F, G and B, which get added by some cluster initiator are not allowed to become a cluster initiator themselves. However, they may be strongly connected to some of their own MKNN neighbors. For this reason, if an MKNN neighbor \(n_i\) is added by some cluster initiator \(c_i\), we go one level deep to check if the MKNN neighbors of \(n_i\) are good candidates for merging into the cluster initiated by \(c_i\). We check that if any MKNN of \(n_i\) upon addition to \(c_i\)'s cluster, does not increase the standard deviation of the cluster beyond
a bound called as SDBound, then we add it to \( c_i \)'s cluster. We use a value of five percent for SDBound. This small value for the bound ensures that the structural and edge label density of the cluster remains homogeneous.

This process continues until we exhaust the list of cluster initiators. This finishes the preliminary phase of clustering. The clusters obtained after the preliminary phase for synthetic graph 2 have been shown in figure 2.8. Further, the pseudocode for this phase is given in algorithm 2.1.

**2.4.2 Clustering Phase 2: Merge Phase**

The preliminary phase of clustering generates SE-dense subgraphs which have strong intra-cluster similarity. However, depending upon the size of the graph, inter-cluster similarities could still be high. The merge phase (1)
Algorithm 2.1: Preliminary Phase of Clustering

Input:

\( IP \): List of i-nodes \( \in V \)

\( MKNN \): MKNN relation matrix

\( SM \): Similarity Matrix

Output: \( C[1..n] \): Cluster Label Array \( C \) labeling the preliminary clusters for each i-node

1. for each i-node \( i \in IP \) in decreasing order of Radius do
2.  if cluster label \( C_i \) is not set for i-node \( i \) then
3.     Set \( C_i \) as the cluster initiator Set \( C_i \leftarrow \Theta \)
4.     for each i-node \( j \in MKNN \) of i-node \( i \) do
5.         if cluster label \( C_j \) is not set for i-node \( j \) then
6.             Set \( C_j \leftarrow \Theta \)
7.         else
8.             assign \( C_i \) as \( \Theta \) only if the standard deviation obtained of \( SM \) values in the resultant cluster is less than the standard deviation in i-node \( j \) current cluster
9.     end
10. end
11. end
12. end
13. end
identifies preliminary clusters with high levels of similarity and (2) merges these highly similar sub graphs together in an iterative fashion to obtain the final clusters. Before moving on to the detailed description of the merge phase, we introduce a term called \( c \)-node.

**Definition 2.9.** \([c\text{-node}]\) We refer to the clusters formed in preliminary phase as cluster-nodes or \( c \)-nodes. In the merge phase of clustering, \( c \)-nodes which are quite well-connected to each other are merged together. To give an example of \( c \)-nodes, the preliminary clusters \( p_1 \) (\( A, D, E, F, G \)) and \( p_2 \) (\( B, C \)) represented by blue and pink color respectively in figure 2.8, each represent an independent \( c \)-node. \( c \)-node \( p_1 \) is comprised of five constituent \( i \)-nodes, i.e., \( A, D, E, F \) and \( G \). Similarly, \( c \)-node \( p_2 \) is comprised of two \( i \)-nodes, i.e., \( B \) and \( C \).

There are three main challenges in this phase:

- How do we define a measure of well-connectedness between \( c \)-nodes so as to identify candidates for merging?

- How do we decide the order in which to merge the \( c \)-nodes? and

- How do we ensure that the iterative merging of clusters converges when the right number of clusters have been formed?

We discuss these three challenges in detail below.
2.4.2.1 Identifying c-nodes for merging

In order to identify the c-nodes which will serve as good candidates for merging, we first need to define a measure of well-connectedness or closeness among the c-nodes. We build a connectivity matrix defining such closeness value among the c-nodes. Before providing a formal definition for the connectivity matrix, we define some preliminary concepts used in the calculation of connectivity matrix.

Definition 2.10. [Internal similarity of c-node (Insim)] For a c-node \( cn_i \) \( \text{insim}(cn_i) \) captures the sum of all primary similarities among all the constituent i-nodes of \( cn_i \) (i.e., all \( in_i \in cn_i \)). Mathematically, \( \text{insim}(cn_i) \) is defined as:

\[
\text{insim}(cn_i) = \sum_{\substack{in_i \in cn_i \\in j \neq i}} A(in_i, in_j)
\]

(2.3)

Definition 2.11. [Cut of c-node (Cut)] For a c-node \( cn_i \) \( \text{cut}(cn_i) \) captures the sum of all primary similarities between the i-nodes belonging to \( cn_i \) (i.e., all \( in_i \in cn_i \)) and all the other i-nodes in the graph which may belong to other c-nodes. Mathematically, \( \text{cut}(cn_i) \) is defined as:
\[ \text{insim}(cn_i) = \sum_{i \in cn_i} \sum_{j \not\in cn_i, j \neq i} A(in_i, in_j) \]  
(2.4)

**Definition 2.12.** [Linkage between two c-nodes] Linkage\((cn_i, cn_j)\) between two c-nodes, \(cn_i\) and \(cn_j\) captures the sum of primary similarities of the connection between the i-nodes belonging to \(cn_i\) (i.e., all \(in_i \in cn_i\)) and the i-nodes belonging to \(cn_j\) (i.e., all \(in_j \in cn_j\)). Mathematically, linkage\((cn_i, cn_j)\) is defined as below:

\[ \text{linkage}(cn_i, cn_j) = \sum_{i \in cn_i} \sum_{j \in cn_j} A(in_i, in_j) \]  
(2.5)

**Definition 2.13.** [Connectivity Matrix (CM)] Given the number of c-nodes as \(n\), \(CM_{|n| \times |n|}\) defines a measure of similarity between all pairs of c-nodes. It is defined as follows:

\[ CM(cn_i, cn_j) = \frac{\text{Linkage}(cn_i, cn_j)}{\text{Cut}(cn_i)} + \frac{\text{Linkage}(cn_i, cn_j)}{\text{Cut}(cn_j)} \]  
(2.6)

Intuitively, CM captures the fact that how closely the two c-nodes are connected to each other than to the rest of the c-nodes in the graph. Further, it is to be noted that the dimensions of CM keep changing during the course.
of the iterations of the merge phase as c-nodes merge with each other to form bigger sized c-nodes, thereby reducing the total number of c-nodes, n.

Similar to the SM values for i-nodes in phase 1, CM values provide a good measure of connectivity between the c-nodes; however, on their own they are insufficient for clustering. Using CM values alone for merging would require considering every pair of c-nodes in the descending order of CM values as candidates for merging. In order to avoid this complexity, we again take the help of MKNN relationships to aid in the process of merging. However, the MKNN relationships here are defined between c-nodes as compared to those between i-nodes as in phase 1. Using connectivity matrix (CM) values as a measure of similarity between two c-nodes, we define MKNN relationships among the c-nodes. Similar to the case of i-nodes, calculation of MKNN neighbors is a two-step process: (1) Sort pairs of c-nodes \((cn_i, cn_j)\) in decreasing order of their CM value. (2) Scan the list of connectivity values from (1) in a top to bottom fashion and assign MKNN relationships.

Unlike the preliminary phase, not all MKNN pairs of c-nodes will be good candidates for merging. Before merging two MKNN neighbors, we perform the following two checks to ensure that the merged clusters maintain a good level of structural similarity and edge label similarity.

1. Check for Structural Similarity: c-nodes themselves are clusters with uniform level of structural similarity. However, when we merge two c-nodes, we need to make sure that the final merged cluster will also
be a cohesive unit as well. In order to do so, we make sure that each c-node’s participation in the merged cluster is strong. For this, we make use of a term called Cohesion to measure the strength of structural similarity inside a cluster as opposed to outgoing links.

**Definition 2.14.** [Cohesion, \( \eta \)] For a c-nodes \( cn_i \), cohesion represents how cohesively the i-nodes inside a cluster are connected to each other. Formally, it is given by:

\[
\eta(cn_i) = \frac{Insim(cn_i)}{Insim(cn_i) + Cut(cn_i)}
\]  

Two c-nodes pass the structural similarity check if the Cohesion of the prospective merged c-node is greater than the Cohesion of the initiator c-node.

2. Check for Edge-weight Similarity: The edge-weight similarity inside a c-node is approximately homogeneous. However, before merging two c-nodes, we need to make sure that the final cluster will also be homogenous as per edge label similarity. In order to do so, we make use of the difference between average primary edge similarity of the two c-nodes and their connection. For this, we define a term called cluster separation.

**Definition 2.15.** [Cluster Separation] Let \( cn_i \) and \( cn_j \) be two c-nodes which are MKNN neighbors of each other. Let \( \mu_i \) and \( \mu_j \) represent
the mean of primary similarities among \(i\)-nodes belonging to \(cn_i\) and amongst \(i\)-nodes belonging to \(cn_j\) respectively. Let \(\mu_k\) represent the mean of primary similarities connecting the \(i\)-nodes belonging to \(cn_i\) with the \(i\)-nodes belonging to \(cn_j\). Then, we say that \(cn_i\) and \(cn_j\) are closely separated in terms of edge label similarity if 

\[
|\mu_i - \mu_j| \leq MBOUND, |\mu_i - \mu_k| \leq MBOUND \text{ and } |\mu_j - \mu_k| \leq MBOUND.
\]

We set a value of 0.20 for MBOUND for all our runs on the synthetic and real datasets.

2.4.2.2 Order of merging \(c\)-nodes

c-nodes which satisfy the two constraints mentioned above for structural similarity and edge label similarity are good for being merged together. Now, the problem is to decide the order in which to merge the candidate c-nodes. The order of merging of c-nodes can greatly affect the final structural and edge label density of the clusters. In the merge phase, we use \textit{Cohesion} as defined in definition 2.14 to rank the c-nodes in the cluster initiator order.

We give c-nodes with a high cohesion value the first chance to choose amongst their MKNN neighbors for merging, before some other c-nodes with a relatively lower cohesion value chooses this c-node for merging. In figure 2.8, the c-node (R, S, T, U, V) has the best value of cohesion and thus, will be given the first chance to merge with its MKNN neighbors, (J, K, L, M, P, Z); (A, D, E, F, G); (B, C) and (AM, AN, AP, AQ, Q) in decreasing
order of CM values. However, none of the MKNN neighbors satisfy the constraints of structural similarity. Even though the neighbor c-node \((J, K, L, M, P, Z)\) satisfies the edge label-similarity constraint, it is rejected for merging because of not satisfying the structural constraints. The next best cluster initiator \((A, D, E, F, G)\) has MKNN neighbors \((B, C); (AM, AN, AP, AQ, Q); (J, K, L, M, P, Z)\) and \((R, S, T, U, V)\) in decreasing order of CM values. Here, out of the four MKNN neighbors, the cluster initiator only merges with the MKNN neighbor \((B, C)\) to form a cluster \((A, B, C, D, E, F, G)\). This merged cluster is again made a c-node to be used for merging in the next iteration of the merge phase.

2.4.2.3 Guaranteeing Convergence

Using the cluster initiator order defined by cohesion, we continue the process of merging until we exhaust the whole list of c-nodes. This finishes one iteration of the merge phase. The clusters thus obtained as a result of merging are again made as c-nodes and MKNN neighbors are defined among these new set of c-nodes to iteratively continue with the merging process. However, c-nodes which failed to satisfy either one of the criterion concerning structural or edge label similarity as mentioned before, are not merged together. Further, we save this merge decision, so that for the c-nodes which did not merge together, their children may not be made MKNN neighbors again in future iterations. Continuing with our example of synthetic graph 2 in figure 2.8, since c-nodes \((A, D, E, F, G)\) and \((AM,
AN, AP, AQ, Q) did not merge in the first iteration of merge phase, their descendent c-nodes (A, B, C, D, E, F, G) and (AM, AN, AP, AQ, Q, J, K, L, M, P, Z) will not be made MKNN neighbors in the future iterations of the merge phase. This makes sense because if two c-nodes, say $cn_i$ and $cn_j$ have decided that they are not fit for merging, then no matter how many new c-nodes get added to either $cn_i$ or $cn_j$, they still will remain unfit for merging.

This in turn, helps in guaranteeing the convergence of the merging phase. When all possible merging of c-nodes gets finished, we will reach a point where no c-node can find an MKNN neighbor to merge with. At this stage, we declare the set of c-nodes remaining as our final clusters. This finishes the merge phase. For synthetic graph 2, the final set of clusters is displayed in figure 2.9. Further, the pseudocode for the merge phase is given in algorithm 2.2.

### 2.5 Complexity Analysis

Suppose there are n nodes and m edges in the input graph $G$ for which clustering needs to be performed. We discuss the complexity analysis for each phase separately for graph $G$. 

Algorithm 2.2: Merge Phase of Clustering

**Input:**

*CP*: List of c-nodes from preliminary phase

*C* [1., n]: Cluster Label Array for i-nodes (preliminary clusters)

*SM*: Similarity Matrix

**Output:** *C* [1., n]: Updated Cluster Label Array C for i-nodes

1. Calculate $CM|_n \times |_n$ for c-nodes in CP.
2. Find MKNN neighbors for c-nodes using CM as metric.
3. Assign $D \leftarrow$ temporary cluster label array for c-nodes.
4. repeat
5. for each c-node $i \in CP$ in decreasing order of Cohesion do
6. if cluster label $D_i$ is not set for c-node $i$ then
7. Set c-nodes, as the cluster initiator
8. Set $D_i \leftarrow \Lambda$
9. $\Theta \leftarrow$ cluster label of i-nodes $\in$ c-node $i$
10. for each c-node $j \in MKNN$ of c-node $i$ do
11. if cluster label $D_j$ is not set for c-node $j$ then
12. if checks for structural similarity and edge label similarity are satisfied then
13. Set $D_j \leftarrow \Lambda$
14. Set $C$(i-node) $\leftarrow \Theta \forall$ i-node $j \in$ c-node $j$
15. end
16. next $\Lambda$
17. end
18. end
19. Update CP with new set of c-nodes and recalculate CM
20. until no merges take place for 3 successive iterations;
2.5.1 Complexity Analysis for initialization phase

The initialization phase of the G-MKNN algorithm involves augmenting the adjacency matrix of the graph to obtain a similarity matrix have edge labels defined between all pair of nodes up to four hops away. If the graph already provides weights among all the edges, then this step costs $O(1)$. Otherwise, this step involves using Dijkstra algorithm in local neighborhood of each node. Suppose the maximum degree of a node in the graph $G$ is $d$. Then the four hop neighborhood of a node will have $O(d^4)$ nodes in the worse case. Let us call this number of nodes in this small graph as $V_0$ and the number of edges in this graph as $E_0$. Then Dijkstra’s algorithm will have a worst case complexity of $O(E_0 \log V_0)$ in this small graph. Running Dijkstra’s algorithm for the local neighborhood of each node will give a total time
complexity of \( O(n \times E_0 \log V_0) \).

The next initialization step involves setting up MKNN relationships among the nodes in the augmented graph. This process involves sorting all the secondary edge similarities in decreasing order. In the worst case, the secondary similarities will be \( n^2 \). We use quick sort to sort all the secondary similarities. Therefore the average case complexity for this process of MKNN calculation is \( O(m \log m) \) and the worst case complexity is \( O(m^2) \).

### 2.5.2 Complexity Analysis for Phase 1

The phase 1 of G-MKNN algorithm involves merging all cluster initiators with their MKNN neighbors in the order of radius. Let \( K \) be the user defined parameter in MKNN. The worst case complexity of this phase is \( O(nK) \) as the algorithm must go through each node in order to merge it with its \( O(K) \) MKNN neighbors.

### 2.5.3 Complexity Analysis for Phase 2

Suppose the number of preliminary clusters formed in phase 1 is \( p \). The first iteration of phase 2 involves calculating a connectivity value among all pairs of these \( p \) clusters. Some of these \( p \) clusters merge together in this iteration leaving behind say, \( l \) unique clusters. We assume that roughly half of the clusters merge with each other, thus we can safely assume \( l \) to be equal to \( p/2 \). Thus the complexity of iteration will be \( O(p^2) \), that of
iteration 2 will be $O((p/2)^2)$ and so on. In the worst case, all clusters will merge with each other to form one big cluster. Thus the total complexity of phase 2 can calculated as the sum of the series $O(p^2 + (p/2)^2 + (p/4)^2 + \cdots + 1) \sim O(p^2 \log p)$.

2.6 Experimental Evaluation

We evaluate the performance of G-MKNN by comparing it with three representative graph clustering techniques as described before, namely Markov Cluster (MCL), Molecular Complex Detection (MCODE), and Louvain’s algorithm. We perform both a quantitative as well as a qualitative comparison of results. For the quantitative comparison, we use one synthetic dataset and four real world weighted protein-protein interaction (PPI) databases to evaluate our algorithm. Next, we describe the parameters used for different algorithms used for comparison.

2.6.1 Description of parameters used

For running MCODE, we used the latest version (1.32) of its cytoscape [33] plugin. MCODE has four parameters, depth limit, node score cutoff, haircut and fluff. For our runs on the synthetic as well as real datasets, we used depth limit=4 and the rest of the parameters at their default value, i.e., fluff=no, haircut=yes, node score cutoff = 0.2.

For the implementation of MCL, we used the Network Analysis Tools
(NeAT) web plugin [34]. MCL has a single parameter called inflation (Range: 1.2 to 5) which can be used to tune the granularity of the clusters. For the synthetic dataset, we kept its value to be $\lambda = 2$. For the real datasets, we tuned the inflation parameter as set in the ClusterONE paper [13], from where we have taken our real datasets. Louvain’s algorithm was implemented using its MATLAB version freely available at [35].

For implementing G-MKNN, we used Python 3.0. We set the value of $K=4$ for all the datasets. A small value of $K$ leads to relatively small sized preliminary clusters which tend to pure. Next we present the results of different algorithms on the synthetic as well as real datasets.

### 2.6.2 Quantitative Analysis

We evaluate the performance of the clustering algorithm G-MKNN using one synthetic and four real world datasets. Below we describe these results in detail.

#### 2.6.2.1 Synthetic Dataset

We present here the quantitative results for synthetic dataset 2, as presented in figure 2.2. We first discuss about the evaluation measures used followed by the results.

**2.6.2.1.1 Evaluation Measures** In order to assess the functioning of G-MKNN clustering algorithm on synthetic dataset 2, we have used two sets of
evaluation measures to test the clusters for their edge-label based density as well as structural density. The first measure that we use is involves finding variance (var) of primary similarity edges in the graph. These two quantities help to evaluate the edge label based density of a cluster. To capture clusters with homogenous levels of edge-similarity, low variance is desirable in the resultant clustering. Next, in order to assess that how well the clusters capture the structural similarity in the graph, we define a term call structural density ($\rho_s$).

**Definition 2.16.**

\[
StructuralDensity(\rho_s) = \frac{Number of edges in the cluster}{All possible number of edges in the cluster}
\]

A high value of $s$ for a cluster can be interpreted as high intra-cluster structural similarity.

### 2.6.2.1.2 Results

The clustering results for G-MKNN on synthetic dataset have been presented in figure 2.9. G-MKNN captures the three SE-dense clusters very efficiently as described below.

- Blue cluster: A, B, C, D, E, F, G
- Brown cluster: R, S, T, U, V
All these clusters have good structural density along with homogeneous edge weights. Other than these three SE-dense clusters, G-MKNN also finds three other clusters, which are not very structurally dense, but have homogeneous edge weights. These clusters are well separated from the very dense clusters, rather than merging them partially, as in the case other algorithms MCODE (2.3), MCL (2.4) and Louvain (2.5). These three clusters are listed below.

- Red cluster: W, X, Y
- Yellow cluster: J, K, L, N, H, AA
- Olive green cluster: P, Q, AM, AN, AP, AQ

Next, we present a quantitative comparison of the algorithms G-MKNN, MCL, Louvain and MCODE using the evaluation measures discussed in the previous section. These quantitative clustering results for synthetic dataset 2 are listed in table 2.1.

G-MKNN obtains the lowest value of average variance amongst all the clustering algorithms. This helps in capturing both structurally dense cliquish shaped clusters as well as non-clique structures in the graph with homogeneous edge-weight density. For example, the complete clique cluster R, S, T, U, V is found by all four algorithms perfectly. However, the non-clique dense cluster A, B, C, D, E, F is identified correctly only by G-MKNN.

The average structural density $\rho_s$ obtained for G-MKNN is more than Louvain algorithm and less than that obtained by MCL and MCODE. The
reason being that G-MKNN clustering criterion favors a balance between structural density and edge weight homogeneity. Algorithms like MCODE focus on achieving good structural density even at the cost of diluting the edge weight homogeneity of the clusters.

Louvain's algorithm assigns a cluster label to all the 39 nodes in the graph. MCODE algorithm and MCL algorithm assign cluster labels to 21 and 35 nodes respectively. G-MKNN is able to assign a cluster label to 38 out of 39 nodes. Further, it is to be noted that Louvain's algorithm finds the largest sized clusters of all algorithms. G-MKNN finds a mix of large and small sized clusters as guided by the density criterion.

### 2.6.2.2 Real World Datasets

We now describe the real PPI datasets and the evaluation measures that we have used, followed by the results. We use three large scale weighted yeast protein-protein interaction datasets, Collins [36], Gavin [9], and Krogan
Table 2.2: Real PPI datasets

<table>
<thead>
<tr>
<th></th>
<th>Collins</th>
<th>Gavin</th>
<th>Krogan</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Proteins</td>
<td>1622</td>
<td>1855</td>
<td>3672</td>
</tr>
<tr>
<td>No. Links</td>
<td>9074</td>
<td>7669</td>
<td>14317</td>
</tr>
</tbody>
</table>

database [37] to evaluate our algorithm. These datasets have been compiled in one place by the authors of ClusterONE [13], and we obtained these datasets from there. For details about these datasets, refer to [13]. We compare the set of predicted complexes obtained by G-MKNN and all the four comparison algorithms against a set of gold standard protein complexes. The gold standard has been derived from the gene ontology based annotations in the SGD [13]. We mention the properties of PPI networks that we have used in table 2.2.

2.6.2.2.1 Evaluation Measures In order to compare our predicted protein complexes with the gold standard set of complexes, we used Accuracy as introduced by Brohee and van Helden [38]. Accuracy is calculated as a geometric mean of two other measures namely, clustering-wise sensitivity ($S_n$) and clustering-wise positive predicted value (PPV). Accuracy is computed using the confusion matrix $T = [t_{ij}]$ of the complexes. Given $n$ as the number of predicted complexes and $m$ as the number of gold standard complexes, let $t_{ij}$ denote the number of proteins common between gold standard complex $i$ and predicted complex $j$. Let the total number of proteins belonging to gold standard complex $i$ be $m_i$. $S_n$ and PPV are defined
as follows.

\[ S_n = \frac{\sum_{i=1}^{m} \max_{j=1}^{n} t_{ij}}{\sum_{i=1}^{m} m_i} \quad PPV = \frac{\sum_{j=1}^{n} \max_{i=1}^{m} t_{ij}}{\sum_{j=1}^{n} \sum_{i=1}^{m} t_{ij}} \]  

(2.9)

The value of \( S_n \) will be maximized when all the proteins belong to one predicted complex, as it checks how well every protein in the gold standard complexes is found in the matching predicted complex. A high \( S_n \) value does not guarantee that the predicted complex contains only proteins belonging to the matched gold standard complex. On the other hand, PPV value will be highest when each protein belongs to its own cluster. Thus, to balance these two quantities, we use Accuracy as given below.

\[ Accuracy = \sqrt{S_n \times PPV} \]  

(2.10)

2.6.2.2.2 Results We present the \( S_n \), PPV and Accuracy (Acc.) values obtained by different algorithms SGD complexes in table 2.3. A high value of accuracy can be achieved in one of the following three ways

1. a high \( S_n \) and a low PPV value or by
2. a high PPV and low \( S_n \) value, or by
3. balanced \( S_n \) and PPV values.

In the current scenario of predicted PPI complexes, case (3) means that the predicted complexes capture the proteins in the gold standard com-
plexes well, with minimum extraneous noise proteins in the predicted complex. Thus, a desirable property of the predicted complexes is to achieve a balance between the \( S_n \) and PPV values.

As evident from the accuracy values, we can see that G-MKNN clearly outperforms MCODE which is purely based upon the structural definition of density. It also obtains a higher accuracy value than Louvain’s algorithm and almost similar accuracy as that of MCL for all the three datasets. However, when we look deeper at the \( S_n \) and PPV values, we see that G-MKNN obtains a better balance between \( S_n \) and PPV values. In order to further compare the performance of G-MKNN and MCL, we calculated the average variance and average structural density (\( \rho_s \)) values of the top 20 clusters obtained by both the algorithms on real datasets in table 2.2. These results are listed in table 2.4. As evident from the table, G-MKNN clearly obtains the lowest value of average variance for all the datasets. This difference is clearly evident for Krogan dataset where G-MKNN obtains both a lower variance as well as a higher value of average structural density, thereby obtaining balanced clusters.

### 2.6.3 Robustness Analysis of G-MKNN with changing K

In this section, we provide an analysis of the effect of changing the value of the parameter K in G-MKNN. We study the effect of changing K on average variance, average \( \rho_s \) and number of clusters obtained for synthetic dataset 2. This analysis has been presented in figures 2.10, 2.11 and 2.12.
Table 2.3: Real Dataset Results using SGD complexes as a gold standard

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>No. Clusters</th>
<th>Sn</th>
<th>PPV</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins</td>
<td>G-MKNN</td>
<td>130</td>
<td>0.46</td>
<td>0.54</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>115</td>
<td>0.44</td>
<td>0.58</td>
<td>0.50</td>
</tr>
<tr>
<td></td>
<td>Louvain</td>
<td>72</td>
<td>0.54</td>
<td>0.42</td>
<td>0.47</td>
</tr>
<tr>
<td></td>
<td>MCODE</td>
<td>79</td>
<td>0.39</td>
<td>0.55</td>
<td>0.46</td>
</tr>
<tr>
<td>Gavin</td>
<td>G-MKNN</td>
<td>188</td>
<td>0.44</td>
<td>0.55</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>163</td>
<td>0.40</td>
<td>0.61</td>
<td>0.49</td>
</tr>
<tr>
<td></td>
<td>Louvain</td>
<td>47</td>
<td>0.55</td>
<td>0.29</td>
<td>0.40</td>
</tr>
<tr>
<td></td>
<td>MCODE</td>
<td>110</td>
<td>0.38</td>
<td>0.57</td>
<td>0.47</td>
</tr>
<tr>
<td>Krogan</td>
<td>G-MKNN</td>
<td>267</td>
<td>0.47</td>
<td>0.55</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>339</td>
<td>0.46</td>
<td>0.56</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>Louvain</td>
<td>34</td>
<td>0.63</td>
<td>0.21</td>
<td>0.36</td>
</tr>
<tr>
<td></td>
<td>MCODE</td>
<td>91</td>
<td>0.36</td>
<td>0.27</td>
<td>0.31</td>
</tr>
</tbody>
</table>

Table 2.4: Average Variance and Structural Density ($\rho_s$) comparison of G-MKNN vs MCL for real datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Algorithm</th>
<th>Avg. Variance (top 20 clusters)</th>
<th>Avg.$\rho_s$ (top 20 clusters)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collins</td>
<td>G-MKNN</td>
<td>0.016</td>
<td>0.72</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>0.021</td>
<td>0.71</td>
</tr>
<tr>
<td>Gavin</td>
<td>G-MKNN</td>
<td>0.014</td>
<td>0.44</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>0.016</td>
<td>0.76</td>
</tr>
<tr>
<td>Krogan</td>
<td>G-MKNN</td>
<td>0.064</td>
<td>0.48</td>
</tr>
<tr>
<td></td>
<td>MCL</td>
<td>0.092</td>
<td>0.22</td>
</tr>
</tbody>
</table>
Figure 2.10: Variation of Average Variance with K for synthetic dataset 2

Figure 2.11: Variation of Average $\rho_s$ with K for synthetic dataset 2

Figure 2.12: Variation of Num. clusters with K for synthetic dataset 2
Average variance remains fairly constant for low values of K and begins to increase beyond a certain threshold, in this case K=6. Average structural density and number of clusters vary much lesser with the changing value of K. We suggest experimenting with relatively small values of K when using G-MKNN.

2.6.4 Conclusion

In this chapter, we have presented a weighted graph clustering algorithm G-MKNN, which aims to achieve a balance between structural and edge-label based density. A node-affinity measure called MKNN has been used to build a two phase iterative clustering algorithm. The first phase builds preliminary clusters which are then iteratively merged together in the second merge phase. The algorithm converges when no more merges are possible among the mutual k-nearest neighbors. Experimental results on the real world datasets show that we obtain comparable accuracy results with state-of-the art clustering algorithms like MCL and significantly better results as compared with Louvain's algorithm and MCODE. Further, on comparison of the top few predicted complexes by G-MKNN with MCL, we find that G-MKNN obtains low variance clusters, thereby capturing clique as well as non-clique structures in the graph.
Chapter 3

Discovery of Core Periphery structures in weighted graphs

In this chapter, we present two algorithms that we have developed to find core periphery structures in weighted graphs. The first one is a greedy growth algorithm based upon an overlapping clustering algorithm called ClusterONE [13]. The second one is an MKNN based algorithm called Core Periphery-MKNN (CP-MKNN). We formally present our definition of core periphery structures for weighted graphs. We also develop methodologies to score and categorize the identified core periphery structures. Using synthetic and real world graphs, we illustrate the advantages of studying these structures in weighted graphs. We further provide a quantitative and qualitative comparison between the two core periphery algorithms that we have developed.
3.1 Background

Community structure is a widely studied meso-scale property in complex networks. The detection of community structures in complex networks conveys useful information about the understanding of such networks [39]. In Chapter 2, we presented a graph clustering algorithm to find community structures in weighted graphs. We demonstrated that these structures can provide useful insights into the study of real world complex networks by taking the example of protein-protein interaction networks. In this chapter, we focus our attention to a yet another important meso-scale property of complex networks, called as the core periphery structure.

The traditional graph clustering techniques focus on finding clusters or cohesive subgroups in a network. Community structure might not always be the best choice for understanding the meso-scale properties of a complex network. Let us consider the example of hubs in complex networks, or nodes with very high degree. It has been widely studied that such hubs occur in most real world complex networks [N2003]. Such hubs are often connected to many other nodes lying in different parts of the network, and thus, can have strong ties with several different communities. Thus, these hub nodes may be assigned to different communities depending upon the algorithm used for community detection. An overlapping clustering algorithm may assign multi-cluster membership to these hub nodes. In such a scenario, considering hub nodes to be part of a core in a core periphery
structure might make more sense in understanding the network. Further, there might be nodes which do not have membership in any of the cohesive subgroups. It is possible that these nodes which do not belong to any cohesive subgroup might be sufficiently well connected (As per structure density or edge weight density) to one or more of the communities so as to be considered as peripheral nodes surrounding the communities. Another membership model for nodes in a graph is a core periphery model, where core clusters consist of dense cohesive subgroups, surrounded by one or more loosely connected peripheries. Such a model can reveal interesting information about the interconnections or interdependencies between different subgroups in the network.

As per the traditional definition of a core-periphery structure given by Borgatti and Everett [11], a core periphery structure consists of two sets of nodes, a dense, cohesive core set and a sparse, unconnected periphery set. A core set is the one in which members are cohesively connected to each other, and a periphery set is the one in which the members are loosely connected to the core set. While Borgatti and Everett’s model supports only one core periphery structure in a network, they further extended the definition of core periphery structure to allow for weak connections among the periphery nodes [40]. Thereby, allowing a graph to have multiple core periphery structures occurring simultaneously in the network. Figure 3.1 shows an example of a core periphery structure in a graph as per this traditional definition. The blue and yellow node sets corresponds to the cohe-
Figure 3.1: Example Core Periphery Structures in a Weighted Graph

sively connected core sets. The red node set represents the periphery set. It is to be noted that the nodes in the periphery set are loosely connected to the members of the core set.

The analysis of core periphery structures and their relationships can be useful in many domains. In PPI networks, core periphery structures can correspond to a static set of proteins surrounded by a dynamic periphery set of proteins loosely connected to the core set [18]. Another interesting application of core periphery structures is in the area of social networks. For example, in a social network of movie fans, cores may be formed by comedy fans, action fans, etc. The periphery nodes for such core sets may correspond to those people who recently joined the community, or those people who like very few movies in the corresponding genre. Periphery nodes belonging to two core genre sets at the same time might represent people who for example, loosely like both, comedy and action movies. Availability of such
core periphery information can be of great importance to e-commerce websites like Netflix, Amazon, etc. Some other examples where core periphery structures have been used include the study of national elites and collective action [41] and the study of proximity among Japanese monkeys [42]. In [12], Yang et al. demonstrate that core periphery structures arise naturally in many domains due to the presence of overlapping communities.

In this chapter, we present our definition of core periphery structures which is more suited to weighted graphs. Next, we describe the two algorithms that we have developed for identifying core periphery structures in weighted graphs with extensive comparison analysis between the two approaches. Further, we also explore the type and scores corresponding to peripheries. We study the importance of core periphery structures in real world PPI networks, social networks and a spatio-temporal network of crimes in San Francisco.

3.2 Related Work

Borgatti and Everett were the first ones to formalize the concept of core periphery structures in a network [11]. According to their model, a network contains a core periphery structure if it contains a core set in which members are cohesively connected to each other and a periphery set, in which members are loosely connected to the core set. They developed two genetic algorithm based approaches for detecting both discrete and
continuous versions of core periphery structures. However, their approach divides the whole network into a single core periphery structure. They further extended their work in [40] to find multiple core periphery structures in graphs. Here, they further relax their definition of core periphery structure to allow for some low density connections among the periphery nodes. Boyd et al.

Boyd et al. [17] proposed the use of Kernighan-Lin algorithm to partition social networks into core and periphery sets. Their algorithm was further enhanced by Luo et al. [18] to identify k-plex cores in protein interaction networks. A random walk model has been used by Rossa et al. [43] to assign a coreness value to nodes in a network. Yang and Leskovec [12] developed an algorithm to find overlapping clusters and core periphery structures using a network model of overlapping tiles. In [44], a graph modification based approach has been used to find core periphery structures in graphs.

Core periphery structures are useful for studying relationships between different subgroups. However, traditionally, overlapping clustering has been used for finding relationships among clusters in the network. Several overlapping clusters have been developed in the research community [29], [45], [46]. In [12], Yang and Leskovec demonstrate using several real world examples that core periphery structures arise naturally as a result of overlapping nature of communities. Our first core periphery identification algorithm is based upon an overlapping clustering algorithm called clus-
3.2.1 Clustering with Overlapping Neighborhood Expansion (ClusterONE)

A weighted and overlapping graph clustering algorithm called Clustering with Overlapping Neighborhood Expansion (ClusterONE) has been recently proposed by Nepusz et al. [13]. Just like MCODE, ClusterONE is based upon the greedy growth of vertex seeds. It uses a measure of cohesiveness to decide when a group of nodes is connected well enough to form an independent cluster of their own. This cohesiveness measure is calculated based upon the weight of the edges inside the clusters as compared to the weight of the edges from nodes inside the cluster to the rest of the nodes in the graph. After the greedy growth process stops for a cluster, a new seed is chosen and the growth process starts again. This process is repeated until no more seeds remains. There is also a second step in the algorithm where it identifies highly overlapping groups of clusters to merge them into a single cluster.

ClusterONE is capable of finding not only clique like structures in the graph, but also other density connected structures. Figure 3.2 shows the clusters obtained by ClusterONE on the synthetic graph 2 (As described in figure 2.2). Clusters (R, S, T, U, V); (A, B, C, D, E, F, G); and (AA, AB, AC, AD, AE, AF, AG, AH, AJ, AL) captured by ClusterONE coincide almost completely with the communities embedded in synthetic graph 2.
ClusterONE ensures that the final clusters are cohesively connected in terms of structural density, however, the edge labels inside some clusters may not be homogeneous. For example, even though clusters (W, X, Y, AJ, AK, AL, AH) and (A, B, C, AM, AN, AP) look structurally dense, they do not have a uniform edge label density. Moreover, the algorithm ClusterONE may not assign a cluster label to all the nodes in the graph. In other words, it is an incomplete graph clustering algorithm.

3.3 Our Approach

In this work, we develop two algorithms to find core periphery structures in weighted graphs. Both the algorithms take into account both the structural as well as edge weight density based properties in weighted graphs. Along with finding these core periphery structures, we also develop
aa methodology to score and categorize peripheries. We first provide our
definition for core periphery structures in a weighted graph before moving
on to our approaches for finding these structures.

3.3.1 Formal Definitions

A very simplistic approach to find core periphery structures in a graph
would be to (1) first find cohesive subgroups in the graph using any graph
clustering approach and then (2) defining peripheries as all the nodes that
are adjacent to at least one member of the core. However, this approach
will limit the peripheries to be in the 1-hop neighborhood of the cores. For
practical purposes, we would also want to include nodes which lie at cer-
tain hop distance from the core as periphery nodes, introducing the notion
of periphery cluster. This allows for some low density connections among
the periphery nodes as well. Below we provide our formal definition of a
core periphery structure for a weighted graph.

Definition 3.1. [Core Periphery Structure] We define a core periphery struc-
ture \((C \cup P)\) in a weighted graph as an organization of core cluster \(C\) and
periphery cluster \(P\), such that

1. Nodes \(\in C\) are cohesively connected to each other.

2. Nodes \(\in P\) are loosely connected to the nodes \(\in C\). The periphery
   connections are less dense than the core connections, either in terms
   of structure, or in terms of edge weight density. Figure 3.3 shows two
peripheries for the center core. Type 1 periphery differs from the core mainly in terms of structural density. Type two periphery differs from the core mainly in terms of edge weight density.

A graph can contain more than one core periphery structures. Further, a subset can have the role of both core and periphery. Next, we provide a formal definition of different types of peripheries.

**Definition 3.2.** [Sibling Relationship] We define a sibling relationship as a relationship between two core periphery structures \((C_i, P_i)\) and \((C_j, P_j)\) that arises either if \(P_i = P_j\) or if a weak periphery-periphery relationship exists between the peripheries \(P_i\) and \(P_j\).

**Definition 3.3.** [Core Periphery Types] Each core periphery relationship \((C_i, P_j)\) is given a type (1 or 2) based upon the density difference between core and periphery. Let \(E\) be the edgeset connecting the core \(C_i\) and periph-
1. Type 1 periphery: This type of periphery is less dense than the core cluster as per structure density. Formally, Cohesion($C_i$) is greater than the Cohesion of the merged set ($C_i \cup E \cup P_j$). The structural density measure Cohesion was defined formally in definition 2.14.

2. Type 2 periphery: This type of periphery is less dense than the core cluster as per edge weight density. Formally, $\mu(C_i) - \mu(P_j) > m$, or $\mu(C_i) - \mu(E) > m$. Here $m$ is $MEAN OFFSET$, a user defined parameter with a default value of 0.2.

**Definition 3.4.** [Core Periphery Structure Distance (CPDistance)] A core periphery structure distance is defined based upon how many standard deviations away is the mean of periphery edges from the core edges.

$$CPDistance(C_i, P_j) = \left| \frac{\mu(w_{in}(C_i)) - \mu(w_{in}(P_j \cup E))}{\sigma(w_{in}(C_i))} \right|$$

Here $E$ corresponds to the edgeset connecting $C_i$ to $P_j$. Intuitively, this score captures how many standard deviations away is the periphery from the core cluster’s mean. This score gives a measure of edge weight density similarity between the core and the periphery. A lower value of CPDistance signifies a strong edge weight density similarity between the core and the periphery.
**Definition 3.5.** [Core Periphery Structure Score (CPScore)] A core periphery structure Score is defined as the structural density \( \rho_s \) of the core edges combined with periphery edges and the edges connecting the two sets.

\[
CPScore(C_i, P_j) = \rho_s(C_i \cup E \cup P_j)
\] (3.2)

Here \( E \) corresponds to the edgeset connecting \( C_i \) to \( P_j \). \( \rho_s \) or structural density has been defined in definition 2.16. Intuitively, this score captures the structural density of the hypothetical cluster formed by merging the core and periphery together. This score gives a measure of structural similarity of periphery with the core. A higher value of CPScore signifies a strong structural connection between the core and periphery.

For the algorithm description, we will use the synthetic graph 2. This graph has 39 nodes and 74 edges and was also used in Chapter 2 in figure 2.2. The width of the edges has been drawn proportional to the edge weights.

We have embedded three cores surrounded by multiple peripheries in the graph. The three cores are listed below:

- Core 1: (A, B, C, D, E, F, G)
- Core 2: (AA, AB, AC, AD, AE, AF, AG, AH, AJ, AK, AL)
- Core 3: (R, S, T, U, V).
### 3.3.2 CP-Greedy: A Greedy Core Periphery Algorithm

We develop a greedy growth based algorithm to find core periphery structures in a weighted graph. This algorithm is based upon an overlapping clustering algorithm called ClusterONE. ClusterONE only uses a measure of structural density to called Cohesion to make clustering decisions. We also make use of edge weight density in the form of mean of edges to identify core periphery structures. ClusterONE defines Cohesion $C$ for a cluster as below.

$$Cohesion(C) = \left( \frac{\sum(w_{in}(C))}{\sum(w_{in}(C)) + \sum(w_{out}(C)) + p \times |C|} \right)$$  \hspace{1cm} (3.3)

Here $w_{in}$ denotes the internal edge weights $\in C$ and $w_{out}$ denotes the weight of outgoing edges from $C$. $p \times |C|$ is a penalty term used to model the existence of yet undiscovered edges. We define a user specified parameter called $MEAN.OFFSET$ as the minimum allowable difference in edge weight density (mean) of core and periphery clusters. A value of 0.2 is used as a default value for $MEAN.OFFSET$. The algorithm operates in four phases.

#### 3.3.2.1 Initialization Phase

The initialization phase involves ordering the nodes in the graph as per their weighted degree. This order defines the cluster initiator order, or the
order in which nodes initiate cluster formation in the next phase.

3.3.2.2 Greedy Growth Phase

In the greedy growth phase, clusters are allowed to grow from cluster initiator seed nodes as long as the structural density measure, \( \text{Cohesion} \) of the cluster keeps increasing. This process might involve node addition or node deletion steps, whichever leads to a greater increase in \( \text{Cohesion} \). A growing seed stops when its cluster becomes a locally optimal cohesive group as per the \( \text{Cohesion} \) measure. At this point, nodes surrounding this locally optimal cluster are assigned to boundary nodesets to facilitate the formation of core periphery clusters in the later phases of the algorithm. This process of boundary nodeset formation is explained next.

3.3.2.2.1 Boundary nodeset formation

All external nodes incident on a locally optimal cluster, say \( L_i \), are checked for possible addition to one of the boundary nodesets \( \text{inrange, low or high} \). These nodesets are defined next.

i) Boundary Nodeset \( \text{inrange} \): All external nodes of \( L_i \), which do not satisfy the structural density constraint, cohesion for \( L_i \), but majority of their adjacent edges to \( L_i \) are within \( \text{MEAN}_{\text{OFFSET}} \) away from \( \text{mean}(w_{in}(L_i)) \).

ii) Boundary Nodeset \( \text{low} \): All external nodes of \( L_i \), which do not satisfy the structural density constraint cohesion for \( L_i \), and majority of
iii) Boundary Nodeset high: All external nodes of $L_i$, which do not satisfy the structural density constraint cohesion for $L_i$, and majority of their adjacent edges to $L_i$ have greater edge weight than (mean($w_{in}(L_i)$) + $MEAN OFFSET$).

In figure 3.4, nodes A, B and C are examples of boundary nodes of type $inrange$, $low$ and $high$ respectively for the center blue cluster.

3.3.2.2.2 Using Boundary nodesets to grow peripheries around cores

Growth phase involves choosing cluster initiators which have not yet been
assigned a cluster and using them for cluster growth. If a cluster initiator, say $L_i$, adds a node $n$ which belongs to the boundary nodeset $low$ of a previously formed cluster, say $L_{prev}$, then growth phase makes sure to not consider any node $\in L_{prev}$ for further growth of $L_i$’s cluster. This helps in growing $L_i$’s cluster around $L_{prev}$’s cluster, possibly leading to a core periphery relationship between $L_{prev}$ and $L_i$.

For example, in figure 3.4, if node B initiates cluster formation, it will avoid the addition of blue nodes to its cluster as B lies in the blue cluster’s boundary nodeset $low$. Instead, it will consider the addition of all red neighboring nodes to its cluster. This helps the prospective periphery clusters to grow around the neighboring core (blue cluster), rather than merging partially with the core.

The steps for growth phase and for extracting boundary nodesets are described in algorithms 3.1 and 3.2 respectively.

### 3.3.2.3 Overlap Phase

The growth phase results in locally optimal cohesive overlapping clusters. Merge phase merges pairs of clusters from the growth phase with an overlap score greater than an overlap threshold ($= 0.5$). The overlap score between two clusters $C_i$ and $C_j$ is calculated as

$$OverlapScore(C_i, C_j) = \frac{|C_i \cap C_j|^2}{|C_i| \times |C_j|}$$  \hspace{1cm} (3.4)
Algorithm 3.1: Greedy Growth Phase

Input:

- \( SM \): Similarity matrix for graph \( G = (V, E) \)
- \( P \): List of nodes
- \( NODE\_PENALTY \): (for calculating cohesion)

Output:

- Cluster label set \( S \) for each node
- Boundary nodesets \( \text{inrange, low, high} \) for each cluster

1. Sort nodelist \( P \) in decreasing order of weighted degrees
2. Initialize cluster label \( \theta \)
3. for each node \( p_i \in P \) do
   4. if \( S(p_i) \) empty for \( p_i \) then
      5. add \( \theta \) to \( S(p_i) \)
      6. Cluster nodeset \( C_n \leftarrow p_i \)
      7. cohesion\_bound \leftarrow cohesion(\( C_n \))
      8. for \( p_j \in \text{external nodes incident on at least one boundary edge of } C_n \) and \( p_i \notin \text{exclude\_set} \) do
         9. if cohesion\((C_n \cup p_j) > \text{cohesion\_bound}\) then
            10. chosen\_node \leftarrow p_j
            11. cohesion\_bound \leftarrow cohesion\((C_n \cup p_j)\)
            12. add\_flag \leftarrow true
            13. delete\_flag \leftarrow false
      end
   end
5. for \( p_j \in \text{internal nodes of } C_n \) incident on at least one boundary edge of \( C_n \) do
   6. if cohesion\((C_n \setminus p_j) > \text{cohesion\_bound}\) then
      7. chosen\_node \leftarrow p_j
      8. cohesion\_bound \leftarrow cohesion\((C_n \setminus p_j)\)
      9. add\_flag \leftarrow false delete\_flag \leftarrow true
   end
7. if add\_flag is true then
   8. \( C_n \leftarrow C_n \cup \text{chosen\_node} \)
   9. For all sets \( C_n' \) s.t. \( \text{chosen\_node} \) is in boundary nodeset \( \text{inrange} \) of \( C_n' \), add all nodes \( \in C_n' \) to \( \text{exclude\_set} \)
   10. else if delete\_flag is true then
      11. \( C_n \leftarrow C_n \setminus \text{chosen\_node} \)
      12. For all sets \( C_n' \) s.t. \( \text{chosen\_node} \) is in boundary nodeset \( \text{inrange} \) of \( C_n' \), remove all nodes \( \in C_n' \) from \( \text{exclude\_set} \).
   13. else
      14. Declare \( C_n \) as a locally optimal cluster
      15. Generate \( \text{boundary\_nodesets}(C_n) \)
   end
end
Algorithm 3.2: Generate Boundary Nodesets

Input:

- $C_n$: locally optimal cohesive cluster nodeset
- $SM$: similarity matrix
- $MEAN OFFSET$: user defined parameter

Output:

Boundary nodesets $inrange, low$ and $high$ for $C_n$

for $p_j \in$ external nodes of $C_n$ incident on at least one boundary edge do

1. $E \leftarrow$ Set of edges connecting $p_j$ and $C_n$
2. $lbound = \mu(C_n) - MEAN OFFSET$
3. $ubound = \mu(C_n) + MEAN OFFSET$
4. $n_{short} \leftarrow$ no. edges in $E < lbound$
5. $n_{long} \leftarrow$ no. edges in $E > ubound$
6. $n_{inrange} \leftarrow$ no. edges in $E \geq lbound$ and $\leq ubound$
7. if $n_{inrange} \geq n_{short} + n_{long}$ then
   1. boundary nodeset $inrange \leftarrow$ boundary nodeset $inrange \cup \{p_j\}$
   1. end
8. else if $n_{short} \geq n_{inrange} + n_{long}$ then
   1. boundary nodeset $low \leftarrow$ boundary nodeset $low \cup \{p_j\}$
   1. end
9. else
   1. boundary nodeset $high \leftarrow$ boundary nodeset $high \cup \{p_j\}$
   1. end
10. end
3.3.2.4 Extraction of Core Periphery associations

Growth phase leads to cluster formation in the order of their density due to its choice of cluster initiators as per weighted degree of nodes. Therefore, it makes sense to form periphery relations for cores in the order of their formation in the growth phase. Once the overlap phase is finished, all clusters, \( O_i \), are traversed in the order in which they were formed to form periphery relationships. Specifically, each cluster \( O_i \) checks for nodes in its boundary nodesets \textit{inrange} and \textit{low}. These nodes should also have been assigned a cluster label, say, \( P_j \) in the growth phase. Core periphery relationships are formed between \( O_i \) and \( P_j \) if one of the following two conditions are specified.

- Mean of edges in \( O_i \) \( \geq \) Mean of edges in \( P_j \)  
  \( (P_j \) being a \textit{low} boundary cluster).

- Cohesion(\( O_i \)) \( \geq \) cohesion(\( P_j \))  
  \( (P_j \) being a \textit{low} inrange boundary cluster).

Each core periphery relationship \((O_i, P_j)\) is given a type (1 or 2) based upon the majority nodeset type in the periphery cluster. Type 1 peripheries have majority of nodes belonging to nodeset \textit{inrange}, while type 2 peripheries have majority of the nodes belonging to nodeset \textit{low}. Intuitively, Type 1 peripheries are lower in structural density than their cores, while type 2 are lower in edge weight density than the core.
Finally, each core periphery relationship is scored as per definitions 3.4 and 3.5. An example core periphery structure found by CP-Greedy on synthetic dataset 2 is demonstrated in figure 3.5. Algorithm 3.3 describes the steps for extracting core periphery associations.

3.3.3 CP-MKNN : An MKNN based Core Periphery Algorithm

In chapter 2, we described a graph clustering algorithm using the concept of Mutual K-Nearest Neighbors (MKNN). In this section, we leverage the concept of MKNN to find core periphery structures in weighted graphs. Similar to G-MKNN clustering algorithm, CP-MKNN is built in multiple phases.
Algorithm 3.3: Extract Core-Periphery Structures

Input:

\( S_n \): Set containing cluster nodesets generated after merge phase

Output:

\( C_P \): Data structure containing core-periphery associations \((C, P)\) along with their type and score

1. Initialize \( C_P \) as an empty data structure.
2. Store in \( O_n \) the ordered merged clusters in \( S_n \) by the order in which their constituent clusters were generated in the greedy growth phase
3. for \( O_i \in O_n \) do
4. Set \( P \) to cluster nodesets to which nodes \( \in O_i \)'s boundary nodeset in range, low belong
5. for \( P_j \in P \) do
6. if \((P_j \in \text{low and } \mu(O_i) > \mu(P_j)) \) or \((P_j \in \text{inrange and } \text{cohesion}(O_i) > \text{cohesion}(P_j))\) then
7. Form \((O_i, P_j)\) as a core-periphery association
8. Set Type\((O_i, P_j)\) as the type of majority of boundary nodes \( \in P_j \) (Type 1: inrange, Type 2: low)
9. Let \( E \) be the edgeset connecting \( O_i \) and \( P_j \)
10. \( \text{CPDistance} \ (O_i, P_j) \leftarrow \frac{||\mu(\text{edge set } O_i) - \mu(\text{edge set } P_j \cup E)||}{\sigma(\text{edge set } O_i)} \)
11. Add \((O_i, P_j)\) along with its type and CPDistance to \( C_P \)
12. end
13. end
14. end
3.3.3.1 Initialization Phase

The initialization phase for CP-MKNN is the same as described previously for G-MKNN. The graph is augmented to form secondary edges among nodes upto 4 hops away. Then, MKNN relationships are formed among the nodes.

3.3.3.2 CP-MKNN Phase 1

The phase 1 for CP-MKNN is the same as for G-MKNN as described in previous chapter. In G-MKNN, the output of phase 1 yields small sized preliminary clusters at approximately the same level of SE-density. These preliminary clusters are called as \textit{c-nodes} in phase 2. In figure 3.6, we show the preliminary clusters obtained by CP-MKNN on synthetic graph 2 after phase 1. Here each color represents a different preliminary cluster. It is to be noted that at this time, these clusters are not yet categorized as cores or peripheries. The only characteristic feature of these small sized clusters is that the MKNN heuristic capture nodes at the same level of SE-density together. The pseudocode for this phase is given in algorithm 2.1 in chapter 2.

3.3.3.3 CP-MKNN Phase 2 (Merge and Boundary Formation)

The preliminary clusters thus obtained after phase 1 have a property that the nodes in each cluster lie at approximately homogeneous SE-density.
Figure 3.6: Preliminary Clusters obtained by CP-MKNN on Synthetic dataset 2
These preliminary clusters from phase 1 are called as c-nodes in this phase. The phase 2 of CP-MKNN identifies core periphery structures amongst these preliminary clusters in an iterative fashion. The formation of core periphery structures involves identifying c-node pairs for merging and boundary formation. Specifically, it involves

- Identification of the c-node pairs which can be merged together.

- Identification of c-nodes lying in the boundary/neighborhood of core c-nodes which are eligible to be peripheries to the c-nodes.

- Further, the identification of sibling relationships among the c-nodes requires the identification of weak connections amongst periphery c-nodes.

Next, we describe the steps involved in CP phase 2 in detail.

1. Similar to the phase 2 for G-MKNN, CP-MKNN involves constructing a connectivity matrix (CM) as a measure of similarity among all pairs of c-nodes. This similarity metric is defined in terms of Linkage and Cut between c-nodes. These terms were defined in detail in chapter 2. Intuitively, CM captures how closely two c-nodes are connected to each other than to the rest of the c-nodes in the graph.

2. Next, MKNN relationships are established amongst c-nodes using CM values as a similarity measure.
3. The next process in line is to construct a cluster initiator order by ordering all c-nodes as per their Cohesion value. The term cohesion was defined in chapter 2 in definition 2.14. Intuitively, this involves ordering all c-nodes as per the density inside them.

4. All c-nodes c-node, are traversed in their cluster initiator order to explore their MKNN c-nodes, c-node for either

(a) Merging c-node and c-node to form a bigger c-node, or

(b) Setting c-node as a boundary c-node for c-node.

In order to decide which of the two actions are taken for each pair of MKNN c-nodes, the following two checks are performed to ascertain if the two c-nodes are synchronized in terms of structural density and edge-weight density.

(a) Constraint A, Check for Structural density: Two c-nodes pass the structural similarity check if the Cohesion of the prospective merged c-node is greater than the Cohesion of the initiator c-node.

(b) Constraint B, Check for edge-weight density: Two c-nodes pass the edge weight density check if the ClusterSeparation between the two c-nodes as defined in definition 2.15 is within MBOUND. This ensures that the two c-nodes are homogeneous in terms of
edge weights and the variance of the merged cluster remains low.

The following rules are established.

(a) If \textit{c-node}$_i$ and its MKNN \textit{c-node}$_j$ satisfy both constraint A and B, then the two \textit{c-nodes}

- merge with each other to form a bigger sized core if they are cores, or are peripheries belonging to the same set of cores. Otherwise, the two \textit{c-nodes}

- form a weak periphery-periphery relationship. In this case, \textit{c-node}$_j$ is put in boundary nodeset \textit{pp} (periphery-periphery) of \textit{c-node}$_i$.

\textit{c-node}.

(b) If \textit{c-node}$_i$ and its MKNN \textit{c-node}$_j$ satisfy only the structural density constraint A, \textit{c-node}$_j$ is put in boundary nodeset \textit{low} of \textit{c-node}$_i$.

(c) If \textit{c-node}$_i$ and its MKNN \textit{c-node}$_j$ satisfy only the edge weight density constraint B, \textit{c-node}$_j$ is put in boundary nodeset \textit{inrange} of \textit{c-node}$_i$.

(d) If \textit{c-node}$_i$ and its MKNN \textit{c-node}$_j$ do not satisfy any of the constraint A or B, \textit{c-node}$_j$ is put in boundary nodeset \textit{low} of \textit{c-node}$_i$ if its edge weight density is lower than that of \textit{c-node}$_i$, or else in boundary nodeset \textit{high} of \textit{c-node}$_i$.

84
The pseudocode for phase 2 is given in algorithm 3.4.

3.3.3.4 CP Phase: Core Periphery Structure Formation

The phase 2 of CP-MKNN results in the formation of dense c-nodes and their corresponding c-nodesets low, in-range, high and pp lying in their neighborhood. The c-nodes lying in boundary c-nodesets could not merge with the dense core c-node due to being lower either in terms of structural density, or in terms of edge weight density than the core.

In the CP phase, c-nodes are traversed in the order of their Cohesion to explore their boundary c-nodesets for the formation of core — periphery and periphery — periphery relationships. A c-node could be assigned both as a Core as well as a Periphery. We call such c-nodes as Core — Periphery. Further, each relationship is assigned a type based upon the majority c-nodeset type as below.

1. Core with Type 1 periphery: This type of periphery has a lower structural density than its core. c-nodes lying in boundary c-nodesets in — range of the core c-node are made as type 1 peripheries of the core.

2. Core with Type 2 periphery: This type of periphery has a lower edge weight density than its core. c-nodes lying in boundary c-nodesets low of the core c-node are put in type 2 periphery of the core. If the core c-node lies in boundary c-nodeset high of the periphery c-node, then this periphery is also made a type 2 periphery of the core.
Algorithm 3.4: Merge Phase

Input:

\( SM \): Similarity matrix for graph \( G = (V, E) \)

\( P \): List of c-nodes (preliminary clusters)

\( C \): Cluster label array from preliminary phase

\( NODE\_PENALTY \): (for calculating cohesion)

Output:

Cluster label array \( C \) for i-nodes

Boundary nodesets inrange, low, high for each cluster.

1. Calculate \( CM_{[n \times n]} \) where \( n \) is the length of \( P \).
2. Find MKNN neighbors of all c-nodes \( \in P \) using CM for similarity weights.
3. Assign \( D \leftarrow \) temporary cluster label array for c-nodes.
4. Sort c-node list \( P \) in decreasing order of their cohesion.
5. Initialize c-node cluster label \( \lambda \)

repeat

for \( c\_node \_i \in P \) do

if \( D_i \) not set for \( c\_node_i \) then

Set \( c\_node_i \) as cluster initiator

\( D_i \leftarrow \lambda \)

\( \theta \leftarrow \) cluster label of i-nodes \( \in c \_node_i \)

for \( c\_node \_j \in MKNN \) of \( c\_node_i \) do

if \( c\_node_j \) not found cores for \( c\_node_i \) & structural sim. check is satisfied & edge-weight sim. check is satisfied then

if \( core.set(c \_node_i) = core.set(c \_node_j) \) then

Set \( D_j \leftarrow \) next \( \lambda \)

end

else

Make a periphery-periphery relation b/w \( c\_node_i \) and \( c\_node_j \)

end

else if structural sim. check is satisfied & edge-weight sim. check is not satisfied then

Put \( c\_node_j \) in boundary c-nodeset low of \( c\_node_i \).

else if \( core.set(c \_node_i) = core.set(c \_node_j) \) then

Put \( c\_node_j \) in boundary c-nodeset in-range of \( c\_node_i \)

else if edge-weight sim. check is not satisfied then

Put \( c\_node_j \) in boundary c-nodeset low or high of \( c \_node_i \) depending upon their mean difference.

end

Refresh \( P \) with the new set of merged c-nodes Recalculate \( CM \) and MKNN matrices

until no merging takes place in an iteration;
3. Periphery-Periphery: This type of relationship is formed between two peripheries which do not share the same set of cores, but still have a weak connection between the two. c-nodes lying in boundary c-nodeset \( pp \) are candidates for this relationship. These relationships seem to weakly connect two core periphery structures in a sibling relationship.

Next, the CPDistance and CPScore for all extracted relationships is calculated as per the definitions 3.4 and 3.5. A resultant core periphery structure of type 2 formed for Synthetic dataset 2 is shown in figure 3.7. Figure 3.8 shows an example of a core periphery structure with periphery type 1 for synthetic dataset 1. Further, the red core in figure 3.7 and the green core in figure 3.8 share a periphery \( (H, J, K, L, M, N, P, Q, Z) \), thereby forming a sibling relationship between the two cores. Algorithm 3.5 describes the steps of CP-Phase for extracting core periphery associations.

### 3.3.4 Complexity Analysis

CP-MKNN operates in four phases, namely, Initialization Phase, Preliminary Phase or Phase 1, Merge and Boundary Formation Phase or Phase 2 and lastly, Core Periphery Formation Phase or CP phase. The complexity analysis for the first three phases is the same as that for G-MKNN. The CP phase involves going through all the clusters and forming core periphery relationships. The overall worst case complexity of CP-MKNN remains the same as that of G-MKNN.
Figure 3.7: Core Periphery Structure with peripheries of Type 2 (Core: Red nodes, Peripheries: Brown and Pink nodes) for Synthetic Dataset 1

Figure 3.8: Core Periphery Structure with periphery Type 1 (Core: Green nodes, Periphery: Brown nodes) for Synthetic Dataset 1
**Algorithm 3.5: Extract Core-Periphery Structures**

**Input:**

$S_n$: Set containing c-nodes generated after merge phase

**Output:**

$C_p$: Data structure containing core-periphery associations $(C, P)$ along with their type and score

$P_p$: Data structure containing periphery-periphery associations $(P, P)$ along with their score

1. Initialize $C_p$ and $P_p$ as empty data structures.
2. Store in $O_i$ the c-nodes in $S_n$ by the order of Cohesion.
3. **for** $O_i \in O_n$ **do**
   4. Set $P$ to merged c-nodes in $O_i$’s boundary c-nodeset $inrange, low$
   5. **for** $P_j \in P$ **do**
      6. **if** ($P_j \in boundaryset low$ and $\mu(O_i) > \mu(P_j)$) or ($P_j \in boundaryset inrange and Cohesion(O_i) > Cohesion(P_j)$) **then**
         7. Form $(O_i, P_j)$ as a core-periphery association
         8. Set Type$(O_i, P_j)$ as the type of majority of boundary nodes in $P_j$ (Type 1: $inrange$, Type 2: low)
         9. Add $(O_i, P_j)$ along with its type, CPDistance and CPScore to $C_p$
      **end**
   10. **else if** ($P_j \in boundaryset pp$) **then**
        11. Form $(O_i, P_j)$ as a periphery-periphery relationship
    **end**
14. **end**
3.4 Experimental Evaluation

We performed extensive experimental evaluation on two synthetic and multiple real world datasets.

3.4.1 Results for Synthetic Datasets

We built two synthetic weighted graph datasets 2 and 3. Synthetic dataset 2 has 39 nodes and 74 edges (figure 2.2). We embedded two strong cores Core 1: (A, B, C, D, E, F, G, H) and Core 2: (AA, AB, AC, AD, AE, AF, AG, AH, AJ, AK, AL) and a weak core Core 3: (R, S, T, U, V) surrounded by peripheries. Synthetic dataset 3 has 49 nodes and 97 edges (figure 3.9). It was built so as to embed three cohesive cores surrounded by peripheries. The three cores are: Core 1: (A, B, C, D, E, F, G, H) Core 2: (J, K, L, M, N, P, Q) and Core 3: (R, S, T, U, V, W). Figures 3.11 and 3.10 present an example core and its peripheries found by CP-Greedy and CP-MKNN respectively on synthetic dataset 3.

3.4.1.1 Evaluation Measures

We use a term called \textit{StructuralDensity} ($\rho_S$) to measure the topological similarity of cores and peripheries.

$$\rho_S = \frac{\text{Number of edges in the cluster}}{\text{All possible number of edges in the cluster}}$$  \hspace{1cm} (3.5)
Figure 3.9: Synthetic Graph 3
Figure 3.10: An example Core and its Peripheries found by CP-MKNN on Synthetic Dataset 3

Figure 3.11: An example Core and its Peripheries found by CP-Greedy on Synthetic Dataset 3
A high value of $S$ for a cluster can be interpreted as high intra-cluster structural similarity.

Further, we use the average variance of weighted edges inside a cluster as a measure of edge weight density inside a graph. A low value of variance signifies homogenous edge weight connections inside a graph.

3.4.1.2 Results

3.4.1.2.1 Comparison of CP-MKNN and CP-Greedy for Synthetic Datasets

First of all, we present a comparison of clusters (core or periphery) obtained by CP-MKNN vs CP-Greedy. These results are presented in table 3.1. CP-Greedy’s cluster optimization criterion focusses only on the structural density measure $\text{Cohesion}$. Therefore, it obtains clusters with a higher structural density than that obtained by CP-MKNN. However, the variance of clusters obtained by CP-Greedy is higher than the variance of clusters obtained by CP-Greedy. CP-MKNN aims at obtaining clusters with a balance between high structural density and homogeneous edge weight density. This property helps CP-MKNN to separate very dense cores from their sparse surroundings. We illustrate this difference further using real datasets.

In table 3.2, we provide a comparison of the properties of core periphery structures obtained by CP-Greedy vs. CP-MKNN. We can see a common trend in both the datasets as well as both the algorithms that cores have a higher structural density than peripheries. For CP-MKNN, we also find that cores obtain a lower average variance than the periphery clusters.
Table 3.1: Comparison of clusters by CP-MKNN vs CP-Greedy on synthetic datasets

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Dataset</th>
<th>nClusters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>size &gt; 2</td>
</tr>
<tr>
<td>CP-Greedy</td>
<td>Synthetic 2</td>
<td>7</td>
</tr>
<tr>
<td>CP-MKNN (K=4)</td>
<td>Synthetic 2</td>
<td>6</td>
</tr>
<tr>
<td>CP-Greedy</td>
<td>Synthetic 3</td>
<td>8</td>
</tr>
<tr>
<td>CP-MKNN (K=4)</td>
<td>Synthetic 3</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3.2: Core Periphery structures by CP-MKNN vs CP-Greedy for Synthetic Datasets 2 and 3

<table>
<thead>
<tr>
<th></th>
<th>No.</th>
<th>Avg. Mean</th>
<th>Avg. Standard Deviation</th>
<th>Avg. Structure Density($\rho_s$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP − MKNN(K = 4) for Synthetic Dataset 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>3</td>
<td>0.71</td>
<td>0.043</td>
<td>0.56</td>
</tr>
<tr>
<td>Peripheries</td>
<td>3</td>
<td>0.39</td>
<td>0.061</td>
<td>0.45</td>
</tr>
<tr>
<td>Core Peripheries</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

| CP − Greedy for Synthetic Dataset 2 |     |           |                         |                                 |
| Cores              | 2   | 0.87      | 0.046                   | 0.4                             |
| Peripheries        | 2   | 0.37      | 0.069                   | 0.58                            |
| Core Peripheries   | 3   | 0.52      | 0.078                   | 0.65                            |

| CP − MKNN(K = 4) for Synthetic Dataset 3 |     |           |                         |                                 |
| Cores              | 3   | 0.88      | 0.025                   | 0.48                            |
| Peripheries        | 2   | 0.43      | 0.068                   | 0.64                            |
| Core Peripheries   | 2   | 0.65      | 0.089                   | 0.38                            |

| CP − Greedy for Synthetic Dataset 3 |     |           |                         |                                 |
| Cores              | 2   | 0.85      | 0.088                   | 0.46                            |
| Peripheries        | 2   | 0.45      | 0.083                   | 0.5                             |
| Core Peripheries   | 4   | 0.59      | 0.11                    | 0.57                            |
3.4.2 Results for Real Datasets

We use two real world PPI datasets widely used in the research community for the evaluation of core periphery structures obtained by CP-Greedy and CP-MKNN. These datasets are

- Gavin (1855 proteins with 7669 edges) [9]
- Krogan (2708 proteins and 7123 edges) [37]

Further, we use the MIPS [47] catalog of complexes as our gold standard or validation complexes. This validation dataset consists of 203 protein complexes and a total of 1189 proteins.

3.4.2.1 Evaluation Measures

We use the Accuracy (Acc.) measure, coined by Brohee and Helden [6] for evaluating the performance of core periphery structures. Accuracy is defined in terms of clustering-wise Sensitivity ($S_n$) and clustering-wise positive predicted value (PPV). $S_n$ and PPV are defined as follows. Let $n$ be the number of predicted clusters and $m$ be the number of gold standard complexes. Let $T= [t_{ij}]$ be the confusion matrix defined by the complexes. Let $m_i$ be the number of proteins belonging to gold standard complex. $S_n$, PPV and Accuracy are defined as follows.

\[
S_n = \left( \frac{\sum_{i=1}^{m} \max_{1 \leq j \leq n} t_{ij}}{\sum_{i=1}^{m} m_i} \right) \quad PPV = \left( \frac{\sum_{j=1}^{n} \max_{1 \leq i \leq m} t_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}} \right)
\]

(3.6)
Further, we use a measure called *Essentiality* to differentiate between cores and peripheries. Essential proteins are known to be critical for cell survival. The essentiality of proteins has been studied to be related to their structural characteristics in PPI networks [48] [18]. The essentiality of all proteins has been provided as quantitative values in the Essential Proteins Database [49]. We extracted the yeast protein essentiality values from this database. As an hypothesis, we expect core nodes to have a high value of essentiality on average than the periphery nodes.

### 3.4.2.2 Results

#### 3.4.2.2.1 Comparison of CP-Greedy and CP-MKNN for Real PPI datasets

In table 3.3, we provide a comparison analysis of clusters (both core and periphery) obtained by CP-Greedy and CP-MKNN using the *Accuracy* measure. The accuracy values are almost the same in the case of both the algorithms. However, when we look at $S_n$ and $PPV$ values for Gavin dataset, we can see that CP-MKNN obtains a better balance between $S_n$ and PPV. Intuitively, we can say that CP-MKNN favors cores to be very dense with a high PPV with proteins with low false positives.

The clustering results presented in table 3.3 provide insights into the structure of the datasets. We can gain more deeper insights into the datasets using a core periphery analysis. Next, we provide a comparison analysis of
Table 3.3: Comparison of CP-MKNN (K=4) vs CP-Greedy for PPI datasets

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>nClusters (size &gt; 2)</th>
<th>Sn</th>
<th>PPV</th>
<th>Acc.</th>
<th>Avg. Mean</th>
<th>Avg. ρs</th>
<th>Avg. Var.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gavin Dataset</td>
<td>CP-MKNN</td>
<td>233</td>
<td>0.44</td>
<td>0.41</td>
<td>0.43</td>
<td>0.41</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>CP-Greedy</td>
<td>245</td>
<td>0.56</td>
<td>0.39</td>
<td>0.47</td>
<td>0.38</td>
<td>0.63</td>
</tr>
<tr>
<td>Krogan Dataset</td>
<td>CP-MKNN</td>
<td>293</td>
<td>0.42</td>
<td>0.46</td>
<td>0.44</td>
<td>0.75</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td>CP-Greedy</td>
<td>332</td>
<td>0.49</td>
<td>0.44</td>
<td>0.46</td>
<td>0.73</td>
<td>0.49</td>
</tr>
</tbody>
</table>

The core periphery structures obtained by CP-Greedy and CP-MKNN on the PPI datasets. This analysis is presented in table 3.4. In both the algorithms, clusters are identified as being a core or a periphery. A cluster can be both a core and to some periphery and a periphery to another core. We identify such a cluster as core-periphery. It is to be noted that for both the algorithms, the cores have a higher average mean and structural density than the core-peripheries and peripheries. As per the t-statistic with $\alpha = 0.05$, this difference in cores and peripheries is found to be statistically significant. Further, both the algorithms find core-peripheries to have a statistically significantly higher average mean than peripheries for both the datasets. The difference in the structural density of core-peripheries and peripheries is not statistically significant. We also find that the cores found by CP-MKNN have a higher average essentiality value than the average essentiality value of peripheries (t-test, $\alpha=0.05$). This suggests that cores found by CP-MKNN have more essential proteins on average than the peripheries. This confers with existing literature suggesting the essentiality
Table 3.4: Core Periphery structures by CP-MKNN and CP-Greedy for PPI datasets

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CP – MKNN (K = 4) for Gavin Dataset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>67</td>
<td>0.56</td>
<td>0.14</td>
<td>0.84</td>
<td>0.38</td>
</tr>
<tr>
<td>Peripheries</td>
<td>45</td>
<td>0.28</td>
<td>0.024</td>
<td>0.52</td>
<td>0.28</td>
</tr>
<tr>
<td>Core Peripheries</td>
<td>101</td>
<td>0.35</td>
<td>0.083</td>
<td>0.49</td>
<td>0.33</td>
</tr>
<tr>
<td><strong>CP – Greedy for Gavin Dataset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>27</td>
<td>0.52</td>
<td>0.15</td>
<td>0.91</td>
<td>0.37</td>
</tr>
<tr>
<td>Peripheries</td>
<td>52</td>
<td>0.32</td>
<td>0.06</td>
<td>0.58</td>
<td>0.16</td>
</tr>
<tr>
<td>Core Peripheries</td>
<td>149</td>
<td>0.38</td>
<td>0.12</td>
<td>0.57</td>
<td>0.34</td>
</tr>
<tr>
<td><strong>CP – MKNN (K = 4) for Krogan Dataset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>88</td>
<td>0.89</td>
<td>0.093</td>
<td>0.58</td>
<td>0.37</td>
</tr>
<tr>
<td>Peripheries</td>
<td>60</td>
<td>0.51</td>
<td>0.096</td>
<td>0.52</td>
<td>0.16</td>
</tr>
<tr>
<td>Core Peripheries</td>
<td>112</td>
<td>0.77</td>
<td>0.095</td>
<td>0.41</td>
<td>0.32</td>
</tr>
<tr>
<td><strong>CP – Greedy for Krogan Dataset</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cores</td>
<td>14</td>
<td>0.93</td>
<td>0.059</td>
<td>0.71</td>
<td>0.23</td>
</tr>
<tr>
<td>Peripheries</td>
<td>55</td>
<td>0.48</td>
<td>0.079</td>
<td>0.54</td>
<td>0.22</td>
</tr>
<tr>
<td>Core Peripheries</td>
<td>247</td>
<td>0.77</td>
<td>0.13</td>
<td>0.45</td>
<td>0.29</td>
</tr>
</tbody>
</table>

Properties of nodes in a PPI network could be related to their topological properties in the network [48] [18]. For CP-Greedy, this trend is supported in the average essentiality results only for Gavin dataset.

### 3.4.2.2.2 Illustration of properties of CP types for Real PPI datasets

In table 3.5, we provide a summary of properties of different types of core-periphery relations found by CP-MKNN. Specifically, CP-MKNN finds two
types of peripheries in core-periphery relationships.

- Type 1 Peripheries (By definition, these type of peripheries are lower in structural density than the core).
- Type 2 Peripheries (By definition, these type of peripheries are lower in edge weight density than the core).

Further, CP-MKNN also finds sibling relations between core-periphery structures based upon loose periphery-periphery relations. Table 3.5 presents the number of core-periphery relationships belonging to each type, along with the average CPDistance (definition 3.4) and the average CPScore (3.5) for each type.

CPDistance is a measure of edge weight distance between core and periphery. A smaller value signifies that the core and periphery are closer to each other as per edge weight density. CPScore is a measure of structure similarity between core and periphery. A larger value signifies that the core and periphery are closer to each other as per structure density. It can be noted that type 1 peripheries are closer to their cores than type 2 peripheries as per edge weight density(lower value of average CPDistance). On the other hand, type 2 peripheries are closer to their cores than type 1 peripheries as per structural density(higher average CPScore). These results are statistically significant as per the t-test with $\alpha = 0.05$. This confirms our formulation of type 1 and type 2 peripheries.
Table 3.5: Comparison of relationships of different types obtained for CP-MKNN (K=4) for Gavin Dataset

<table>
<thead>
<tr>
<th>Relationship Type</th>
<th>No. Type Relations</th>
<th>Avg. CP Distance</th>
<th>Avg. CP Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>CP-Type 1</td>
<td>214</td>
<td>0.85</td>
<td>0.20</td>
</tr>
<tr>
<td>CP-Type 2</td>
<td>194</td>
<td>3.04</td>
<td>0.32</td>
</tr>
<tr>
<td>PP</td>
<td>109</td>
<td>4.9</td>
<td>0.22</td>
</tr>
</tbody>
</table>

3.4.2.2.3 Qualitative example of core periphery relationship in PPI datasets

In this section, we provide qualitative examples of core periphery structures found by CP-MKNN and CP-Greedy on Krogan PPI dataset. In figure 3.12, we demonstrate a core-periphery and a periphery-periphery relation found by CP-MKNN on the real Krogan dataset. The algorithm identifies a core periphery relation between a core cluster which maps to Exosome complex and a periphery cluster which maps to RNA-polymerase III complex. The algorithm also finds a periphery-periphery relationship between this periphery mapping to RNA-polymerase III complex and another periphery mapping to RNA-polymerase II complex.

These structures are in sync with research which shows that exosome complex is a central factor in processing stable RNA species produced by RNA polymerases I, II, and III [50].

A core periphery relation is found between a core cluster matching to Replication complexes and a periphery cluster matching to cytoplasmic-ribosomal-large-subunit A core periphery relation is found between a core
cluster matching to Replication complexes and a periphery cluster matching to cytoplasmic-ribosomal-small-subunit.

Further periphery-periphery relationship is found between cluster matching to cytoplasmic-ribosomal-large-subunit and a cluster matching to cytoplasmic-ribosomal-small-subunit.

Further, these two complexes share periphery-periphery and core-periphery relationships when matched to different clusters.

Figure 3.13 demonstrates example core periphery structures found by CP-Greedy on Krogan PPI dataset. A core clusters which maps to RSC-
complex is found to have two peripheries, namely,

- Periphery Type 1, matching to SWI-SNF-transcription-activator complex.
- Periphery Type 2, matching to Mitochondrial-processing complex.

A study of literature in bioinformatics revealed that these relationships have been studied by researchers. It has been pointed out in [51] that both RSC complex and SWI-SNF-transcription-activator complex are ATP dependent chromatic-remodeling complexes and are closely related to each other. Both regulate different chromatic regions with the RSC complex being required for a larger spectrum of genes. This makes RSC more abundant with its function as indispensable for cell survival [52]. Similarly, it has been studied in a very recently by Imamura et. al. [53] that RSC-complex is important for mitochondrial function in *Saccharomyces cerevisiae*.

### 3.4.2.2.4 Comparison of CP-Greedy and CP-MKNN for a Movie Co-appearance dataset

Using the real world PPI datasets, we demonstrated the usefulness of core periphery structures obtained by CP-Greedy and CP-MKNN. It became clear that both the algorithms obtain cores which are denser than the obtained peripheries. We also highlighted the differences between CP-MKNN and CP-Greedy using $S_n$, PPV and accuracy measures. It was shown that CP-MKNN favors denser cluster cores than CP-Greedy. In this section, we further demonstrate this difference between CP-MKNN
Figure 3.13: A qualitative example of Core Periphery structures obtained by CP-Greedy on PPI dataset, Krogan
Figure 3.14: A dense core by CP-MKNN (K=2) (blue nodes) and CP-Greedy (blue and red nodes) on *Les Miserables*, a movie co-appearance dataset.

**Dataset Description**

The *Les Miserables* dataset is a network of interactions amongst the major characters in Victor Hugo’s novel called Les Miserables. This dataset has been compiled by Knuth [54]. The corresponding graph contains 77 nodes and 508 edges with the edges being weighted by the number of scenes in which the characters appear together. The clustering coefficient of this dataset is 0.57.

Figure 3.14 illustrates a core found by CP-MKNN, K=2, (blue colored nodes) and a core found by CP-Greedy (Red and blue nodes combined). The blue nodes correspond to a very dense core comprising of the novel’s actor, Marius (MA); actress, Cosette (CO) and the actresses’s father, Jean Valjean (JV).
Valjean (JV). The red nodes comprise other important characters in the novel. Further, it can be noticed that even though the red and blue nodes are connected well as per structural density, they have different levels of edge weight density. CP-MKNN is able to extract the very dense part of nodes separately as a core because it uses both structure and edge weight density constraints in constructing core periphery structures.

3.4.2.2.5 Case Study: CP-Greedy and CP-MKNN for a spatio-temporal dataset

We illustrate the usefulness of core periphery structures found by CP-Greedy and CP-MKNN using a spatio-temporal dataset of crimes taking place in San Francisco (SF) [55]. We further demonstrate the differences between the two algorithms using this dataset.

Dataset Description

The SF crime dataset provides about 12 years of crime reports from across all of San Francisco’s neighborhoods. The data spans temporally from 2003 to 2015 and spatially across different zip codes across SF as shown in figure 3.15. Further each crime incident is categorized into one of the crime categories, such as, Larceny and Theft. Figure 3.16 presents the variation in SF crime dataset across 39 different crime types.

Construction of Graph from the data In order to study the differences in the spread of data for different crime types, we plotted contour plots for different crime types on a map of San Francisco. We present two of these plots for crime types Larceny/Theft and Prostitution in figure 3.17. These
Figure 3.15: Variation in SF crime dataset across time and space

(a) Temporal Variation in dataset

(b) Spatial Variation in dataset
Figure 3.16: Variation in SF dataset across 39 different crime types.

Density plots reveal that different crime types are spread differently across the map of SF. This motivated us to study the similarity between crime types based upon their spatial similarities and how these similarities change with time. We constructed a graph $G = (V, E)$ where $V$ corresponds to the set of 39 crime types. $E$ was constructed using the Tanimoto similarity measure between crimes based upon geographical proximity. A cutoff of 0.2 was used as a minimum similarity level between crime types required for an edge to exist between the two. Further, one graph was constructed for years from 2005, 2010 and 2015.

Given a graph of crime types, we sought to analyze if the core periphery analysis of this graph over years can help in understanding the modus operandii of crimes belonging to different crime types. In figures 3.18, 3.19 and 3.20 we present the core periphery structures found by CP-GMKNN and CP-Greedy for the years 2005, 2010 and 2015. These graphs represent...
(a) Crime density plot for crime type: 
*Larceny/Theft*

(b) Crime Density plot for crime type: 
*Prostitution*

Figure 3.17: Crime density contour plot for two crime types on a map of SF
Figure 3.18: Core Periphery structures for the year 2005

**Graph Details**
- No. of nodes: 37
- Structural Density: 0.88
- Standard Deviation: 0.27
- Clustering Coefficient: 0.943
(a) Core Periphery Structures by CP-MKNN (K=5)

(b) Core Periphery structures by CP-Greedy

Figure 3.19: Core Periphery structures for the year 2010

**Graph Details**

- No. of nodes: 38
- Structural Density: 0.95
- Standard Deviation: 0.245
- Clustering Coefficient: 0.962
Figure 3.20: Core Periphery structures for the year 2015

Graph Details
No. of nodes: 37
Structural Density: 0.76
Standard Deviation: 0.31
Clustering Coefficient: 0.86
crimes categorized as belonging to the core (green colored center cluster) and the surrounding peripheries with their respective score.

Crimes occurring in core represent the crimes that are more prevalent as per frequency as well as location and are similar other in these respects than the crime types in the periphery. It is further interesting to see that how certain crime types like Larceny/Theft, Vehicle Theft and Fraud are in core for all three years. Crime types like Gambling, Prostitution and Bribery are always located in the periphery. A crime type Embezzlement was located in the core in the year 2005 moves into being a periphery crime in the years 2010 and 2015. Such an analysis of crimes can be very useful to the county.

It is evident from these graphs that CP-MKNN is able to separate the very densest cores from their peripheries for all the three years. On the other hand, CP-Greedy results in one core as the whole graph for the years 2005 and 2010. This is because CP-Greedy uses only structure based criterion for growing cores and peripheries. CP-MKNN is able to capture the different gradients in edge weight density even in a structurally dense graph.

Robustness Analysis with varying K

We demonstrate the core periphery structures found by CP-MKNN are robust with small values of K. Figure 3.21 presents the results of the changing number of core and periphery nodes with three different values of K. The number of nodes in cores vs. peripheries changes very little with changing values of K.
Figure 3.21: Robustness of CP-MKNN with changing K for SF crime dataset

(a) Number of core nodes common between results for different values of K

(b) Number of periphery nodes common between results for different values of K
3.5 Conclusion

In this chapter, we presented a formal definition for core periphery structures in weighted graphs. Next, we presented two algorithms for finding core periphery structures in weighted graphs. The first approach, CP-Greedy is a greedy growth algorithm based upon an overlapping clustering algorithm called ClusterONE. The second algorithm called CP-MKNN is an MKNN based algorithm for finding core periphery structures in a weighted graph. The second algorithm also finds sibling relationships between two core periphery structures based upon a loose periphery-periphery connection. The algorithms also score and categorize the identified core periphery structures. We illustrate the usefulness of core periphery structures using two synthetic as well as two real protein-protein interaction networks. We provide quantitative as well qualitative results to demonstrate the effectiveness of core periphery structures found by CP-MKNN and CP-Greedy. Core proteins are found to be more essential on average than the periphery proteins. Further, we highlight the significant differences between CP-MKNN and CP-Greedy using a spatio-temporal dataset of crimes taking place in San Francisco. We show that CP-MKNN is able to separate cores even in very dense weighted graphs. These cores can be either structurally dense or dense as per edge weight density. While CP-Greedy algorithm uses only structural density constraint, and hence is not able to separate different gradients of cores and peripheries in the SF crime dataset.
Chapter 4

Overlapping Graph-Mutual K-Nearest Neighbor Clustering (OG-MKNN)

In this chapter, we present an algorithm called OG-MKNN that we have developed to find overlapping clusters in weighted graphs. We present a brief motivation followed by a description of the working of the algorithm. Further, we present the results of the clustering algorithm using synthetic and real world datasets.

4.1 Background

There are classic algorithms that exist to find overlapping group of points in real data space, such as fuzzy clustering [56] and biclustering [57]. How-
ever, a lot less research has been done in the area of finding overlapping clusters for graphs. Overlapping clusters hold a great significance in many real world domains; some of them are fairly new. For example, in social networks, overlapping clusters correspond to communities of people sharing common interests [58][59]. The study of such overlaps can help in understanding the dynamics of these networks, for example, the study of how different trends evolve. In PPI network clusters, overlaps may correspond to multi-functional proteins which take part in multiple functions in the cell simultaneously or at different time intervals. Thus, an overlapping clustering algorithm will be a better choice for the description of modules in PPI networks. Given the relevance of overlapping graph clusters in different domains, in this work, we build an overlapping graph clustering algorithm.

### 4.2 Related Work

The very first overlapping clustering algorithm was proposed by Jardine and Sibson [60] who introduced the concept of k-ultrametrics. The initial research efforts on overlapping clustering mainly laid emphasis on the theoretical distance analysis studies and model evaluation [61] [62] [56]. Bertrand and Janowitz [63] developed an overlapping clustering algorithm based upon the pyramid model proposed by Diday [61]. An overlapping version of the K-means algorithm was proposed by Cleuziou [64]. However,
their approach is not suitable for graphs.

A genetic algorithm based adaptive k-means algorithm has been proposed by Orgaz et al. [65] which finds overlapping clusters in graphs. All these algorithms work on unweighted graphs. We compare the results of our clustering algorithm with two other overlapping weighted graph clustering algorithms. The first one is ClusterONE [13], and has been described in detail in chapter 3. The second ones are called CPMw/CFinder [29]. We describe them below.

### 4.2.1 Clique Percolation Method (CFinder)

CFinder [29] [66] is an overlapping graph clustering algorithm based upon the Clique Percolation Method [67] to locate the k-clique percolation clusters as modules. The original algorithm works on undirected and unweighted graphs, however, it has been extended to support weighted graphs as well [25]. CFinder operates by finding k-cliques in the graph. These k-cliques are said to be adjacent if they share at least k-1 nodes. An accessibility graph is next built to define the connectivity between the k-cliques based upon this adjacency criterion. Each vertex in the accessibility graph corresponds to one k-clique, and the edges correspond to the connectivity between the k-cliques. The connected components of this accessibility graph represent the final overlapping clusters. A reference implementation of CFinder can be downloaded from [68].
4.3 Our Approach

We develop an overlapping clustering algorithm, OG-MKNN based upon the concept of MKNN. The algorithm is built in two phases similar to G-MKNN. In the first phase, preliminary clusters are built to divide the entire graph into small sized sub structures. In the next phase, the preliminary clusters are merged or overlapped together, with the overlapping decisions incorporated into the merging algorithm. We explain the working of these two phases in detail in the following sub sections. We will use Synthetic dataset 2, as described before in Chapter 2 in figure 2.2 for the description of this algorithm.

4.3.1 OG-MKNN Phase 1

In G-MKNN, the output of phase 1 yields small sized preliminary clusters, which are called as c-nodes in phase 2. In figure 2.8, we show the preliminary clusters obtained by G-MKNN on synthetic graph 2 after phase 1. The phase 1 of OG-MKNN is the same as the phase 1 of G-MKNN, and thus, these are also the preliminary clusters obtained after the phase 1 of OG-MKNN. Further, the pseudocode for this phase is given in algorithm 2.1.
4.3.2 OG-MKNN Phase 2: Merge and Boundary Formation

This phase of OG-MKNN is identical to the merge and boundary formation phase of CP-MKNN. To summarize, this phase involves merging $c$-nodes which satisfy the constraint of structural density as well as edge weight density. $c$-nodes which don’t satisfy either of these constraints are put in boundary $c$-nodesets of the core ($low$, $inrange$ or $high$). The pseudocode for phase 2 is given in algorithm 3.3.

4.3.3 OG-MKNN Phase 3: Overlap Phase

To begin with, this phase involves identifying two types of core periphery relations (Type 1 and Type 2) between $c$-nodes, as described in the phase 3 of CP-MKNN. The two types of core periphery relations extracted by this phase are summarized again below.

1. Type 1 periphery: This type of periphery has a lower structural density than its core. $c$-nodes lying in boundary $c$-nodesets $in-range$ of the core $c$-node are made as type 1 peripheries of the core.

2. Type 2 periphery: This type of periphery has a lower edge weight density than its core. $c$-nodes lying in boundary $c$-nodesets $low$ of the core $c$-node are put in type 2 periphery of the core. If the core $c$-node lies in boundary $c$-node $high$ of the periphery $c$-node, then
this periphery is also made a type 2 periphery of the core.

The next step involves identifying candidate c-nodes for overlap.

- A Type 1 periphery with less than 3 i-nodes is merged with all of the core c-nodes which have a core-periphery relationship with this periphery.

- A Type 2 periphery with less than 3 i-nodes is merged with all of the core c-nodes which have a core-periphery relationship with this periphery.

4.3.4 Computational Complexity

OG-MKNN involves one extra phase called the Overlap phase than CP-MKNN. The overlap phase involves going through all the clusters (core/periphery) and performing the overlap if possible. The worst case complexity of OG-MKNN remains the same as that of CP-MKNN.

4.4 Experimental Evaluation

4.4.1 Synthetic Dataset

We use synthetic dataset 3 as described in figure 3.9 in chapter 3. Figures 4.1, 4.2 and 4.3 present the overlapping clusters obtained by CPM, ClusterONE and OG-MKNN on this dataset. Each cluster is represented by a
color. Yellow color denotes overlapping nodes, whereas gray color denotes that the node has not been assigned any clustering label by the clustering algorithm.

CPM is able to identify the two high density cliquish clusters (R, S, T, U, V, W, CE) and (J, K, L, M, N, P, Q, DC, DD, DE). K=3. It fails to identify the third high density cluster (A, B, C, D, E, F, G) which is high in edge weight density but different from a cliquish structure. Further, the low density clusters surrounding the three density clusters are not completely identified by
Figure 4.2: Overlapping clusters obtained by ClusterONE on Synthetic dataset 3
Figure 4.3: Overlapping clusters obtained by OG-MKNN on Synthetic dataset 3 (K=4)
ClusterONE is able to capture the three high density clusters well. However, it also merges the lower density clusters partially with the three high density clusters (E.g., cluster (BA, BF, BE, BB, BG)). This compromises the variance or edge weight homogeneity of the obtained clusters.

OG-MKNN is able to capture the three high density clusters surrounded by lower density clusters. Further, it is able to identify the overlap between the clusters with least effect on the density of either of the overlapping clusters.

Table 4.1 shows a quantitative comparison of CPM, ClusterONE and OG-MKNN on Synthetic dataset 3.

### 4.4.2 Real Dataset Results

#### 4.4.2.1 Dataset Description: A Social Network

We use a widely used social network dataset called as Zachary’s Karate Club Network. Zachary collected a dataset of associations amongst 34 members
of a Karate club at a US university for a total period of three years [69]. The network represents each person as a node labeled from 1 to 34. Node 1 being the club’s instructor and node 34 being the club’s president. A disagreement happened between nodes 1 and 34 and the club got split into two factions, one of the groups including the supporters of node 1, i.e., the club’s instructor, and the other group consisting of the supporters of club’s president. Further, the network has weights associated with edges representing the common activities in which the club members took part in. Zachary was able to split this network into two factions using Ford-Fulkerson binary community algorithm [70]. The clustering coefficient of this dataset is 0.57, which is not a very high value indicating regions of low network topology.

4.4.2.2 Evaluation Measure used

We perform a comparison of the overlapping clustering algorithms using a metric called Omega Index [71]. The Rand Index [72] metric is meant to be used for only disjoint clusters, therefore it is insufficient for use in the case of overlapping communities. Omega Index builds upon Rand Index and Adjusted Rand Index by accounting for disjoint solutions and correcting for chance agreement [71]. The Omega Index between two clustering solutions $s_1$ and $s_2$ is given as below.

$$Omega(s_1, s_2) = \frac{Obs(s_1, s_2) - Exp(s_1, s_2)}{1 - Exp(s_1, s_2)}$$ (4.1)
The numerator corresponds to the observed agreement between two clusterings, Obs(s1, s2) adjusted by the expected agreement Exp(s1, s2). The denominator represents the maximum possible agreement adjusted by the expected agreement. The value of Omega Index is 1 if the two clustering solutions perfectly agree upon how each pair of objects is clustered. The Observed agreement Obs(s1, s2) represents the proportion of pairs of objects classified the same way by the two clustering solutions. Mathematically, it is calculated as below.

$$\text{Obs}(s1, s2) = \frac{\min(J,K)}{N} \sum_{j=0}^{A_j}$$

(4.2)

Here J and K correspond to the maximum number of clusters in which any pair of objects appear together in solutions 1 and 2 respectively. N is the number of pairs of objects. $A_j$ is the number of pairs in agreement by both solutions to be assigned to number of clusters j. Formally, the expected agreement is calculated as below.

$$\text{Exp}(s1, s2) = \frac{\min(J,K)}{N} \sum_{j=0}^{N_{j1}N_{j2}}$$

(4.3)

Here $N_{j1}$ and $N_{j2}$ is the total number of pairs of objects assigned to number of clusters j in solution 1 and solution 2 respectively.
4.4.2.3 Results

Figures 4.4, 4.5 4.6 represent the overlapping clusters found by CPM (K=3) and ClusterONE and OG-MKNN (K=4) respectively. The yellow nodes represent overlapping nodes. OG-MKNN is able to identify the two factions correctly, whereas ClusterONE detects three overlapping clusters. It splits node 1’s faction into two parts. Since OG-MKNN uses both structural density and edge-weight density constraints for clustering, it is able to avoid this cluster split.

The clique percolation method fails to detect the primary division in this graph into two factions. It favors clique shaped clusters.
Figure 4.5: Overlapping clusters obtained by ClusterONE on Zachary's Karate club dataset

Table 4.2: Comparison of overlapping clustering algorithms on Zachary’s Karate club dataset

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>nClusters size &gt; 2</th>
<th>Nodes with no clabel</th>
<th>Omega Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>OG-MKNN (K=4)</td>
<td>2</td>
<td>1</td>
<td>0.67</td>
</tr>
<tr>
<td>ClusterONE</td>
<td>3</td>
<td>3</td>
<td>0.63</td>
</tr>
<tr>
<td>CPM</td>
<td>3</td>
<td>0</td>
<td>0.04</td>
</tr>
</tbody>
</table>
Figure 4.6: Overlapping clusters obtained by OG-MKNN (K=4) on Zachary's Karate club dataset
In table 4.2, we provide a value of Omega Index obtained for each algorithm upon comparison with the gold standard communities for the Karate Club dataset as provided by Zachary in [69]. The Omega Index is very low for CPMw algorithm indicating that this algorithm is not able to capture the inherent real world clustering information in the network. OG-MKNN obtains the highest value of Omega Index along with capturing the two factions. The second highest value of Omega Index is obtained by ClusterONE, however, as pointed out before, it splits one of the factions further into two.

4.5 Conclusion

In this chapter, we build an overlapping graph clustering algorithm called OG-MKNN for weighted networks. The algorithm is based upon the idea of MKNN and does not make any prior assumption about the shape of the clusters. We implement the OG-MKNN algorithm using Python. We evaluate the performance of the algorithm on synthetic and real world datasets. We also compare the functioning of the algorithm with two other overlapping graph clustering techniques, namely, ClusterONE and CPMw.
Chapter 5

Functional Information Guided Semi-supervised Graph Clustering (SG-MKNN)

In this chapter, we develop a semi-supervised graph clustering algorithm specially targeted towards finding modules in Protein-Protein Interaction (PPI) networks. The module formation process is guided by the presence of functional information in the form of Gene Ontology enrichment annotations. We demonstrate the results of our approach using real-world yeast PPI datasets.
5.1 Background

With the advent of microprocessors and super computers, the sequencing of the human genome has become very much possible. Given this rich source of information, understanding of biological objects at a systems level has become very popular in this post-genomic era. It is a known fact now that several components inside a living cell are interconnected with each other. Therefore, a study of dynamic and complex networks such as protein-protein interaction networks, gene regulatory networks and metabolic networks is a key towards understanding the complexities within a cell. One popularly investigated method for studying such networks is to identify sub structures inside them performing specific tasks. These sub structures are mostly recognized in the research community by the name of functional modules. According to Harwell et al [73], a functional module is a discrete entity whose function is separable from those of other entities or modules. Examples of functional modules inside a cell are operons, regulons, interaction cascades, molecular complexes and signaling pathways [73]. The concept of functional modules provides a deeper understanding of the organization of bio molecular networks. Recently, several modularity based approaches have been proposed in the literature to find out functional modules in biological networks, for example, [4] and [6].

In this chapter, we mainly deal with protein-protein interaction or PPI networks. Protein-protein interactions refer to the associations between
protein molecules [74]. A thorough study of these associations is critical to the systems-level understanding of biological networks. Several high and low-throughput techniques have been developed recently to construct a large amount of PPI networks for different organisms. Examples of these techniques include yeast-two-hybrid (Y2H) [75], co-immunopurification or co-affinity purification followed by mass spectroscopy [76] and curation of published low-throughput experiments. With the accumulation of such large-scale PPI networks for different organisms, there is a great demand for developing computational methods to analyze and interpret such networks. In this chapter, we develop an MKNN based semi-supervised clustering technique to find modules in PPI networks. We make use of the GO Ontology database [77] to enhance our unsupervised clustering algorithm G-MKNN, and fine tune it to be used especially for clustering PPI networks.

5.2 Related Work

Protein interaction networks can be interpreted as graphs, with the proteins being represented as nodes and their associations as edges. Given this representation, classic graph clustering or graph partitioning techniques can be applied to find clusters or modules in PPI networks. Most of the previously studied techniques apply unsupervised graph clustering to detect densely connected sub structures as functional modules in PPI networks. Examples include algorithms such as MCODE, MCL and ClusterONE, which have been
described previously in detail in Chapter 2. Even the algorithm G-MKNN developed by us in chapter 2 is an unsupervised graph clustering algorithm, which we show to be suitable for clustering PPI networks. Further, most of the existing techniques for the identification of PPI functional modules are based upon the assumption that modules correspond to cliquish structures in graphs. This assumption may not hold true for all protein complexes. Several other structures like star or spoke model might exist as modules in many PPI networks [78]. Figure 5.1 shows some examples of different shapes of yeast MIPS protein complexes that are possible. One possible way to extract structures other than cliques in PPI networks is to take help from some domain knowledge to guide the process of clustering. In other words, to do supervised or semi-supervised graph clustering in place of completely unsupervised clustering.
For example, Qi et al. [78] built a supervised learning algorithm using a probabilistic Bayesian model to identify protein modules in PPI graphs. They used both the topological and biological properties of protein complexes to learn the parameters of the Bayesian model. The biological properties that they use are the average and maximum protein length and average and maximum protein weight of each sub graph. However, their method relies on a lot of prior knowledge about complexes, which may not be available for complexes in many real situations. In such cases, a semi-supervised approach [79] might prove to be suitable, where there is some prior knowledge and some unlabeled data.

Many supervised or semi-supervised clustering techniques for PPI networks are based upon taking the help of GO Ontology database [77] to make clustering decisions. The Gene Ontology (GO) database holds functional gene annotation in a hierarchical structure that reflects the relationship between the biological terms and associated gene products [80]. Thus, this database can be used to calculate similarities between gene/protein products based upon the GO hierarchical structure. Gene products having a high similarity between them will be much more functionally related to each other than any two random gene pairs. In other words, these two gene products are highly likely to fall into the same functional module. Several Semantic similarity measures have been proposed in literature to compute a measure of similarity amongst the GO terms. We categorize and describe some of the existing measures of semantic similarities in the
following section.

5.2.1 Semantic Similarity Measures between GO Terms

Gene Ontology provides well-defined and structured relationships amongst biological terms and the associated gene products. Semantic similarity scores between GO terms can be utilized to quantify the functional similarity scores between genes/proteins. The existing methods to find semantic similarity between GO terms can be divided into four categories.

1. Edge based Methods: These methods exploit the paths between GO terms in a Directed Acyclic Graph structure of GO. For example, such a method might involve calculating the depth of the common ancestor of two GO terms $t_1$ and $t_2$ as a measure of their similarity.

2. Node based Methods: These methods are based on measuring the overlap between the common ancestor terms of two GO terms [81]. Example, simUI [82] calculates the semantic similarity score between GO terms $t_1$ and $t_2$ as the overlap normalized by the union of ancestor terms of $t_1$ and $t_2$ respectively. Mathematically, simUI is formalized as below.

$$sim_{simUI} = \frac{C(t_1) \cap C(t_2)}{C(t_1) \cup C(t_2)}$$  \hspace{1cm} (5.1)

Here $C(t_1)$ and $C(t_2)$ correspond to the ancestor terms of $t_1$ and $t_2$.
respectively.

3. Annotation based methods: These methods define a measure of information content for each GO term based upon the number of proteins annotated for each GO term. Mathematically, the information content of a GO term $t_i$ is defined as $-\log P(t_i)$ where $P(t_i)$ is the proportion of proteins annotated to GO term $t_i$. For example, Resnick score [83] computes the semantic similarity between two GO terms as the greatest information content of the common ancestor terms of the two GO terms. Mathematically, the Resnick score between GO terms $t_1$ and $t_2$ is defined as below.

$$sim_{Resnick}(t_1, t_2) = \max_{t_0 \in C(t_1, t_2)} -\log(P(t_0))$$  (5.2)

Here $C(t_1, t_2)$ corresponds to all the common ancestor terms of $t_1$ and $t_2$.

4. Hybrid Methods: Many approaches combine the score calculation methodologies in two or more of the above techniques to achieve higher accuracy in calculating the functional similarity between proteins. For example, Wang method [84] is a hybrid of normalized node-based method and edge-based method. simICNP [85] uses a combination of Resnick’s method and edge based path length between the GO terms to compute the similarity score. Mathematically,
ICNP score between GO terms \( t_1 \) and \( t_2 \) is calculated by normalizing the Resnick score by the graph theoretic distance between \( t_1 \) and \( t_2 \).

\[
sim_{ICNP}(t_1, t_2) = \frac{-\log P(t_0)}{\text{len}(t_1, t_2) + 1}
\]  

(5.3)

In literature, functional similarity information generated based upon the similarity between GO terms has been used in different ways to cluster PPI networks. In the next section, we categorize and describe some of the existing techniques for the same.

### 5.2.2 Gene Ontology based Clustering Methods for PPI networks

Some techniques such as [30] [31] use this functional information as a final step of the clustering process to filter out functionally non-homogeneous clusters. Some other techniques use GO information to modify the input data or to guide the clustering process [80]. Below we discuss each of these types of techniques in detail.

#### 5.2.2.1 Using Gene Ontology (GO) to filter out clusters

King et al. [30] proposed a graph clustering algorithm for weighted PPI networks using the GO functional information in the final step to filter out clusters. Their algorithm is called as Restricted Neighborhood Search Clustering (RNSC) and it operates in two phases. In the first phase, the PPI
network is clustered using a set of random partitioning and diversification moves. In the second phase, GO annotations are used to calculate a measure of functional homogeneity of clusters. A p-value is attached to each cluster ranking its match to the GO annotations. Finally, clusters above a certain p-value are discarded to obtain the final set of predicted clusters. Li et al. [31] proposed a clustering algorithm, which as a final step of clustering, uses GO annotations to obtain a functional reliability score for clusters. Initially, clusters are obtained based upon hub removal to identify dense sub-graphs in the PPI networks. Clusters below a certain threshold of the reliability score are discarded.

5.2.2.2 Using Gene Ontology (GO) in the clustering process

In [80], Jing et al propose two semi-supervised strategies based upon prior knowledge about the GO functional information to find functional modules in PPI networks. They use GO to obtain a functional similarity score among protein pairs. This score is next used to supervise the clustering process in two ways. In the first method, the input graph of protein interactions is augmented with these functional similarity scores to form a new modified input graph. This modified graph is then fed as input to existing clustering strategies, such as MCODE and MCL. The second strategy that they propose involves using the functional similarity values in the learning process. Existing algorithms such as MCL, MCODE, and hierarchical clustering method are first used to build small sized clusters. The connection between these
small sized clusters is then built using their pair wise functional similarities. However, both these methods will inherit the shortcomings inherent in the base algorithm used for clustering.

5.3 Our Approach

We develop a semi-supervised graph clustering algorithm to find functional modules in PPI networks. The algorithm is based upon the G-MKNN graph clustering algorithm proposed in Chapter 2.

We incorporate semi-supervised information in the form of GO ontology terms to aid the clustering process. GO terms are organized as a hierarchical tree like structure. Several algorithms have been proposed [86] to find semantic similarity between GO terms based upon either the graph theoretical similarities in the tree or based upon the information content of the GO terms on the path connecting the two terms. We use a hybrid similarity measure called simICNP [85] which makes use of both the graph theoretic similarities as well the similarities based upon information content. This method has been described in the previous section.

Each gene is treated as a vector of corresponding related GO terms. In order to compute similarity between two genes, say gene1 and gene2, a corresponding semantic similarity is calculated between all pairs of GO terms in vectors corresponding to the two genes.

We incorporate the semantic similarity between GO terms into the SG-
MKNN algorithm in two possible ways to get functionally annotated clusters. These two ways are:

1. Pre-processing based SG-MKNN
2. Post-processing based SG-MKNN

We describe these two approaches in the following subsections.

### 5.3.1 Pre-processing based SG-MKNN

In this method, the input similarity matrix for a PPI network is augmented with GO functional similarities between proteins. This augmented similarity matrix is then fed to the G-MKNN clustering algorithm. The block diagram for this approach is presented in figure 5.2.

### 5.3.2 Post-processing based SG-MKNN

In this approach, the clustering results for a PPI network are first obtained using G-MKNN algorithm. This results in modules consisting of proteins. Using enrichment analysis, each module is associated with a vector of matching GO terms. Using this functional information from GO terms, each module is assigned a functional annotation score. Then, a score cutoff is defined. All modules with functional annotation score less than the cutoff are filtered out. The block diagram for this approach is presented in figure 5.3.
Figure 5.2: Block Diagram for Pre-processing based SG-MKNN

Figure 5.3: Block Diagram for Post-processing based SG-MKNN
Table 5.1: Comparison of post-SGMKNN (K=4) and pre-SGMKNN (K=4) using MIPS complexes as a gold standard

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$S_n$</th>
<th>PPV</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Collins Dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-SGMKNN</td>
<td>0.57</td>
<td>0.42</td>
<td>0.49</td>
</tr>
<tr>
<td>Post-SGMKNN</td>
<td>0.53</td>
<td>0.45</td>
<td>0.49</td>
</tr>
<tr>
<td><strong>Gavin Dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-SGMKNN</td>
<td>0.52</td>
<td>0.38</td>
<td>0.45</td>
</tr>
<tr>
<td>Post-SGMKNN</td>
<td>0.43</td>
<td>0.40</td>
<td>0.42</td>
</tr>
<tr>
<td><strong>Krogan Dataset</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pre-SGMKNN</td>
<td>0.44</td>
<td>0.43</td>
<td>0.44</td>
</tr>
<tr>
<td>Post-SGMKNN</td>
<td>0.42</td>
<td>0.45</td>
<td>0.44</td>
</tr>
</tbody>
</table>

5.3.3 Complexity Analysis

Pre-processing based SG-MKNN involves creating a input matrix of functional similarities among the proteins. This matrix is then fed to the G-MKNN algorithm. Therefore, Pre-processing based SG-MKNN involves an extra task of constructing the initial matrix than G-MKNN. Let the number of proteins in the graph be $n$. Also, let the maximum number of GO terms mapped to a protein be $g$. Then the pre-processing steps using $O(n^2)$ steps to go over all pairs of proteins and further $O(g^2)$ steps to find GO-based similarity between a pair of proteins.

Post-processing based SG-MKNN involves operating on the clustering obtained by G-MKNN. The first step involves annotating each cluster using GO enrichment analysis. After that a functional score is calculated for each cluster. This involves calculating GO-based pair wise functional similarities.
between proteins in a cluster. If the size of the largest cluster is $p$, then, this overhead is $O(p^2)$ for a cluster.

### 5.4 Experimental Evaluation

In table 5.1, we present a comparative analysis of pre-processing based SG-MKNN and post-processing based SG-MKNN using real world PPI datasets as used in chapter 2. Both post-SGMKNN and pre SGMKNN obtain almost similar values of accuracy in the case of all the three PPI datasets. However, when we look individually into $Sn$ and PPV values, we observe that pre-SGMKNN obtains high $Sn$ values, thereby getting a better coverage of all genes in the gold standard. On the other hand, post-SGMKNN obtains higher PPV values than pre-SGMKNN, meaning that it is able to obtain less false positives at the cost of less coverage of genes. Thus, the post-processing version of SG-MKNN can be seen as a filtering process to lower the false positives in the clustering result. Pre-SGMKNN can be seen as a pre-processing step in SG-MKNN effective in improving the coverage of genes clustered by SG-MKNN.

### 5.5 Conclusion

In this chapter, we build a GO based functional information guided semi-supervised graph clustering algorithm for weighted PPI networks. The algorithm is implemented in two ways, incorporating the functional in-
formation as a pre-processing step (pre-SGMKNN) or as a post-processing step (post-SGMKNN). The GO information is merged with the topological properties of the graph to make preliminary cluster merging decisions. We validate the functioning of our algorithm on real weighted PPI datasets. We provide a comparative analysis of pre-SGMKNN and post-SGMKNN using these datasets. Even though we use PPI datasets to demonstrate the usefulness of SG-MKNN, this algorithm can be used in any domain where semi-supervised information is available. For example, for clustering products in a retail store, semi-supervised information might be available in the form of semantic similarity between products calculated using ontological nearness between the products.
Chapter 6

Conclusions and Future Directions

Graph Mining is an important problem in data mining and has tremendous applications. The analysis of graphs at different levels can help in understanding the functionality of the underlying real world complex networks. Many different concepts have been used for the study of complex networks at the global, local and intermediate scale.

In this dissertation, we focused on developing meso-scale structures for weighted graphs, namely Community Structure and Core Periphery Structure. In the research community, there are many widely acceptable definitions for these structures. Methods such as modularity optimization, random walks, clustering coefficient, spectral clustering, stochastic block models, and overlapping tiles formulation have been used to develop efficient algorithms for finding community and core periphery structures.
In our work, we develop a heuristic called Mutual K nearest neighbors to capture a notion of structure and edge weight density in weighted graphs. We use this heuristic to build an algorithm to identify clusters and core periphery structures in weighted graphs. Specifically, we build a disjoint community detection algorithm called G-MKNN, an overlapping graph clustering algorithm called OG-MKNN and a semi-supervised graph clustering algorithm called SG-MKNN. We also build two algorithms to find core periphery structures in weighted graphs. We provide a definition of core periphery structures suitable for weighted graphs. The first algorithm that we build (CP-MKNN) is based on using the MKNN heuristic. The second algorithm is built as a greedy methodology called CP-Greedy to find core periphery structures in weighted graphs.

Further, we demonstrated the effectiveness of all our developed algorithms by applying them on synthetic and real world datasets. We compared the functionality of each algorithm with state of the art algorithms in the research community.

### 6.1 Future Work

The different threads of research on Graph Clustering that we explored in this dissertation can be further extended. We will summarize some directions that we find worth exploring in future work.
6.1.1 G-MKNN Clustering with Node Attributes

In many real-world networks, nodes are assigned with meaningful attributes. For example, in a social network, each person might be assigned with a score based upon his/her popularity. This information in the form of node attributes can be used to find communities in networks, (E.g. [87]). Traditionally, algorithms have mostly focused on network structure information to find communities in graphs. The heuristic MKNN can be modified to incorporate the information about node attributes in forming mutual relationships. Therefore, two nodes with similar node attributes could be each other’s MKNN even if structurally they are not very well connected. Once the MKNN relations have been found in this manner, clustering can be performed similar to that in the G-MKNN algorithm that we have developed in this dissertation.

6.1.2 G-MKNN clustering on the Cloud

In our work we have proposed algorithms to find clusters and core periphery structures using MKNN. It will be interesting to explore that how MKNN will be calculated on distributed computing platform like Hadoop. Specifically, this will involve identifying the steps of the algorithm that can be performed in a disjoint and parallel manner to get some intermediate results. Further, the algorithm will require merging these intermediate results from different parallel steps to get the final clustering output. The
obtained speedup will be very useful in large scale real world networks.

6.1.3 Using MKNN for other meso-scale structures in graph

Role equivalence/assignment [88] is a meso-scale structure which is gaining popularity in the research community. This problem involves assigning roles to nodes in the network to identify their function in the community. Further, nodes with a similar role need not be a part of the same densely connected set. This assignment is not simply based upon the density of connections. For example in a social network, two nodes will be equivalent if they play similar social roles. For example, given a network of people working at an office, node roles could be based upon who is a manager, who is a senior level worker, or who is a team lead. Such an assignment of role requires looking into more than just density surrounding the nodes. Perhaps, looking into the social environment of nodes in some measurable way.

Traditionally, the notion of roles has been defined based upon graph equivalences such as structural, regular and stochastic equivalences. We propose the use of MKNN heuristic for the problem of role equivalence. MKNN helps to capture nodes at the same level of edge weight density. We combined this heuristic with structural density measure in a graph to identify dense edge-weight or structurally dense regions in graphs. Such a measure could have a potential in identifying nodes with similar roles in the network.
Bibliography


