The Dissertation Committee for Joyce Jiyoung Whang
certifies that this is the approved version of the following dissertation:

Overlapping Community Detection in Massive Social Networks

Committee:

Inderjit S. Dhillon, Supervisor

Kristen Grauman

Raymond J. Mooney

Keshav Pingali

David F. Gleich

Overlapping Community Detection in Massive Social Networks

by

Joyce Jiyoung Whang, B.S.

Dissertation

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

December 2015

Dedicated to my parents

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Inderjit Dhillon for his knowledgeable advice, valuable discussions, and constant support. Professor Dhillon's expertise and deep insight enabled me to complete this thesis. Professor Dhillon has encouraged me to explore diverse research ideas and has always guided me towards better research directions. I believe Professor Dhillon has played the most important role in making my Ph.D. life worthwhile and memorable.

I also would like to thank Professor David Gleich at Purdue University who has provided good motivation for my first paper on overlapping community detection. Professor Gleich's suggestions and numerous discussions have contributed to improving the quality and the presentation of my work. It was a great pleasure to work with Professor Gleich.

I am also grateful to my other committee members, Professor Kristen Grauman, Professor Raymond Mooney, and Professor Keshav Pingali for their insightful discussions and valuable comments.

I also thank my labmates and co-authors. I want to thank Nagarajan Natarajan for being a good friend and kindly helping me whenever I need help, Xin Sui for his kind help and giving me happy memories of my first paper at UT Austin, Piyush Rai for giving me a chance to think about new ideas in stochastic blockmodels, Yangyang Hou at Purdue University for being a good collaborator and having

good discussions, David Inouye for giving me useful comments on my presentations, and Andrew Lenharth for helping me to implement parallel graph mining algorithms. Also, I thank my Korean friends in the CS and ECE departments for letting me have such a great time in Austin.

Finally, I would like to thank my parents for their love and warm support. I dedicate this thesis to my parents.

Overlapping Community Detection in Massive Social Networks

Publication No. _____

Joyce Jiyoung Whang, Ph.D.
The University of Texas at Austin, 2015

Supervisor: Inderjit S. Dhillon

Massive social networks have become increasingly popular in recent years. Community detection is one of the most important techniques for the analysis of such complex networks. A community is a set of cohesive vertices that has more connections inside the set than outside. In many social and information networks, these communities naturally overlap. For instance, in a social network, each vertex in a graph corresponds to an individual who usually participates in multiple communities. In this thesis, we propose scalable overlapping community detection algorithms that effectively identify high quality overlapping communities in various real-world networks.

We first develop an efficient overlapping community detection algorithm using a seed set expansion approach. The key idea of this algorithm is to find good seeds and then greedily expand these seeds using a personalized PageRank clustering scheme. Experimental results show that our algorithm significantly outperforms other state-of-the-art overlapping community detection methods in terms of run time, cohesiveness of communities, and ground-truth accuracy.

To develop more principled methods, we formulate the overlapping community detection problem as a non-exhaustive, overlapping graph clustering problem where clusters are allowed to overlap with each other, and some nodes are allowed to be outside of any cluster. To tackle this non-exhaustive, overlapping clustering problem, we propose a simple and intuitive objective function that captures the issues of overlap and non-exhaustiveness in a unified manner. To optimize the objective, we develop not only fast iterative algorithms but also more sophisticated algorithms using a low-rank semidefinite programming technique. Our experimental results show that the new objective and the algorithms are effective in finding ground-truth clusterings that have varied overlap and non-exhaustiveness.

We extend our non-exhaustive, overlapping clustering techniques to co-clustering where the goal is to simultaneously identify a clustering of the rows as well as the columns of a data matrix. As an example application, consider recommender systems where users have ratings on items. This can be represented by a bipartite graph where users and items are denoted by two different types of nodes, and the ratings are denoted by weighted edges between the users and the items. In this case, co-clustering would be a simultaneous clustering of users and items. We propose a new co-clustering objective function and an efficient co-clustering algorithm that is able to identify overlapping clusters as well as outliers on both types of the nodes in the bipartite graph. We show that our co-clustering algorithm is able to effectively capture the underlying co-clustering structure of the data, which results in boosting the performance of a standard one-dimensional clustering.

Finally, we study the design of parallel data-driven algorithms, which enables us to further increase the scalability of our overlapping community detection algorithms. Using PageRank as a model problem, we look at three algorithm design axes: work activation, data access pattern, and scheduling. We investigate the impact of different algorithm design choices. Using these design axes, we design and test a va-

riety of PageRank implementations finding that data-driven, push-based algorithms are able to achieve a significantly superior scalability than standard PageRank implementations. The design choices affect both single-threaded performance as well as parallel scalability. The lessons learned from this study not only guide efficient implementations of many graph mining algorithms but also provide a framework for designing new scalable algorithms, especially for large-scale community detection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Community detection is one of the most important and fundamental tasks in network analysis. Given a network, a community is defined to be a set of cohesive nodes that has more connections inside the set than outside. Since a network can be modelled as a graph with vertices and edges, community detection can be thought as a graph clustering problem where each community corresponds to a cluster in the graph. The goal of traditional graph clustering algorithms (e.g., Metis [46], Graclus [32]) is to partition a graph such that every node belongs to exactly one cluster. However, in many social and information networks, nodes participate in multiple communities. For instance, in a social network, nodes represent individuals and edges represent social interactions between the individuals. In this setting, a node's communities can be interpreted as its social circles. Thus, it is likely that a node belongs to multiple communities, i.e., communities naturally overlap. To find these communities, we study the problem of overlapping community detection where communities are allowed to overlap with each other and some nodes do not belong to any cluster.

There are many practical applications of overlapping community detection. For example, one interesting problem in network analysis is information propaga-

tion where the goal is to analyze how information flows through the network. When certain information is given to a set of nodes in the network, the information will be gradually propagated through the network. The speed of information propagation depends on the influence of the nodes that deliver the information. Intuitively, if nodes are placed on the overlapped region between two different communities, then the information can be quickly delivered to both of the communities. As a result, in marketing, for example, when a company wants to maximize the effect of advertisement by offering some special promotions to a certain set of people, they might target the nodes that are placed on the overlapped region between communities. Also, in bioinformatics, mining gene networks is an important task. In a gene network, each community corresponds to a functional class. In this setting, finding overlapping communities corresponds to finding genes that participate in different functional classes. Other than these examples, there are many other different practical applications of overlapping community detection in various disciplines.

In this thesis, we propose scalable overlapping community detection algorithms that effectively identify high quality overlapping communities in various real-world networks.

We first propose an efficient overlapping community detection algorithm using a seed expansion approach in Chapter 2. The key idea of this algorithm is to find good seeds and then greedily expand these seeds based on a community metric. Within this seed expansion method, we investigate the problem of how to determine good seed nodes in a graph. In particular, we develop new seeding strategies for a personalized PageRank clustering scheme that optimizes the conductance community score. An important step in our method is the neighborhood inflation step where seeds are modified to represent their entire vertex neighborhood. Experimental results show that our seed expansion algorithm outperforms other state-of-the-art overlapping community detection methods in terms of producing cohesive clusters

and identifying ground-truth communities. We also show that our new seeding strategies are better than existing strategies and are thus effective in finding good overlapping communities in real-world networks.

By exploiting the connection between community detection and clustering, we develop more principled algorithms in Chapter 3 where we formulate the overlapping community detection problem as a non-exhaustive, overlapping graph clustering problem. The goal of the non-exhaustive, overlapping clustering is to identify overlapping clusters and also detect outliers simultaneously. We propose a simple and intuitive objective function that captures the issues of overlap and non-exhaustiveness in a unified manner. Our objective function can be viewed as a reformulation of the traditional k-means objective, with easy-to-understand parameters that capture the degrees of overlap and non-exhaustiveness. By studying the objective, we are able to obtain a simple iterative algorithm which we call NEO-K-Means (Non-Exhaustive, Overlapping K-Means). By considering an extension to weighted kernel k-means, we can tackle the case of non-exhaustive and overlapping graph clustering, and this allows us to apply our NEO-K-Means algorithm to the overlapping community detection problem. Our experimental results show that the new objective and algorithm are effective in finding ground-truth clusterings that have varied overlap and non-exhaustiveness; for the case of graphs, we show that our algorithm outperforms state-of-the-art overlapping community detection methods.

The iterative NEO-K-Means algorithm is fast but suffers from the classic problem that iterative algorithms for k-means fall into local minimizers given poor initialization. To get a more accurate and reliable solution, we propose a novel convex semidefinite program (SDP) as a relaxation of the non-exhaustive, overlapping clustering problem in Chapter 4. Although the SDP formulation enjoys attractive theoretical properties with respect to global optimization, it is computationally intractable for large problem sizes. As an alternative, we optimize a low-rank fac-

torization of the solution. The resulting problem is non-convex but has a smaller number of solution variables. We construct an optimization solver using an augmented Lagrangian methodology that enables us to deal with problems with tens of thousands of data points. The new solver provides more accurate and reliable answers than other approaches.

We further extend our NEO-K-Means ideas to co-clustering problems in Chapter 5. The goal of co-clustering is to simultaneously identify a clustering of the rows as well as the columns of a data matrix. Indeed, the co-clustering problem can be thought as a clustering of a bipartite graph. For example, in recommender systems, users have ratings on items and this can be represented by a bipartite graph where users and items are denoted by two different types of nodes, and the ratings are denoted by weighted edges between the users and the items. To find overlapping clusters as well as outliers on both types of the nodes in the bipartite graph, we propose a new objective function and a simple iterative algorithm which we call NEO-Co-Clustering (Non-Exhaustive, Overlapping Co-Clustering). We show that our NEO-Co-Clustering technique is able to effectively capture the underlying co-clustering structure of the data, and thus can be a useful tool for clustering of bipartite graphs that can arise in many real-world applications.

Finally, we study the design of parallel data-driven algorithms in Chapter 6, which enables us to further increase the scalability of our overlapping community detection algorithms. Using PageRank as a model problem, we look at three algorithm design axes: work activation, data access pattern, and scheduling. We investigate the impact of different algorithm design choices. Using these design axes, we design and test a variety of PageRank implementations finding that data-driven, push-based algorithms are able to achieve a significantly superior scalability than standard PageRank implementations. The design choices affect both single-threaded performance as well as parallel scalability. The lessons learned from this

study not only guide efficient implementations of many graph mining algorithms but also provide a framework for designing new scalable algorithms.

Chapter 2

Personalized PageRank-based Overlapping Community Detection

We develop an efficient overlapping community detection algorithm using a seed set expansion approach. The main contribution of our work is a new overlapping community detection algorithm with performance that greatly exceeds the state-of-the-art. This contribution was accomplished by studying new ideas in the prototypical "seed-and-grow" meta-algorithm for overlapping communities. We study each step of the overall computational pipeline in detail on real-world networks to demonstrate the utility of each component of the algorithm. Our experimental results show that our overlapping community detection algorithm significantly outperforms other methods in terms of run time, cohesiveness of communities, and ground-truth accuracy.

These local seed expansion methods are among the most successful strategies

The materials presented in this chapter have been published in [79], and an extended version of the work is currently under review [80]. Joyce developed the algorithms and conducted experiments. Professor Gleich and Professor Dhillon supervised the work.

for overlapping community detection [83]. However, principled methods to choose the seeds are few and far between. When they exist, they are usually computationally expensive (e.g., using maximal cliques as seeds [72]). Empirically successful strategies include exhaustively exploring all individual seeds and greedy methods that randomly pick a vertex, grow a cluster, and continue with any unassigned vertex.

To find a set of good seeds, we present two effective seeding strategies that we call "Graclus centers" and "Spread hubs." The "Graclus centers" seeding is based on the same distance kernel that underlies the equivalence between kernel k-means and graph clustering objectives [32]. Using this distance function, we can efficiently locate a good seed *within* an existing set of cohesive vertices of the graph. Specifically, we first compute many clusters using a multi-level weighted kernel k-means algorithm on the graph (the Graclus algorithm) [32], then use the corresponding distance function to compute the "centroid vertex" of each cluster. We use the neighborhood set of each centroid vertex as a seed region for community detection. The idea of "Spread hubs" seeding is to select an independent set of high degree vertices. This seeding strategy is inspired by recent observations that there should be good clusters around high degree vertices in real-world networks with a power-law degree distribution [82], [37].

The algorithm we use to grow a seed set is based on personalized PageRank (PPR) clustering [7]. The high level idea of this expansion method is to first compute the PPR vector for each of the seeds, and then expand each seed based on the PPR score. It is important to note that we can have multiple nodes in the personalization vector, and indeed we use the entire vertex neighborhood of a seed node as the personalization vector for PPR. This *neighborhood inflation* plays a critical role in the success of our algorithm. The full algorithm to compute overlapping clusters from the seeds is discussed in Chapter 2.2. We name our algorithm NISE by

abbreviating our main idea, Neighborhood-Inflated Seed Expansion.

Our experimental results show that our seeding strategies are better than existing seeding strategies and effective in finding good overlapping communities in real-world networks. More importantly, we observe that NISE significantly outperforms other state-of-the-art overlapping community detection methods in terms of producing cohesive clusters and identifying ground-truth communities. Also, our method scales to problems with over 45 million edges, whereas other existing methods could not run to completion on these large datasets.

2.1 Preliminaries

We formally describe the overlapping community detection problem and review some important concepts in graph clustering. Also, we introduce real-world networks which are used in our experiments.

2.1.1 Problem Statement

Given a graph $G = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$, we can represent the graph as an adjacency matrix A such that $A_{ij} = e_{ij}$ where e_{ij} is the edge weight between vertices i and j, or $A_{ij} = 0$ if there is no edge. We assume that graphs are undirected, i.e., A is symmetric. The goal of the traditional, exhaustive graph clustering problem is to partition a graph into k pairwise disjoint clusters $\mathcal{C}_1, \cdots, \mathcal{C}_k$ such that $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k = \mathcal{V}$. On the other hand, the goal of the overlapping community detection problem is to find overlapping clusters whose union is not necessarily equal to the entire vertex set $\mathcal{V}$. Formally, we seek k overlapping clusters such that $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k \subseteq \mathcal{V}$.

2.1.2 Measures of Cluster Quality

There are some popular measures for gauging the quality of clusters: cut, normalized cut, and conductance. Let us define $\texttt{links}(\mathcal{C}_p, \mathcal{C}_q)$ to be the sum of edge weights between vertex sets $\mathcal{C}_p$ and $\mathcal{C}_q$.

Cut. The cut of cluster $\mathcal{C}_i$ is defined as the sum of edge weights between $\mathcal{C}_i$ and its complement, $\mathcal{V}\backslash\mathcal{C}_i$:

$$\texttt{cut}(\mathcal{C}_i) = \texttt{links}(\mathcal{C}_i, \mathcal{V}\backslash\mathcal{C}_i). \tag{2.1}$$

Normalized Cut. The normalized cut of a cluster is defined by the cut with volume normalization as follows:

$$\texttt{NCut}(\mathcal{C}_i) = \frac{\texttt{cut}(\mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}. \tag{2.2}$$

Conductance. The conductance of a cluster is defined to be the cut divided by the least number of edges incident on either set $\mathcal{C}_i$ or $\mathcal{V}\backslash\mathcal{C}_i$:

$$\texttt{cond}(\mathcal{C}_i) = \frac{\texttt{cut}(\mathcal{C}_i)}{\min\big(\texttt{links}(\mathcal{C}_i, \mathcal{V}), \texttt{links}(\mathcal{V}\backslash\mathcal{C}_i, \mathcal{V})\big)}.$$

By definition, $\texttt{cond}(\mathcal{C}_i) = \texttt{cond}(\mathcal{V}\backslash\mathcal{C}_i)$. The conductance of a cluster is the probability of leaving that cluster by a one-hop walk starting from the smaller set between $\mathcal{C}_i$ and $\mathcal{V}\backslash\mathcal{C}_i$. Notice that $\texttt{cond}(\mathcal{C}_i)$ is always greater than or equal to $\texttt{NCut}(\mathcal{C}_i)$.

2.1.3 Graph Clustering and Weighted Kernel k-means

The normalized cut objective of a graph G is defined by

$$\texttt{NCut}(G) = \min_{\mathcal{C}_1,\ldots,\mathcal{C}_k} \sum_{i=1}^{k} \frac{\texttt{links}(\mathcal{C}_i, \mathcal{V}\backslash\mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}. \tag{2.3}$$

Table 2.1: Summary of real-world networks.

Category	Graph	No. of vertices	No. of edges	Avg. CC	Source
Collaboration	HepPh	11,204	117,619	0.6216	[4]
	AstroPh	17,903	196,972	0.6328	[4]
	CondMat	21,363	91,286	0.6417	[4]
	DBLP	317,080	1,049,866	0.6324	[4]
Product	Amazon	334,863	925,872	0.3967	[4]
Social	Orkut	731,332	21,992,171	0.2468	[4]
	Flickr	1,994,422	21,445,057	0.1881	[61]
	Myspace	2,086,141	45,459,079	0.1242	[75]
	LiveJournal	1,757,326	42,183,338	0.2400	[75]
	LiveJournal2	1,143,395	16,880,773	0.2535	[4]

This objective is equivalent to a weighted kernel k-means objective with the weight of each data point set to the degree of a vertex, and the kernel matrix set to $K = \sigma D^{-1} + D^{-1}AD^{-1}$, where D is the diagonal matrix of degrees (i.e., $D_{ii} = \sum_{j=1}^{n} A_{ij}$ where n is the total number of nodes), and σ is a scalar typically chosen to make K positive-definite [32]. Then, we can quantify the kernel distance between a vertex $v \in \mathcal{C}_i$ and a cluster $\mathcal{C}_i$, denoted $\texttt{dist}(v, \mathcal{C}_i)$, as follows:

$$\texttt{dist}(v, \mathcal{C}_i) = -\frac{2\,\texttt{links}(v, \mathcal{C}_i)}{\texttt{deg}(v)\,\texttt{deg}(\mathcal{C}_i)} + \frac{\texttt{links}(\mathcal{C}_i, \mathcal{C}_i)}{\texttt{deg}(\mathcal{C}_i)^2} + \frac{\sigma}{\texttt{deg}(v)} - \frac{\sigma}{\texttt{deg}(\mathcal{C}_i)} \tag{2.4}$$

where $\texttt{deg}(v) = \texttt{links}(v, \mathcal{V})$, and $\texttt{deg}(\mathcal{C}_i) = \texttt{links}(\mathcal{C}_i, \mathcal{V})$.

2.1.4 Datasets

We use ten different real-world networks including collaboration networks, social networks, and a product network from [4], [75], and [61]. The networks are presented in Table 2.1. All the networks are loop-less, connected, undirected graphs.

In a collaboration network, vertices indicate authors, and edges indicate co-authorship. If authors u and v wrote a paper together, there exists an edge between

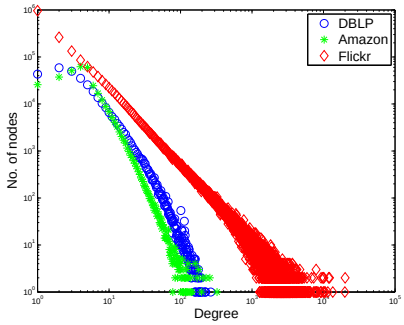


Figure 2.1: Degree distributions of real-world networks – the degree distributions follow a power-law.

them. For example, if a paper is written by four authors, this is represented by a clique of size four in the network. HepPh, AstroPh, and CondMat networks are constructed based on the papers submitted to the arXiv e-print service. The DBLP network is constructed based on the DBLP computer science bibliography website.

We use five social networks: Flickr, Myspace, LiveJournal, LiveJournal2 (a variation with ground-truth), and Orkut. Flickr is an online photo sharing application, Myspace is a social entertainment networking service, LiveJournal is a blogging application where users can publish their own journals, and Orkut was a social networking website operated by Google.

In the Amazon product network, vertices represent products and edges represent co-purchasing information. If products u and v are frequently co-purchased, there exists an edge between them. This network is constructed based on *Customers Who Bought This Item Also Bought* feature of the Amazon website.

In Table 2.1, we present the number of nodes/edges and the average clustering coefficient (CC) of each of the networks. Figure 2.1 shows the degree distributions of DBLP, Flicker and Amazon networks. We can see that the real-world networks have distinguishing characteristics: a power-law degree distribution [13] and a high clustering coefficient [77], [34].

We have ground-truth communities [4] on some of the datasets. In DBLP,

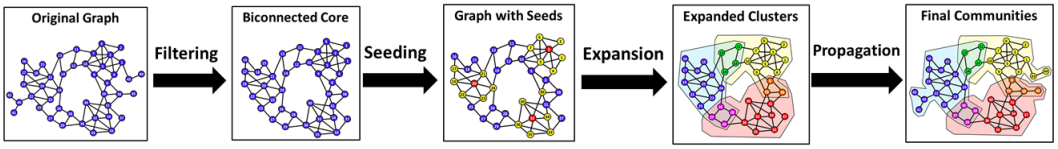


Figure 2.2: Overview of NISE. NISE consists of four main phases: filtering, seeding, seed expansion, and propagation.

each publication venue (i.e., journal or conference) can be considered as an individual ground-truth community. In the Amazon network, each ground-truth community can be defined to be a product category that Amazon provides. In LiveJournal2 and Orkut networks, there exists user-defined social groups. On LiveJournal2 and Orkut networks, the ground-truth communities do not cover a substantial portion of the graph, so we use a subgraph which is induced by the nodes that have at least one membership in the ground-truth communities. In Table 2.1, the statistics about LiveJournal2 and Orkut are based on the induced subgraphs we used in our experiments.

2.2 Overlapping Community Detection Using Neighborhood-Inflated Seed Expansion

We introduce our overlapping community detection algorithm, NISE which consists of four phases: filtering, seeding, seed expansion, and propagation. In the filtering phase, we remove regions of the graph that are trivially separable from the rest of the graph. In the seeding phase, we find good seeds in the filtered graph, and in seed expansion phase, we expand the seeds using a personalized PageRank clustering scheme. Finally, in the propagation phase, we further expand the communities to the regions that were removed in the filtering phase. Figure 2.2 shows the overview of the NISE algorithm.

2.2.1 Filtering Phase

The goal of the filtering phase is to identify regions of the graph where an algorithmic solution is required to identify the overlapping clusters. To explain our filtering step, recall that almost all graph partitioning methods begin by assigning each connected component to a separate partition. Any other choice of partitioning for disconnected components is entirely arbitrary. The Metis procedure [46], for instance, may combine two disconnected components into a single partition in order to satisfy a balance constraint on the partitioning. For the problem of overlapping clustering, an analogous concept can be derived from biconnected components. We now review a series of definitions and formalizations of these ideas in order to analyze our filtering phase and prove new theorems about our propagation phase in Chapter 2.2.4. A biconnected component is defined as follows:

Definition 1. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, a biconnected component is a maximal induced subgraph $G' = (\mathcal{V}', \mathcal{E}')$ that remains connected after removing any vertex and its adjacent edges in G'.*

Let us define the size of a biconnected component to be the number of edges in G'. Now, consider all the biconnected components of size one. Notice that there should be no overlapping partitions that use these edges because they bridge disjoint communities. Consequently, our filtering procedure is to find the largest connected component of the graph after we remove all single-edge biconnected components. We call this the "biconnected core" of the graph even though it may not be biconnected. Let $\mathcal{E}_S$ denote all the single-edge biconnected components. Then, the biconnected core graph is defined as follows:

Definition 2. *The biconnected core $G_C = (\mathcal{V}_C, \mathcal{E}_C)$ is the maximum size connected subgraph of $G'' = (\mathcal{V}, \mathcal{E} \setminus \mathcal{E}_S)$.*

Subgraphs connected to the biconnected core are called *whiskers* by Leskovec

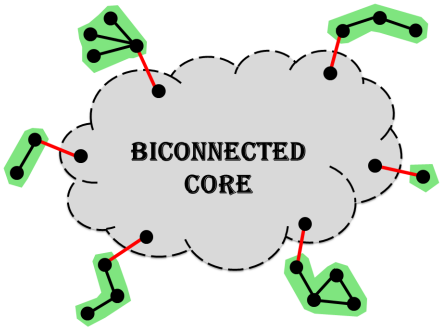


Figure 2.3: Biconnected core, whiskers, and bridges – grey region indicates the biconnected core where vertices are densely connected to each other, and green components indicate whiskers. Red edges indicate bridges which connect the biconnected core and the whiskers.

et al. [53] and we use the concept of a bridge to define them:

Definition 3. *A bridge is a biconnected component of size one which is directly connected to the biconnected core.*

Whiskers are then defined as follows:

Definition 4. *A whisker $W = (\mathcal{V}_W, \mathcal{E}_W)$ is a maximal subgraph of G that can be detached from the biconnected core by removing a bridge.*

Let $\mathcal{E}_B$ be all the bridges in a graph. Notice that $\mathcal{E}_B \subseteq \mathcal{E}_S$. On the region which is not included in the biconnected core graph G_C, we define the detached graph G_D as follows:

Definition 5. *$G_D = (\mathcal{V}_D, \mathcal{E}_D)$ is the subgraph of G which is induced by $\mathcal{V} \setminus \mathcal{V}_C$.*

Finally, given the original graph $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{V}$ and $\mathcal{E}$ can be decomposed as follows:

Proposition 1. *Given a graph $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = \mathcal{V}_C \cup \mathcal{V}_D$ and $\mathcal{E} = \mathcal{E}_C \cup \mathcal{E}_D \cup \mathcal{E}_B$.*

Proof. This follows from the definitions of the biconnected core, bridges, and the detached graph. □

Table 2.2: Biconnected core and the detached graph (in the last column, LCC refers to the largest connected component).

	Biconnected core		Detached graph	
	No. of vertices (%)	No. of edges (%)	No. of components	Size of the LCC (%)
HepPh	9,945 (88.8%)	116,099 (98.7%)	1,123	21 (0.1874%)
AstroPh	16,829 (94.0%)	195,835 (99.4%)	957	23 (0.1285%)
CondMat	19,378 (90.7%)	89,128 (97.6%)	1,669	12 (0.0562%)
DBLP	264,341 (83.4%)	991,125 (94.4%)	43,093	32 (0.0101%)
Amazon	291,449 (87.0%)	862,836 (93.2%)	25,835	250 (0.0747%)
Flickr	954,672 (47.9%)	20,390,649 (95.1%)	864,628	107 (0.0054%)
Myspace	1,724,184 (82.7%)	45,096,696 (99.2%)	332,596	32 (0.0015%)
LiveJournal	1,650,851 (93.9%)	42,071,541 (99.7%)	101,038	105 (0.0060%)
LiveJournal2	1,076,499 (94.2%)	16,786,580 (99.4%)	59,877	91 (0.0080%)
Orkut	729,634 (99.8%)	21,990,221 (99.9%)	1,529	15 (0.0021%)

Figure 2.3 illustrates the biconnected core, whiskers, and bridges. The output of our filtering phase is the biconnected core graph where whiskers are filtered out (we remove regions that are clearly partitionable from the remainder). Note that there is no overlap between any of the whiskers. This indicates that there is no need to apply overlapping community detection algorithm on the detached regions.

Table 2.2 shows the size of the biconnected core and the connectivity of the detached graph in our real-world networks. Details of these networks are presented in Table 2.1. We compute the size of the biconnected core in terms of the number of vertices and edges. The number reported in the parenthesis shows how many vertices or edges are included in the biconnected core, i.e., the percentages of $|\mathcal{V}_C|/|\mathcal{V}|$ and $|\mathcal{E}_C|/|\mathcal{E}|$, respectively. We also compute the number of connected components in the detached graph, and the size of the largest connected component (LCC in Table 2.2) in terms of the number of vertices. The number reported in the parenthesis indicates the relative size of the largest connected component compared to the number of vertices in the original graph.

We can see that the biconnected core contains a substantial portion of the edges. In terms of the vertices, the biconnected core contains around 80 or 90 percentage of the vertices for all datasets except Flickr. In Flickr, the biconnected core

only contains around 50 percentage of the vertices while it contains 95 percentage of edges. This indicates that the biconnected core is dense while the detached graph is quite sparse. Recall that the biconnected core is one connected component. On the other hand, in the detached graph, there are many connected components, which implies that the vertices in the detached graph are likely to be disconnected with each other. Notice that each connected component in the detached graph corresponds to a whisker. So, the largest connected component can be interpreted as the largest whisker. Based on the statistics of the detached graph, we can see that whiskers tend to be separable from each other, and there are no significant size whiskers. Also, the gap between the sizes of the biconnected core and the largest whisker is significant. All these statistics and observations support that our filtering phase creates a reasonable and more tractable input for an overlapping community detection algorithm.

2.2.2 Seeding Phase

Once we obtain the biconnected core graph, we find seeds in this filtered graph. The goal of an effective seeding strategy is to identify a diversity of vertices, each of which lies within a cluster of good conductance. This identification should not be too computationally expensive.

Graclus Centers. One way to achieve these goals is to first apply a high quality and fast graph partitioning scheme (disjoint clustering of vertices in a graph) in order to compute a collection of sets with fairly small conductance. Then, we select a set of seeds by picking the most central vertex from each set (cluster). The idea here is roughly that we want something that is close to the partitioning – which ought to be good – but that allows overlap to produce better boundaries between the partitions.

See Algorithm 1 for the full procedure. In practice, we perform top-down

Algorithm 1 Seeding by Graclus Centers

Input: graph G, the number of seeds k.
Output: the seed set $\mathcal{S}$.

1: Compute exhaustive and non-overlapping clusters $\mathcal{C}_i$ (i=1, ..., k) on G.
2: Initialize $\mathcal{S} = \emptyset$.
3: **for** each cluster $\mathcal{C}_i$ **do**
4: **for** each vertex $v \in \mathcal{C}_i$ **do**
5: Compute $\texttt{dist}(v, \mathcal{C}_i)$ using (2.4).
6: **end for**
7: $\mathcal{S} = \{\underset{v}{\text{argmin}}\ \texttt{dist}(v, \mathcal{C}_i)\} \cup \mathcal{S}$.
8: **end for**

Algorithm 2 Seeding by Spread Hubs

Input: graph $G = (\mathcal{V}, \mathcal{E})$, the number of seeds k.
Output: the seed set $\mathcal{S}$.

1: Initialize $\mathcal{S} = \emptyset$.
2: All vertices in $\mathcal{V}$ are unmarked.
3: **while** $|\mathcal{S}| < k$ **do**
4: Let $\mathcal{T}$ be the set of unmarked vertices with max degree.
5: **for each** $t \in \mathcal{T}$ **do**
6: **if** t is unmarked **then**
7: $\mathcal{S} = \{t\} \cup \mathcal{S}$.
8: Mark t and its neighbors.
9: **end if**
10: **end for**
11: **end while**

hierarchical clustering using Graclus [32] to get a large number of clusters. Then, we take the center of each cluster as a seed – the center of a cluster is defined to be the vertex that is closest to the cluster centroid (as discussed in Chapter 2.1.3, we can quantify the distance between a vertex and a cluster centroid by using the kernel that underlies the relationship between kernel k-means and graph clustering); see steps 5 and 7 in Algorithm 1. If there are several vertices whose distances are tied for the center of a cluster, we include all of them.

Spread Hubs. From another viewpoint, the goal is to select a set of well-distributed seeds in the graph, such that they will have high coverage after we

expand the sets. We greedily choose an independent set of k points in the graph by looking at vertices in order of decreasing degree. For this heuristic, we draw inspiration from the distance function (2.4), which shows that the distance between a vertex and a cluster is inversely proportional to degree. Thus, high degree vertices are expected to have small distances to many other vertices. This also explains why we call the method *spread hubs*. It also follows from the recent results in [37], [82] which show that there should be good clusters around high degree vertices in power-law graphs with high clustering coefficients. We use an independent set in order to avoid picking seeds nearby each other.

Our full procedure is described in Algorithm 2. In the beginning, all the vertices are unmarked. Until k seeds are chosen, the following procedure is repeated: among unmarked vertices, the highest degree vertex is selected as a seed, and then the selected vertex and its neighbors are marked. As the algorithm proceeds exploring hubs in the network, if there are several vertices whose degrees are the same, we take an independent set of those that are unmarked. This step may result in more than k seeds, however, the final number of returned seeds typically does not exceed the input k too much because there usually are not too many high degree vertices.

2.2.3 Seed Expansion Phase

Once we have a set of seed vertices, we wish to expand the clusters around those seeds. An effective technique for this task is using a personalized PageRank (PPR) vector [64], also known as a random-walk with restart [66]. A personalized PageRank vector is the stationary distribution of a random walk that, with probability α follows a step of a random walk and with probability $(1-\alpha)$ jumps back to a seed node. If there are multiple seed nodes, then the choice is usually uniformly random. Thus, nodes close by the seed are more likely to be visited. Recently, such techniques have been shown to produce communities that best match communities

Algorithm 3 Seed Expansion by PPR

Input: graph $G = (\mathcal{V}, \mathcal{E})$, a seed node $s \in \mathcal{S}$, PageRank link-following probability parameter $0 < \alpha < 1$, accuracy $\varepsilon > 0$

Output: low conductance set $\mathcal{C}$

1: Set $\mathcal{T} = \{s\} \cup \{\text{neighbors of s}\}$
2: Initialize $x_v = 0$ for $v \in \mathcal{V}$
3: Initialize $r_v = 0$ for $v \in \mathcal{V} \setminus \mathcal{T}$, $r_v = 1/|\mathcal{T}|$ for $v \in \mathcal{T}$
4: **while** any $r_v > \mathtt{deg}(v)\varepsilon$ **do**
5: Update $x_v = x_v + (1 - \alpha)r_v$.
6: For each $(v, u) \in \mathcal{E}$,
update $r_u = r_u + \alpha r_v/(2\,\mathtt{deg}(v))$
7: Update $r_v = \alpha r_v/2$
8: **end while**
9: Sort vertices by decreasing $x_v/\,\mathtt{deg}(v)$
10: For each prefix set of vertices in the sorted list, compute the conductance of that set and set $\mathcal{C}$ to be the set that achieves the minimum.

found in real-world networks [5]. In fact, personalized PageRank vectors have close relationships to graph cuts and clustering methods. Andersen et al. [7] show that a particular algorithm to compute a personalized PageRank vector, followed by a sweep over all cuts induced by the vector, will identify a set of good conductance within the graph. They prove this via a "localized Cheeger inequality" that states, informally, that the set identified via this procedure has a conductance that is not too far away from the best conductance of any set containing that vertex. Also, Mahoney et al. [59] show that personalized PageRank is, effectively, a seed-biased eigenvector of the Laplacian. They also show a limit to relate the personalized PageRank vectors to the Fiedler vector of a graph.

We briefly summarize the PPR-based seed expansion procedure in Algorithm 3 (each seed is expanded by this procedure). Please see Andersen et al. [7] for a full description of the algorithm. The high level idea of this expansion method is that given a set of restart nodes (denoted by $\mathcal{T}$ in Algorithm 3), we first compute the PPR vector, examine nodes in order of highest to lowest PPR score, and then return the set that achieves the minimum conductance.

It is important to note that we can have multiple nodes in $\mathcal{T}$ (which corresponds to nonzero elements in the personalization vector in PPR), and indeed we use the entire vertex neighborhood of a seed node as the restart nodes (see step 1 in Algorithm 3). Since we do not just use a singleton seed but also use its neighbors as the restart nodes in PPR, we call step 1 *neighborhood inflation*. We empirically observed that this neighborhood inflation plays a critical role in producing low conductance communities. See Chapter 2.4 for details. Recently, Gleich and Seshadhri [37] have provided some theoretical justification for why neighborhood-inflated seeds may outperform a singleton seed in PPR expansion on many real-world networks.

Steps 2-8 are closely related to a coordinate descent optimization procedure [18] on the PageRank linear system. Although it may not be apparent from the procedure, this algorithm is remarkably efficient when combined with the appropriate data structures. The algorithm keeps two vectors of values for each vertex, $\mathbf{x}$ and $\mathbf{r}$. In a large graph, most of these values will remain zero on the vertices and hence, these need not be stored. Our implementation uses a hash table for the vectors $\mathbf{x}$ and $\mathbf{r}$. Consequently, the sorting step is only over a small fraction of the total vertices.

In the original PPR clustering [7], the PPR score is divided by the degree of each node (step 9) to remove bias towards high degree nodes. This step converts a PageRank vector, a left eigenvector of a Markov chain, into the right eigenvector of a Markov chain. Right eigenvectors are close relatives of the Fiedler vector of a graph, and so this degree normalization produces a vector that we call the *Fiedler Personalized PageRank vector* because of this relationship. Fiedler vectors also satisfy Cheeger inequalities, just like the Fiedler Personalized PageRank vectors. However, Kloumann and Kleinberg [47] recently reported that this degree normalization might slightly degrade the quality of the output clusters in terms of matching with ground-truth communities in some real-world networks. So, in our

experiments, we also try using the PPR score which we just call *PPR*. We compare the performance of the Fiedler PPR and PPR in Chapter 2.4.

In Algorithm 3, there are two parameters which are related to PPR computation: α and ε. We follow standard practice for PPR clustering on an undirected graph and set $\alpha = 0.99$ [53]. This value yields results that are similar to those without damping, yet have bounded computational time. The parameter ε is an accuracy parameter. As $\varepsilon \to 0$, the final vector solution $\mathbf{x}$ tends to the exact solution of the PageRank linear system. When used for clustering, however, this parameter controls the effective *size* of the final cluster. If ε is large (about 10^{-2}), then the output vector is inaccurate, incredibly sparse, and the resulting cluster is small. If ε is small, say 10^{-8}, then the PageRank vector is accurate, nearly dense, and the resulting cluster may be large. We thus run the PPR clustering scheme several times, with a range of accuracy parameters that are empirically designed to produce clusters with between 1 and 50,000 times the number of edges in the initial seed set (these values of ε are fixed and independent of the graph). The final community we select is the one with the best conductance score from these possibilities.

2.2.4 Propagation Phase

Once we get the personalized PageRank communities on the biconnected core, we further expand each of the communities to the regions detached in the filtering phase. Our assignment procedure is straightforward: for each detached whisker connected via a bridge, we add that piece to all of the clusters that utilize the other vertex in the bridge. This procedure is described in Algorithm 4. We show that our propagation procedure only improves the quality of the final clustering result in terms of the normalized cut metric. To do this, we need to fix some notation. Let $\mathcal{E}_{B_i}$ be a set of bridges which are attached to $\mathcal{C}_i$, and $W_{\mathcal{C}_i}$ be a set of

Algorithm 4 Propagation Procedure

Input: graph $G = (\mathcal{V}, \mathcal{E})$, biconnected core $G_C = (\mathcal{V}_C, \mathcal{E}_C)$, communities of $G_C : \mathcal{C}_i$ $(i = 1, ..., k) \in \mathcal{C}$.

Output: communities of G.

```
for each C_i ∈ C do
    Detect bridges E_{B_i} attached to C_i.
    for each b_j ∈ E_{B_i} do
        Detect the whisker w_j = (V_j, E_j) which is attached to b_j.
        C_i = C_i ∪ V_j.
    end for
end for
```

whiskers which are attached to the bridges, i.e., $W_{\mathcal{C}_i} = (\mathcal{V}_{W_i}, \mathcal{E}_{W_i})$ where

$$w_j = (\mathcal{V}_j, \mathcal{E}_j) \in W_{\mathcal{C}_i}; \;\; \mathcal{V}_{W_i} = \bigcup_{w_j \in W_{\mathcal{C}_i}} \mathcal{V}_j; \;\; \text{and} \;\; \mathcal{E}_{W_i} = \bigcup_{w_j \in W_{\mathcal{C}_i}} \mathcal{E}_j.$$

Finally, let $\mathcal{C}_i'$ denote the expanded $\mathcal{C}_i$, where $|\mathcal{C}_i'| \geq |\mathcal{C}_i|$. Equality holds in this expression when there is no bridge attached to $\mathcal{C}_i$. When we expand $\mathcal{C}_i$ using Algorithm 4, $\mathcal{C}_i'$ is equal to $\{\mathcal{C}_i \bigcup \mathcal{V}_{W_i}\}$. The following results show that we only decrease the (normalized) cut by adding the whiskers.

Theorem 1. *If a community $\mathcal{C}_i$ is expanded to $\mathcal{C}_i'$ using Algorithm 4, $\texttt{cut}(\mathcal{C}_i') = \texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)$.*

Proof. Recall that $\texttt{cut}(\mathcal{C}_i)$ is defined as follows:

$$\begin{aligned} \texttt{cut}(\mathcal{C}_i) &= \texttt{links}(\mathcal{C}_i, \mathcal{V} \setminus \mathcal{C}_i). \\ &= \texttt{links}(\mathcal{C}_i, \mathcal{V}) - \texttt{links}(\mathcal{C}_i, \mathcal{C}_i). \end{aligned}$$

Let us first consider $\texttt{links}(\mathcal{C}_i', \mathcal{V})$ as follows:

$$\texttt{links}(\mathcal{C}_i', \mathcal{V}) = \texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i). \tag{2.5}$$

Notice that $\texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}) = \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)$ by definition of

whiskers. Thus, $\texttt{links}(\mathcal{C}'_i, \mathcal{V})$ can be expressed as follows:

$$\texttt{links}(\mathcal{C}'_i, \mathcal{V}) = \texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}). \tag{2.6}$$

On the other hand, $\texttt{links}(\mathcal{C}'_i, \mathcal{C}'_i)$ can be expressed as:

$$\texttt{links}(\mathcal{C}'_i, \mathcal{C}'_i) = \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i}) + \texttt{links}(\mathcal{C}_i, \mathcal{C}_i) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i). \tag{2.7}$$

Now, let us compute $\texttt{cut}(\mathcal{C}'_i)$ which is defined by

$$\texttt{cut}(\mathcal{C}'_i) = \texttt{links}(\mathcal{C}'_i, \mathcal{V}) - \texttt{links}(\mathcal{C}'_i, \mathcal{C}'_i). \tag{2.8}$$

By rewriting (2.8) using (2.6) and (2.7), we can express $\texttt{cut}(\mathcal{C}'_i)$ as follows:

$$\texttt{cut}(\mathcal{C}'_i) = \texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i). \qquad \square$$

Theorem 2. *If a community $\mathcal{C}_i$ is expanded to $\mathcal{C}'_i$ using Algorithm 4, $\texttt{NCut}(\mathcal{C}'_i) \leq \texttt{NCut}(\mathcal{C}_i)$.*

Proof. Recall that

$$\texttt{NCut}(\mathcal{C}_i) = \frac{\texttt{cut}(\mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V})}.$$

On the other hand, by Theorem 1, we can represent $\texttt{NCut}(\mathcal{C}'_i)$ as follows:

$$\begin{aligned} \texttt{NCut}(\mathcal{C}'_i) &= \frac{\texttt{cut}(\mathcal{C}'_i)}{\texttt{links}(\mathcal{C}'_i, \mathcal{V})}. \\ &= \frac{\texttt{cut}(\mathcal{C}_i) - \texttt{links}(\mathcal{V}_{W_i}, \mathcal{C}_i)}{\texttt{links}(\mathcal{C}_i, \mathcal{V}) + \texttt{links}(\mathcal{V}_{W_i}, \mathcal{V}_{W_i})}. \end{aligned}$$

Therefore, $\texttt{NCut}(\mathcal{C}'_i) \leq \texttt{NCut}(\mathcal{C}_i)$. Equality holds when there is no bridge attached to $\mathcal{C}_i$, i.e., $\mathcal{E}_{B_i} = \emptyset$. $\square$

Table 2.3 shows the average normalized cut values before and after the prop-

Table 2.3: Average normalized cut values before & after propagation. Theorem 2 shows this should decrease.

Graph	Before Propagation	After Propagation
HepPh	0.1383	0.1282
AstroPh	0.1764	0.1728
CondMat	0.1841	0.1717
DBLP	0.2329	0.2035
Amazon	0.1356	0.1159

agation phase. As predicted by the theorem, these values decrease after the propagation on this set of graphs.

2.3 Related Work

For overlapping community detection, many different approaches have been proposed [83] including clique percolation, line graph partitioning, eigenvector methods, ego network analysis, and low-rank models. Clique percolation methods look for overlap between fixed size cliques in the graph [65]. Line graph partitioning is also known as link communities. Given a graph $G = (\mathcal{V}, \mathcal{E})$, the line graph of $L(G)$ (also called the dual graph) has a vertex for each edge in G and an edge whenever two edges (in G) share a vertex. For instance, the line graph of a star is a clique. A partitioning of the line graph induces an overlapping clustering in the original graph [6]. Even though these clique percolation and line graph partitioning methods are known to be useful for finding meaningful overlapping structures, these methods often fail to scale to large networks like those we consider.

Eigenvector methods generalize spectral methods and use a soft clustering scheme applied to eigenvectors of the normalized Laplacian or modularity matrix in order to estimate communities [86]. Ego network analysis methods use the theory of structural holes [23], and compute and combine many communities through manipulating ego networks [71], [30]. We compare against the Demon method [30] that

uses this strategy. We also note that other low-rank methods such as non-negative matrix factorizations identify overlapping communities as well. We compare against the Bigclam method [85] that uses this approach.

The approach we employ is called local optimization and expansion [83]. Starting from a seed, such a method greedily expands a community around that seed until it reaches a local optima of the community detection objective. Determining how to seed a local expansion method is, arguably, a critical problem within these methods. Strategies to do so include using maximal cliques [72], prior information [36], or locally minimal neighborhoods [37]. The latter method was shown to identify the vast majority of good conductance sets in a graph; however, there was no provision made for total coverage of all vertices.

Different optimization objectives and expansion methods can be used in a local expansion method. For example, Oslom [50] tests the statistical significance of clusters with respect to a random configuration during community expansion. Starting from a randomly picked node, the Oslom method greedily expands the cluster by checking whether the expanded community is statistically significant or not, which results in detecting a set of overlapping clusters and outliers in a graph. We compare our method with the Oslom method in our experiments (see Chapter 2.4).

In our algorithm, we use a personalized PageRank based cut finder [7] for the local expansion method. Abrahao et al. [5] observe that the structure of real-world communities can be well captured by the random-walk-based algorithms, i.e., personalized PageRank clusters are topologically similar to real-world clusters. More recently, Kloumann and Kleinberg [47] propose to use pure PageRank scores instead of the Fiedler PageRank scores to get a higher accuracy in terms of matching with ground-truth communities.

2.4 Experimental Results

We compare our algorithm, NISE, with other state-of-the-art overlapping community detection methods: Bigclam [85], Demon [30], and Oslom [50]. For these three methods, we used the software which is provided by the authors of [85], [30], and [50] respectively. While Demon and Oslom only support a sequential execution, Bigclam supports a multi-threaded execution. NISE is written in a mixture of C++ and MATLAB. In NISE, seeds can be expanded in parallel, and this feature is implemented using parallel computing toolbox provided by MATLAB. We compare the performance of each of these methods on ten different real-world networks which are presented in Chapter 2.1.4. Within NISE, we also compare the performance of different seeding strategies and some variants of expansion methods. We use four different seeding strategies: "graclus centers" (denoted by "nise-grc-*") and "spread hubs" (denoted by "nise-sph-*") which are proposed in this chapter, "locally minimal neighborhoods" (denoted by "nise-lcm-*") which has been proposed in [37], and random seeding strategy (denoted by "nise-rnd-*") where we randomly take k seeds. Andersen and Lang [8] have provided some theoretical justification for why random seeding also should be competitive. We also compare two different expansion methods: the Fiedler Personalized PageRank (denoted by "nise-*-fppr"), and the standard Personalized PageRank (denoted by "nise-*-ppr").

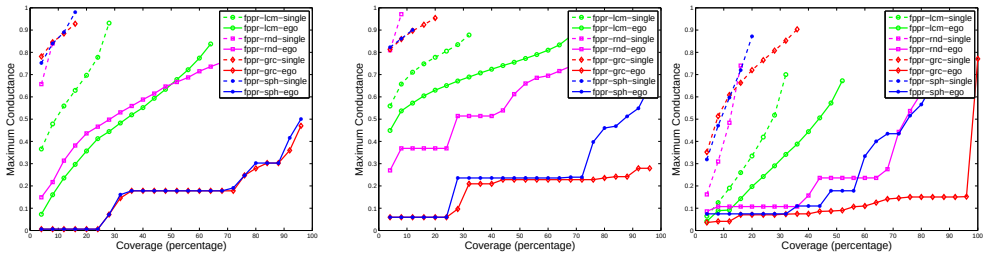
2.4.1 Graph Coverage

We first report the returned number of clusters and the graph coverage of each algorithm in Table 2.4. The graph coverage indicates how many vertices are assigned to clusters (i.e., the number of assigned vertices divided by the total number of vertices in a graph). Note that we can control the number of seeds k in NISE and the number of clusters k in Bigclam. We set k (in our methods and Bigclam) as 100 for HepPh, 200 for AstroPh and CondMat, 15,000 for Flickr, Myspace, and

Table 2.4: Returned number of clusters and graph coverage of each algorithm

Graph		oslom	demon	bigclam	nise-sph-fppr	nise-grc-fppr
HepPh	coverage (%)	100	88.83	84.37	100	100
	no. of clusters	608	5,147	100	99	90
AstroPh	coverage (%)	100	94.15	91.11	100	100
	no. of clusters	1,241	8,259	200	212	246
CondMat	coverage (%)	100	91.16	99.96	100	100
	no. of clusters	1,534	10,474	200	201	249
Flickr	coverage (%)	N/A	N/A	52.13	93.60	100
	no. of clusters	N/A	N/A	15,000	15,349	16,347
LiveJournal	coverage (%)	N/A	N/A	43.86	99.78	99.79
	no. of clusters	N/A	N/A	15,000	15,058	16,271
Myspace	coverage (%)	N/A	N/A	N/A	99.87	100
	no. of clusters	N/A	N/A	N/A	15,324	16,366
DBLP	coverage (%)	100	84.89	100	100	100
	no. of clusters	17,519	174,560	25,000	26,503	18,477
Amazon	coverage (%)	100	79.16	100	100	100
	no. of clusters	17,082	105,685	25,000	27,763	20,036
Orkut	coverage (%)	N/A	N/A	82.13	99.99	100
	no. of clusters	N/A	N/A	25,000	25,204	32,622
LiveJournal2	coverage (%)	N/A	N/A	56.64	99.95	99.99
	no. of clusters	N/A	N/A	25,000	25,065	32,274

LiveJournal, and 25,000 for DBLP, Amazon, LiveJournal2, and Orkut networks without any tuning and using the guidance that larger graphs can have more clusters (Chapter 2.4.6 discusses varying k). For the networks where we have ground-truth communities, we slightly overestimate the number of clusters k since there usually exists a large number of ground-truth communities. Since we remove duplicate clusters after the PageRank expansion in NISE, the returned number of clusters can be smaller than k. Also, since we choose all the tied seeds in "graclus centers" and "spread hubs", the returned number of clusters of these algorithms can be slightly larger than k. Recall that we use a top-down hierarchical clustering scheme in the "graclus centers" strategy. So, in this case, the returned number of clusters



(a) LiveJournal (b) Myspace (c) Flickr

Figure 2.4: Importance of neighborhood inflation – there is a large performance gap between singleton seeds and neighborhood-inflated seeds for all the seeding strategies. Neighborhood inflation plays a critical role in the success of NISE. When neighborhood-inflated seeds are used, "graclus centers" and "spread hubs" seeding strategies significantly outperform other seeding strategies.

before filtering the duplicate clusters is slightly greater than or equal to $2^{\lceil \log k \rceil}$. Demon and Oslom determine the number of clusters based on datasets themselves, although these methods fail on Flickr, Myspace, LiveJournal, LiveJournal2, and Orkut. Bigclam does not finish on the Myspace network (using 4 threads) after running for one week.

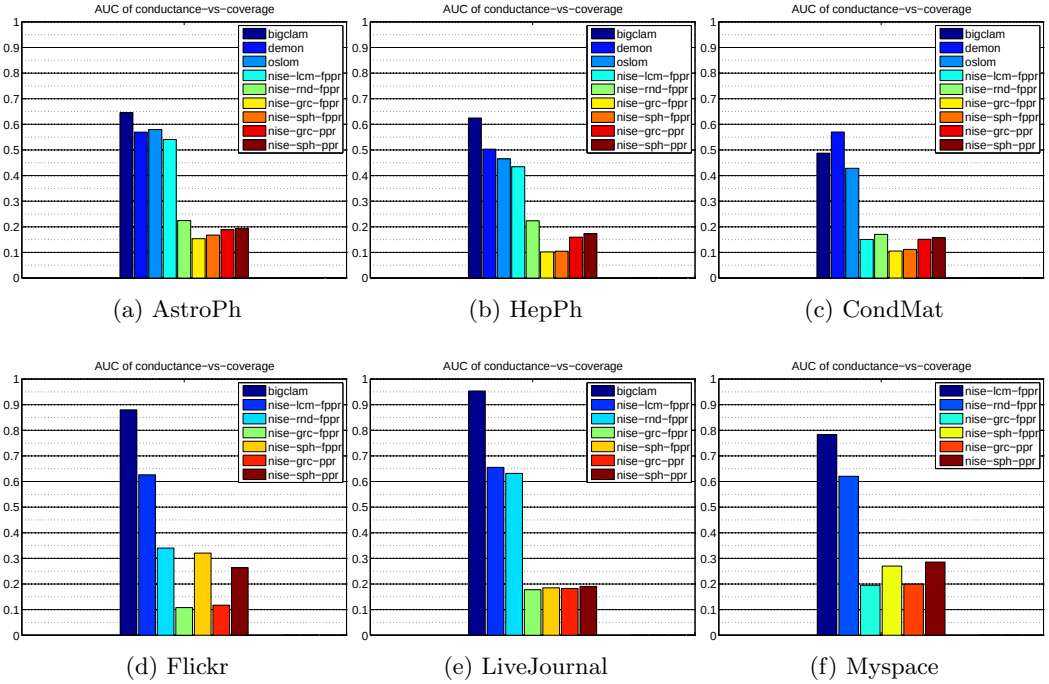
2.4.2 Importance of Neighborhood-Inflation

We evaluate the quality of overlapping communities in terms of the maximum conductance of any cluster. A high quality algorithm should return a set of clusters that covers a large portion of the graph with small maximum conductance. This metric can be captured by a conductance-vs-coverage curve. That is, for each method, we first sort the clusters according to the conductance scores in ascending order, and then greedily take clusters until a certain percentage of the graph is covered. The x-axis of each plot is the graph coverage, and the y-axis is the maximum conductance value among the clusters we take. We can interpret this plot as follows: we need to use clusters whose conductance scores are less than or equal to y to cover x percentage of the graph. Note that lower conductance indicates better quality of

clusters, i.e., a lower curve indicates better clusters.

First, we verify the importance of *neighborhood inflation* in our seed expansion phase. Recall that when we compute the personalized PageRank (PPR) score for each seed node, we use the seed node's entire vertex neighborhood (the vertex neighborhood is also referred to as "ego network") as the restart region in PPR (details are in Chapter 2.2.3). To see how this affects the overall performance of the seed expansion method, we compare the performance of singleton seeds and neighborhood-inflated seeds. Figure 2.4 shows the conductance-vs-coverage plot for singleton seeds and neighborhood-inflated seeds. "*-single" indicates singleton seeds, i.e., each seed is solely used as the restart region in PPR. "*-ego" indicates neighborhood-inflated seeds. We also used four different seeding strategies: "graclus centers" (denoted by "grc-*"), "spread hubs" (denoted by "sph-*"), "locally minimal neighborhoods" (denoted by "lcm-*"), and "random" (denoted by "rnd-*").

We can see that the performance significantly degrades when singleton seeds are used for all the seeding strategies. This implies that neighborhood inflation plays a critical role in the success of our method. Even though we only present the results on LiveJournal, Myspace, and Flickr in Figure 2.4 for brevity, we consistently observed that neighborhood-inflated seeds are much better than singleton seeds on all other networks. We also notice that that when neighborhood-inflated seeds are used, both "graclus centers" and "spread hubs" seeding strategies significantly outperform other seeding strategies. "spread hubs" and "graclus centers" seeding strategies produce similar results on LiveJournal whereas "graclus centers" is better than "spread hubs" on Myspace and Flickr. We used the conventional Fiedler PPR for the expansion phase in Figure 2.4, but we also got the same conclusion using the standard PPR.



(a) AstroPh (b) HepPh (c) CondMat

(d) Flickr (e) LiveJournal (f) Myspace

Figure 2.5: AUC of Conductance-vs-coverage – lower bar indicates better communities. NISE outperforms Demon, Oslom, and Bigclam. Within NISE, "graclus centers" and "spread hubs" seeding strategies are better than other seeding strategies, and the Fiedler PPR produces slightly better communities than the standard PPR.

2.4.3 Community Quality Using Conductance

We compute AUC (Area Under the Curve) of the conductance-vs-coverage to compare the performance of NISE with other state-of-the-art methods. Within NISE, we also compare four different seeding strategies and two different expansion methods. The AUC scores are normalized such that they are between zero and one. Figure 2.5 shows AUC scores on the six networks where we do not have ground-truth community information (see Table 2.1 for details about these networks). We can see several patterns in Figure 2.5. First, within NISE, "graclus centers" and "spread hubs" seeding strategies outperform the other two seeding strategies. Second, for most of the cases, "fppr" leads to slightly better communities than "ppr". Also, we

can see that "nise-grc-fppr" shows the best performance for all networks. Third, NISE outperforms Demon, Oslom, and Bigclam. There is a significant performance gap between NISE and these methods.

We also compared NISE with a non-overlapping clustering to study the benefit of overlap. If we use the clusters produced by Graclus [32], a graph partitioning method which is used within our "graclus centers" seeding strategy, we observed that NISE also significantly outperforms Graclus on all the networks. For example, in terms of conductance measure, the AUC of Graclus is 0.5309 while the AUC of NISE is 0.1019. Due to the large performance gap between these clusterings, we omit more comprehensive numerical evaluation.

2.4.4 Community Quality via Ground-truth

We have ground-truth communities for the DBLP, Amazon, LiveJournal2, and Orkut networks, thus, for these networks, we compare against the ground-truth communities. Given a set of algorithmic communities $\mathcal{C}$ and the ground-truth communities $\mathcal{S}$, we compute the F_1 measure and the F_2 measure to evaluate the relevance between the algorithmic communities and the ground-truth communities. In general, F_β measure is defined as follows:

$$F_\beta(\mathcal{S}_i) = (1+\beta^2)\frac{\mathit{precision}(\mathcal{S}_i)\cdot \mathit{recall}(\mathcal{S}_i)}{\beta^2\cdot \mathit{precision}(\mathcal{S}_i) + \mathit{recall}(\mathcal{S}_i)}$$

where β is a non-negative real value, and the *precision* and *recall* of $\mathcal{S}_i \in \mathcal{S}$ are defined as follows:

$$\mathit{precision}(\mathcal{S}_i) = \frac{|\mathcal{C}_j \bigcap \mathcal{S}_i|}{|\mathcal{C}_j|},$$

$$\mathit{recall}(\mathcal{S}_i) = \frac{|\mathcal{C}_j \bigcap \mathcal{S}_i|}{|\mathcal{S}_i|},$$

Table 2.5: F_1 and F_2 measures. NISE with "spread hubs" seeding strategy achieves the highest F_1 and F_2 scores.

	DBLP		Amazon		LiveJournal2		Orkut	
	F_1	F_2	F_1	F_2	F_1	F_2	F_1	F_2
bigclam	15.1%	13.0%	27.1%	25.6%	11.3%	13.7%	43.0%	47.4%
demon	13.7%	12.0%	16.5%	15.3%	N/A	N/A	N/A	N/A
oslom	13.4%	11.6%	32.0%	30.2%	N/A	N/A	N/A	N/A
nise-lcm-fppr	13.9%	15.4%	46.3%	56.5%	11.3%	13.8%	40.9%	46.8%
nise-rnd-fppr	17.7%	20.5%	48.9%	58.8%	12.1%	16.5%	54.6%	62.9%
nise-sph-fppr	18.1%	21.4%	49.2%	**59.5%**	12.7%	**18.1%**	55.1%	64.2%
nise-sph-ppr	**19.0%**	**22.6%**	**49.7%**	58.7%	**12.8%**	**18.1%**	**57.4%**	**65.2%**
nise-grc-fppr	17.6%	21.7%	46.7%	57.1%	12.2%	17.6%	51.1%	61.4%
nise-grc-ppr	17.6%	22.0%	47.3%	56.0%	**12.8%**	17.6%	53.5%	62.4%

where $\mathcal{C}_j \in \mathcal{C}$, and $F_\beta(\mathcal{S}_i) = F_\beta(\mathcal{S}_i, \mathcal{C}_{j^*})$ where $j^* = \text{argmax}_j \; F_\beta(\mathcal{S}_i, \mathcal{C}_j)$. Then, the average F_β measure is defined to be

$$\bar{F_\beta} = \frac{1}{|\mathcal{S}|} \sum_{\mathcal{S}_i \in \mathcal{S}} F_\beta(\mathcal{S}_i).$$

Given an algorithmic community, *precision* indicates how many vertices are actually in the same ground-truth community. Given a ground-truth community, *recall* indicates how many vertices are predicted to be in the same community in a retrieved community. By definition, the precision and the recall are evenly weighted in F_1 measure. On the other hand, the F_2 measure puts more emphasis on recall than precision. The authors in [85] who provided the datasets argue that it is important to quantify the recall since the ground-truth communities in these datasets are partially annotated, i.e., some vertices are not annotated to be a part of the ground-truth community even though they actually belong to that community. This indicates that it would be reasonable to weight recall higher than precision, which is done by the F_2 measure.

In Table 2.5, we report the average F_1 and F_2 measures on DBLP, Amazon, LiveJournal2, and Orkut networks. A higher value indicates better communities.

Table 2.6: Running times of different methods on our test networks.

Graph	oslom	demon	bigclam	nise-sph-fppr	nise-grc-fppr
HepPh	19 mins. 16 secs.	27 secs.	11 mins. 23 secs.	22 secs.	2 mins. 48 secs.
AstroPh	38 mins. 3 secs.	42 secs.	48 mins. 1 secs.	36 secs.	2 mins. 26 secs.
CondMat	20 mins. 39 secs.	50 secs.	7 mins. 21 secs.	36 secs.	1 min. 14 secs.
DBLP	5 hrs. 50 mins.	3 hrs. 53 mins.	7 hrs. 13 mins.	18 mins. 20 secs.	29 mins. 44 secs.
Amazon	2 hrs. 55 mins.	1 hr. 55 mins.	1 hr. 25 mins.	37 mins. 36 secs.	42 mins. 43 secs.
Flickr	N/A	N/A	69 hrs. 59 mins.	43 mins. 55 secs.	3 hrs. 56 mins.
Orkut	N/A	N/A	13 hrs. 48 mins.	1 hrs. 16 mins.	4 hrs. 16 mins.
LiveJournal	N/A	N/A	65 hrs. 30 mins.	2 hrs. 36 mins.	4 hrs. 48 mins.
LiveJournal2	N/A	N/A	21 hrs. 35 mins.	2 hrs. 15 mins.	6 hrs. 37 mins.
Myspace	N/A	N/A	$>$ 7 days	5 hrs. 27 mins.	9 hrs. 42 mins.

We see that NISE outperforms Bigclam, Demon, and Oslom in terms of both F_1 and F_2 measures on these networks. Within NISE, "spread hubs" seeding is better than "graclus centers" seeding, and the standard PPR is slightly better than the Fiedler PPR in most of the cases. So, we see that the standard PPR is useful for identifying ground-truth communities. This result is also consistent with the recent observations in [47].

2.4.5 Comparison of Running Times

We compare the running times of the different algorithms in Table 2.6. To do a fair comparison, we run the single thread versions of Bigclam and NISE on the HepPh, AstroPh, CondMat, DBLP, and Amazon networks. On larger networks Demon and Oslom fail to complete. So, we switch to the multi-threaded version of Bigclam and NISE with 4 threads for Flickr, Orkut, LiveJournal, LiveJournal2, and MySpace. We see that NISE is the only method which can process the largest dataset (Myspace) in a reasonable time. On small networks (HepPh, AstroPh, and CondMat), "nise-sph-fppr" is faster than Demon, Oslom and Bigclam. On medium size networks (DBLP and Amazon), both "nise-grc-fppr" and "nise-sph-fppr" are faster than other methods. On large networks (Flickr, Orkut, LiveJournal, LiveJournal2, Myspace), NISE is much faster than Bigclam.

Table 2.7: F_1 measures with different numbers of communities.

	DBLP		Amazon	
	k=20,000	k=30,000	k=20,000	k=30,000
bigclam	16.6%	14.0%	31.9%	22.9%
demon	13.7%	13.7%	16.5%	16.5%
oslom	13.4%	13.4%	32.0%	32.0%
nise-sph-fppr	**17.5%**	**18.8%**	**47.6%**	**49.2%**
nise-grc-fppr	17.2%	17.6%	45.6%	46.7%

Table 2.8: AUC of conductance-vs-coverage with different numbers of communities. Lower AUC indicates better communities.

	CondMat		LiveJournal	
	k=150	k=250	k=10,000	k=20,000
oslom	0.4285	0.4285	N/A	N/A
demon	0.5701	0.5701	N/A	N/A
bigclam	0.4762	0.4888	0.9528	0.9535
nise-grc-fppr	**0.1090**	**0.1055**	**0.1839**	**0.1780**
nise-sph-fppr	0.1118	0.1116	0.1890	0.1834

2.4.6 Varying the Number of Communities

We need to specify the number of communities for NISE and Bigclam whereas Demon and Oslom automatically identify the number of communities. Thus, we also conduct experiments using different numbers of communities to ensure that our results are not an extremal case. Table 2.7 shows the F_1 scores of each method with different values of k, the number of communities. The outputs of Demon and Oslom are not affected by different k, but we include these results for reference. Also, Table 2.8 shows the AUC of conductance-vs-coverage with different k. We see that NISE consistently outperforms other methods with a reasonable range of k in terms of both F_1 measures and AUC scores even with untuned values of the number of communities.

2.5 Discussion

We now discuss the results from our experimental investigations. First, we note that NISE is the only method that worked on all of the problems. Also, our method is faster than other state-of-the-art overlapping community detection methods. Perhaps surprisingly, the major difference in cost between using "graclus centers" for the seeds and the other seed choices does not result from the expense of running Graclus. Rather, it arises because the personalized PageRank expansion technique takes longer for the seeds chosen by Graclus. When the PageRank expansion method has a larger input set, it tends to take longer, and the "graclus centers" seeding strategy is likely to produce larger input sets because of the neighborhood inflation and because the central vertices of clusters are likely to be high degree vertices.

We wish to address the relationship between our results and some prior observations on overlapping communities. The authors of Bigclam found that the dense regions of a graph reflect areas of overlap between overlapping communities. By using a conductance measure, we ought to find only these dense regions – however, our method produces much larger communities that cover the entire graph. The reason for this difference is that we use the entire vertex neighborhood as the restart for the personalized PageRank expansion routine. We avoid seeding exclusively inside a dense region by using an entire vertex neighborhood as a seed, which grows the set beyond the dense region. Thus, the communities we find likely capture a combination of communities given by the ego network of the original seed node.

Overall, NISE significantly outperforms other state-of-the-art overlapping community detection methods in terms of run time, cohesiveness of communities, and ground-truth accuracy. Also, our new seeding strategies, "graclus centers" and "spread hubs", are superior than existing methods, thus play an important role in the success of our seed set expansion method.

Chapter 3

A Unified Framework of Non-exhaustive, Overlapping Clustering

The empirical success of the personalized PageRank-based algorithm motivates us to investigate the problem of overlapping community detection in a more principled way. To develop a more principled method, we exploit the connection between community detection and clustering. We revisit the overlapping community detection problem from a clustering perspective. To do that, let us first discuss the traditional clustering problem. The traditional clustering problem involves exhaustively assigning each data point to a single group (or cluster) such that nearby points are also assigned to the same group. When separations between groups are clear and the data does not contain any outliers, then classical methods such as the k-means algorithm may succeed in correctly assigning points to groups in many realistic data models. Now let us revisit the clustering problem from the perspec-

The materials presented in this chapter have been published in [78]. Joyce formulated the problems, developed the algorithms, and conducted experiments. Professor Dhillon and Professor Gleich supervised the work.

tive of real-world data in which groups still exist but may lack clean separations and the data may contain outliers. In this setting, a more reasonable goal is a non-exhaustive, overlapping clustering where a data point may be outside of any cluster, and clusters are allowed to overlap with each other.

There is substantial prior research that has examined both of these problems individually – as would be expected for an area as well studied as clustering. For example, non-exhaustive clustering is highly related to outlier detection in a dataset, which itself has an extensive literature. Regarding overlap, both soft-clustering [17], which only makes probabilistic assignments, and overlapping clustering models are common [11]. Furthermore, many variations of the k-means algorithm have been proposed over the years [44] including recent work that considers overlapping clustering [29], [14], [57]. We discuss the related work in more detail in Chapter 3.3 in the context of the limitations of existing methods and our new contribution. A key difference between our approach and existing ideas is that we treat the issues of non-exhaustiveness and overlap in a unified framework.

The result of our investigations is a novel improvement to the k-means clustering objective that enables a parametric trade-off between a clustering quality measure, overlap among the clusters, and non-exhaustiveness (the number of outliers not assigned to any group). To optimize the new objective, we present a simple iterative algorithm called non-exhaustive, overlapping k-means, or NEO-K-Means in short. Furthermore, by considering an extension to weighted kernel k-means, we can also tackle the problem of non-exhaustive, overlapping graph clustering. In the context of graph clustering, we extend a traditional normalized cut-based graph clustering objective to the non-exhaustive, overlapping setting, and show that this extended graph clustering objective is mathematically equivalent to the weighted kernel NEO-K-Means objective with a specific weight and kernel. This equivalence enables us to apply our NEO-K-Means algorithm to the overlapping community de-

tection problem. Experimental results show that our new objective and algorithm are effective in finding ground-truth clusters; for the case of graphs, we show that our algorithm outperforms state-of-the-art overlapping community detection methods.

3.1 Non-exhaustive, Overlapping k-means

Developing a general purpose clustering algorithm is a challenging task. Even though many different clustering methods have been developed, k-means [55] is still one of the most popular clustering techniques due to its simplicity and empirical success [44]. We begin our discussion by briefly reviewing k-means. We then attempt to formulate an obvious extension of the k-means objective function. However, this obvious extension has serious limitations; after recognizing this, we propose our final objective function and a simple iterative algorithm for non-exhaustive, overlapping clustering.

3.1.1 k-means Clustering

Let us review the standard k-means setting. Given a set of data points $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, k-means seeks a partition of the data points into k clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ such that they cover all points (formally, $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_k = \mathcal{X}$), and the partitions are disjoint ($\mathcal{C}_i \cap \mathcal{C}_j = \emptyset \ \forall i \neq j$). The goal of k-means is to pick the clusters to minimize the distance from the cluster centroid, or the mean of cluster, to each of its assigned data points. The k-means objective may be written as:

$$\min_{\{\mathcal{C}_j\}_{j=1}^k} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_j} \|\mathbf{x}_i - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{\mathbf{x}_i \in \mathcal{C}_j} \mathbf{x}_i}{|\mathcal{C}_j|}. \tag{3.1}$$

It has been shown that minimizing the above objective function is an NP-hard problem even for just two clusters. However, there is an efficient heuristic k-means algorithm [55], also known as Lloyd's algorithm, that proceeds by repeatedly

assigning data points to their closest clusters and recomputing cluster centroids. This algorithm monotonically decreases the objective function.

3.1.2 An Intuitive, but Problematic, Extension

To extend the k-means objective function to a non-exhaustive, overlapping clustering setting, we first introduce an assignment matrix $U = [u_{ij}]_{n \times k}$ such that $u_{ij} = 1$ if $\mathbf{x}_i$ belongs to cluster j; $u_{ij} = 0$ otherwise. Using this notation, if we seek a traditional disjoint and exhaustive clustering, the number of ones in the assignment matrix U should be always equal to n because each data point should be assigned to exactly one cluster.

On the other hand, in a non-exhaustive, overlapping clustering, there are no restrictions on the assignment matrix U; there can be multiple ones in a row, meaning that a data point can belong to multiple clusters. Also, there can be rows of all zeros, meaning that some data points can have no membership in any cluster. Thus, we need to decide how many assignments we will make in U, i.e., we need to control the number of ones in U. One way to do this is to consider $U^T U$ which is a $k \times k$ matrix whose diagonal entries are equal to cluster sizes. The trace of $U^T U$ is equal to the sum of cluster sizes which is also equal to the total number of assignments in the assignment matrix U. To control how many additional assignments we will make in U, we add a constraint that the number of total assignments in U should be equal to $n + \alpha n$. We define our "first extension of k-means" as follows:

$$\begin{aligned} \min_{U} \quad & \sum_{j=1}^{k} \sum_{i=1}^{n} u_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{i=1}^{n} u_{ij} \mathbf{x}_i}{\sum_{i=1}^{n} u_{ij}} \\ \text{s.t.} \quad & \text{trace}(U^T U) = (1 + \alpha) n. \end{aligned} \tag{3.2}$$

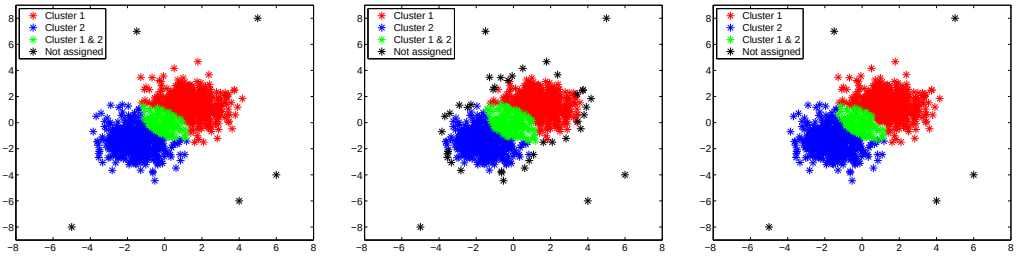
We require $\alpha \ll (k - 1)$ to avoid assigning each data point to every cluster. Similar to k-means, the above objective function is designed to minimize the sum of squared

distances between every data point to its cluster centroid, but now the assignment is not necessarily restricted to be disjoint and exhaustive.

However, the seemingly reasonable objective function (3.2) has a limitation. To illustrate this, we test this objective function on synthetic data. As shown in the leftmost plot in Figure 3.1, we generate two ground-truth clusters which contain both overlap and outliers (details about this dataset are described in Chapter 3.4). Red data points are only assigned to cluster 1, blue data points are only assigned to cluster 2, green data points are assigned to both of the clusters, and black data points are not assigned to any cluster. Using a k-means like algorithm, we can optimize (3.2), and Figure 3.1 (b) shows the clustering result. There are 1,000 data points, and we set α=0.1 (which is the ground-truth value of α). So, 1,100 assignments are made in U. We can see that (3.2) fails to correctly recover the ground-truth clusters. Some data points that are relatively far from the cluster centers are not assigned to any cluster even though they are not outliers. This result indicates that just controlling the total number of assignments in U is not enough, and we need another constraint to correctly control the non-exhaustiveness. Based on these investigations, we now propose our final objective function in the following chapter.

3.1.3 The NEO-K-Means Objective

Recall that in our first extension of k-means objective function (3.2), we just added a constraint on the total number of assignments in the assignment matrix U, and it resulted in more false positive outliers than expected. To fix this problem, we introduce another important constraint which controls the degree of non-exhaustiveness. To state our new optimization problem, let us define the indicator function $\mathbb{I}\{exp\}$ to be $\mathbb{I}\{exp\} = 1$ if exp is true; 0 otherwise, and we let $\mathbf{1}$ denote a $k \times 1$ column vector having all the elements equal to one. Then, the vector $U\mathbf{1}$ denotes the number of clusters to which each data point belongs. Thus, $(U\mathbf{1})_i = 0$



(a) Ground-truth clusters (b) First extension of k-means (c) NEO-K-Means

Figure 3.1: (a) Two ground-truth clusters are generated (n=1,000, α=0.1, β=0.005). Green points indicate overlap between the clusters, and black points indicate outliers. See Chapter 3.4 for details. (b) Our first extension of k-means objective function defined in (3.2) makes too many outlier assignments and fails to recover the ground-truth. (c) The NEO-K-Means objective defined in (3.3) adds an explicit term for non-exhaustiveness that enables it to correctly detect the outliers and find natural overlapping clustering structure which is very similar to the ground-truth clusters (α and β are automatically estimated by the heuristics discussed in Chapter 3.1.5).

means that $\mathbf{x}_i$ does not belong to any cluster. Now, by adding a non-exhaustiveness constraint to (3.2), we define our NEO-K-Means objective function as follows:

$$\begin{aligned} &\min_{U} \sum_{j=1}^{k} \sum_{i=1}^{n} u_{ij} \|\mathbf{x}_i - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{i=1}^{n} u_{ij}\mathbf{x}_i}{\sum_{i=1}^{n} u_{ij}} \\ &\text{s.t. } \operatorname{trace}(U^T U) = (1+\alpha)n, \ \textstyle\sum_{i=1}^{n} \mathbb{I}\{(U\mathbf{1})_i = 0\} \leq \beta n. \end{aligned} \tag{3.3}$$

We allow at most βn data points to be unassigned to any cluster, i.e., at most βn data points can be considered as outliers. We require $0 \leq \beta n$ and note that $\beta n \ll n$ to cause most data points to be assigned to clusters. Specifically, by the definition of "outliers", βn should be a very small number compared to n. The parameters α and β offer an intuitive way to capture the degree of overlap and non-exhaustiveness; by "turning the knob" on these parameters, the user can explore the landscape of overlapping, non-exhaustive clusterings. If α=0 and β=0, the NEO-K-Means objective function is equivalent to the standard k-means objective presented in (3.1). To see this, note that setting the parameter β=0 requires every data point

to belong to at least one cluster, while setting α=0 makes n assignments. Putting these together, the resulting clustering will be disjoint and exhaustive. Note that by setting α=0, objective (3.2) does not have this property.

To see if the objective function (3.3) yields a reasonable clustering, we test it on the same dataset we used in the previous chapter. Figure 3.1 (c) shows the result (α and β are automatically estimated by the heuristics discussed in Chapter 3.1.5). We see that NEO-K-Means correctly finds all the outliers, and produces very similar overlapping structure to the ground-truth clusters.

3.1.4 The NEO-K-Means Algorithm

We now propose a simple iterative algorithm which monotonically decreases the NEO-K-Means objective until it converges to a local minimum. Having the hard constraints in (3.3), we will make $n + \alpha n$ assignments such that at most βn data points can have no membership in any cluster. Note that the second constraint can be interpreted as follows: among n data points, at least $n - \beta n$ data points should have membership to some cluster. When our algorithm makes assignments of points to clusters, it uses two phases to satisfy these two constraints. Thus, each cluster $\mathcal{C}_j$ decomposes into two sets $\bar{\mathcal{C}}_j$ and $\hat{\mathcal{C}}_j$ that record the assignments made in each phase.

Algorithm 5 describes the NEO-K-Means algorithm. We first initialize cluster centroids. Any initialization strategies that are used in k-means may also be applied to our algorithm. Given cluster centroids, we compute all the distances $[d_{ij}]_{n\times k}$ between every data point and every cluster, and record the closest cluster and corresponding distance for every data point. Then, the data points are sorted in an ascending order by the distance to its closest cluster. To ensure at least $n-\beta n$ data points are assigned to some cluster (i.e., to satisfy the second constraint), we assign the first $n-\beta n$ data points to their closest clusters. Let $\bar{\mathcal{C}}_j$ denote the assign-

Algorithm 5 NEO-K-Means

Input: $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n\}$, the number of clusters k, the maximum number of iterations t_{max}, α, β

Output: $\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_k$

Initialize cluster means $\{\mathbf{m}_j\}_{j=1}^{k}$, $t = 0$.
while not converged and $t < t_{max}$ **do**
  Compute cluster means, and then compute distances between every data point and clusters $[d_{ij}]_{n \times k}$.
  Initialize $\mathcal{T} = \emptyset$, $\mathcal{S} = \emptyset$, $p = 0$, and $\bar{\mathcal{C}}_j = \emptyset$, $\hat{\mathcal{C}}_j = \emptyset \ \forall j$.
  while $p < (n + \alpha n)$ **do**
    if $p < (n - \beta n)$ **then**
      Assign $\mathbf{x}_{i^*}$ to $\bar{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}, i \notin \mathcal{S}$.
      $\mathcal{S} = \mathcal{S} \cup \{i^*\}$.
    else
      Assign $\mathbf{x}_{i^*}$ to $\hat{\mathcal{C}}_{j^*}$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}$.
    end if
    $\mathcal{T} = \mathcal{T} \cup \{(i^*, j^*)\}$.
    $p = p + 1$.
  end while
  $\forall j$, update clusters $\mathcal{C}_j = \bar{\mathcal{C}}_j \cup \hat{\mathcal{C}}_j$.
  $t = t + 1$.
end while

ments made by this step. Thus, $\sum_{j=1}^{k} |\bar{\mathcal{C}}_j| = n - \beta n$. Then, we make $\beta n + \alpha n$ more assignments by taking $\beta n + \alpha n$ minimum distances among $[d_{ij}]_{n \times k}$ such that $\mathbf{x}_i \notin \bar{\mathcal{C}}_j$. Let $\hat{\mathcal{C}}_j$ denote the assignments made by this step. Thus, $\sum_{j=1}^{k} |\hat{\mathcal{C}}_j| = \beta n + \alpha n$. Finally, $\sum_{j=1}^{k} (|\bar{\mathcal{C}}_j| + |\hat{\mathcal{C}}_j|) = n + \alpha n$. Once all the assignments are made, we update cluster centroids by recomputing the mean of each cluster. We repeat this procedure until the change in objective function is sufficiently small or the maximum number of iterations is reached. Note that the algorithm does not forcibly choose βn points as outliers; indeed, the number of outliers is *less* than βn and depends on the the distances between data points and their "secondary" clusters. We note that, if $\alpha = 0$ and $\beta = 0$, then the NEO-K-Means algorithm is identical to the standard k-means algorithm. Our algorithm guarantees monotonic decrease in the objective function as the following result shows.

Theorem 3. *Algorithm 5 monotonically reduces the NEO-K-Means objective in*

(3.3) while satisfying the constraints specified by α and β.

Proof. Let $J^{(t)}$ denote the objective at the t-th iteration. Then,

$$
\begin{aligned}
J^{(t)} &= \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_j^{(t)}} \|\mathbf{x}_i - \mathbf{m}_j^{(t)}\|^2 \\
&\geq \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \bar{\mathcal{C}}_j^{(t+1)}} \|\mathbf{x}_i - \mathbf{m}_j^{(t)}\|^2 + \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \hat{\mathcal{C}}_j^{(t+1)}} \|\mathbf{x}_i - \mathbf{m}_j^{(t)}\|^2 \\
&= \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_j^{(t+1)}} \|\mathbf{x}_i - \mathbf{m}_j^{(t)}\|^2 \text{ since } \mathcal{C}_j^{(t+1)} = \bar{\mathcal{C}}_j^{(t+1)} \cup \hat{\mathcal{C}}_j^{(t+1)} \\
&\geq \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_j^{(t+1)}} \|\mathbf{x}_i - \mathbf{m}_j^{(t+1)}\|^2 \text{ (property of centroids)} \\
&= J^{(t+1)}
\end{aligned}
$$

The first inequality follows from the update scheme used to form $\bar{\mathcal{C}}_j$ and $\hat{\mathcal{C}}_j$ by our algorithm (see steps 7 and 10 in Algorithm 5). Clearly the algorithm always maintains feasibility, i.e., the constraints specified by the parameters α and β are always satisfied. □

3.1.5 Parameter Selection

We now discuss how to choose the parameters α and β. Since parameter selection is usually considered as a challenging task, many existing clustering algorithms leave this as an open problem. For example, in k-means-- algorithm [25], the number of outliers is a required input. Many other clustering methods (e.g., [17], [14], [57]) also have their own model parameters that should be set by a user. While some model parameters of other clustering methods tend to be non-intuitive to set or it can be hard to predict the effect of a particular parameter setting, the NEO-K-Means parameters α and β are intuitive parameters that allow users to specify how much overlap/non-exhaustiveness they want. So, users might be able to estimate these parameters from domain knowledge. If any overlap and outlier statistics are

unknown, we can estimate α and β values by using the heuristics discussed in the following chapters. The high level idea of our parameter estimation is to first run a disjoint, exhaustive clustering and perform cheap distance-based computation.

Choosing β.

We first run a traditional k-means. Let d_i denote the distance between data point $\mathbf{x}_i$ and its closest cluster. We compute the mean (denoted by μ) and the standard deviation (denoted by σ) of d_i (i=1,$\cdots$,n). If a distance d_i is greater than $\mu+\delta\sigma$, then we consider the data point $\mathbf{x}_i$ as an outlier, where δ is a constant which controls how far from is it from the mean. We empirically observe that usually $\delta = 6$ leads to a reasonable estimate for β.

Choosing α.

We use two different strategies for choosing α. We empirically observe that the first strategy is better when the overlap is small and the second strategy is better when the overlap is large.

The first strategy considers the distribution of distances in each cluster. For each $\mathcal{C}_j$, we consider the distances between the center of $\mathcal{C}_j$ and the data points which are assigned to $\mathcal{C}_j$, and compute the mean (denoted by μ_j) and the standard deviation (denoted by σ_j) of these distances. Then, for a data point $\mathbf{x}_l \notin \mathcal{C}_j$, we compute the distance between $\mathbf{x}_l$ and $\mathcal{C}_j$, denoted by d_{lj}. If d_{lj} is less than $\mu_j + \delta\sigma_j$ (usually, $-1 \leq \delta \leq 3.5$ gives a good estimate), we consider the data point $\mathbf{x}_l$ should be in the overlapped region. In this way, we can count the number of points which should be considered in the overlapped region, and thus we can estimate α.

The second strategy considers normalized distances. Given a data point $\mathbf{x}_i$, let d_{ij} denote the distance between $\mathbf{x}_i$ and $\mathcal{C}_j$. We compute the normalized distance which is defined by $\bar{d}_{ij} = d_{ij} / \sum_{l=1}^{k} d_{il}$ (note that $\sum_{l=1}^{k} \bar{d}_{il} = 1$). Then, we count

the number of $\bar{d}_{ij}$ whose value is less than $1/(k+1)$. Notice that if a data point is equidistant from every cluster, then the normalized distance is equal to $1/k$. To get a stronger bound, we set the threshold to be $1/(k+1)$. If the normalized distance is less than this threshold, we consider that the data point should be in the overlapped region. In this way, we can estimate the amount of overlap.

3.1.6 Weighted Kernel NEO-K-Means

Now, let us discuss weighted kernel k-means. In kernel k-means, each data point is first mapped into a higher dimensional feature space, and then clustered using k-means in the feature space. A weighted version of kernel k-means [32] also has been introduced to differentiate each data point's contribution to the objective function by assigning a weight to each data point. Let ϕ denote a nonlinear mapping, and w_i denote a nonnegative weight for data point $\mathbf{x}_i$. Then, the weighted kernel k-means objective [32] is defined as follows:

$$\min_{\{\mathcal{C}_j\}_{j=1}^{k}} \sum_{j=1}^{k} \sum_{\mathbf{x}_i \in \mathcal{C}_j} w_i \|\phi(\mathbf{x}_i) - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{\mathbf{x}_i \in \mathcal{C}_j} w_i \phi(\mathbf{x}_i)}{\sum_{\mathbf{x}_i \in \mathcal{C}_j} w_i}. \tag{3.4}$$

Most algorithms to optimize this objective exploit the well-known kernel trick to avoid forming the feature space explicitly and use the kernel matrix of inner products instead. Let us consider the weighted kernel NEO-K-Means objective function. Just like in (3.4), we introduce a nonlinear mapping ϕ and a weight w_i for each data point $\mathbf{x}_i$. Then the weighted kernel NEO-K-Means objective is defined as follows:

$$\begin{aligned} \min_{U} \quad & \sum_{c=1}^{k} \sum_{i=1}^{n} u_{ic} w_i \|\phi(\mathbf{x}_i) - \mathbf{m}_c\|^2, \text{ where } \mathbf{m}_c = \frac{\sum_{i=1}^{n} u_{ic} w_i \phi(\mathbf{x}_i)}{\sum_{i=1}^{n} u_{ic} w_i} \\ \text{s.t.} \quad & \text{trace}(U^T U) = (1+\alpha)n, \ \sum_{i=1}^{n} \mathbb{I}\{(U\mathbf{1})_i = 0\} \leq \beta n. \end{aligned} \tag{3.5}$$

The extension of NEO-K-Means to the weighted kernel case enables us to tackle the problem of non-exhaustive, overlapping graph clustering (overlapping community detection), which we describe in the next chapter.

3.2 Graph Clustering using NEO-K-Means

We first review some of the traditional graph clustering objectives, and then present an extension of the traditional graph cut objectives to non-exhaustive, overlapping clustering setting. We show that this extended graph clustering objective is equivalent to the weighted kernel NEO-K-Means objective. Thus, we present a principled method to compute non-exhaustive, overlapping graph clustering by applying the NEO-K-Means algorithm.

3.2.1 Graph Clustering via Normalized Cut

Given a graph $G = (\mathcal{V}, \mathcal{E})$, the corresponding adjacency matrix is defined as $A = [a_{ij}]$ such that a_{ij} is equal to the edge weight between vertex i and j if there is an edge, and zero otherwise. We assume we are working with undirected graphs, where the matrix A is symmetric. We also assume that there is no self-loop in the graph, i.e., the diagonal elements of A are all zeros. The traditional graph partitioning problem seeks k pairwise disjoint clusters such that $\mathcal{C}_1 \cup \mathcal{C}_2 \cup \cdots \cup \mathcal{C}_k = \mathcal{V}$.

Normalized cut [73] is a popular measure to evaluate the quality of a graph partitioning or graph clustering. Let $\texttt{links}(\mathcal{C}_p, \mathcal{C}_q)$ denote the sum of edge weights between two sets $\mathcal{C}_p, \mathcal{C}_q$. Then, the normalized cut of a graph is defined as follows:

$$\texttt{NCut}(G) = \min_{\mathcal{C}_1, \mathcal{C}_2, \cdots \mathcal{C}_k} \sum_{j=1}^{k} \frac{\texttt{links}(\mathcal{C}_j, \mathcal{V} \backslash \mathcal{C}_j)}{\texttt{links}(\mathcal{C}_j, \mathcal{V})}. \tag{3.6}$$

Using a linear algebraic formulation, the normalized cut objective may be

expressed as follows:

$$\begin{aligned}\texttt{NCut}(G) &= \min_{\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_k} \sum_{j=1}^{k} \frac{\mathbf{y}_j^T(D-A)\mathbf{y}_j}{\mathbf{y}_j^T D\mathbf{y}_j} \\ &= \max_{\mathbf{y}_1,\mathbf{y}_2,\cdots\mathbf{y}_k} \sum_{j=1}^{k} \frac{\mathbf{y}_j^T A\mathbf{y}_j}{\mathbf{y}_j^T D\mathbf{y}_j},\end{aligned} \tag{3.7}$$

where D is the diagonal matrix of vertex degrees, and $\mathbf{y}_j$ denotes an indicator vector for cluster $\mathcal{C}_j$, i.e., $\mathbf{y}_j(i) = 1$ if a vertex v_i belongs to cluster $\mathcal{C}_j$, zero otherwise.

3.2.2 Extending Graph Cut Objectives to Non-exhaustive, Overlapping Clustering

Note that the traditional normalized cut objective (3.7) is for disjoint, exhaustive graph clustering. To consider non-exhaustive, overlapping graph clustering, we first introduce an assignment matrix $Y = [y_{ij}]_{n\times k}$ such that y_{ij}=1 if a vertex v_i belongs to cluster $\mathcal{C}_j$; y_{ij}=0 otherwise. Let $\mathbf{y}_j$ denote jth column of Y. Then, we can extend (3.7) to non-exhaustive, overlapping graph clustering by introducing the same constraints as in (3.3):

$$\begin{aligned}&\max_{Y} \sum_{j=1}^{k} \frac{\mathbf{y}_j^T A\mathbf{y}_j}{\mathbf{y}_j^T D\mathbf{y}_j} \\ &\text{s.t. trace}(Y^TY) = (1+\alpha)n,\ \ \textstyle\sum_{i=1}^{n} \mathbb{I}\{(Y\mathbf{1})_i = 0\} \le \beta n.\end{aligned} \tag{3.8}$$

By adjusting α and β, we can control the degree of overlap and non-exhaustiveness. If α=0, and β=0, the above objective enforces disjoint and exhaustive clustering, thus is equivalent to the traditional normalized cut objective. We have focused on the normalized cut objective, but other graph clustering objectives (e.g., ratio association [73]) also can be extended to non-exhaustive, overlapping clustering using the same approach.

3.2.3 Equivalence of the Objectives

We now show that (3.5) is equivalent to (3.8) by defining an appropriate kernel and weights. Let $W = [w_{ii}]_{n \times n}$ denote a diagonal weight matrix whose diagonal entries are equal to vertex weights, let K denote a kernel matrix such that $K_{ij} = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$, and let $\mathbf{u}_c$ denote the cth column of U. Then, (3.5) can be rewritten as follows:

$$\begin{aligned} &\min_U \sum_{c=1}^{k} \sum_{i=1}^{n} u_{ic} w_i \|\phi(\mathbf{x}_i) - \mathbf{m}_c\|^2 \\ &= \min_U \sum_{c=1}^{k} \left(\sum_{i=1}^{n} u_{ic} w_i K_{ii} - \frac{{\mathbf{u}_c}^T W K W \mathbf{u}_c}{{\mathbf{u}_c}^T W \mathbf{u}_c} \right) \end{aligned} \tag{3.9}$$

Let us define the kernel as $K \equiv \gamma W^{-1} + W^{-1} A W^{-1}$ where γ is a positive constant typically chosen to make K positive definite. Then (3.9) can be expressed as follows:

$$\begin{aligned} &= \min_U \sum_{c=1}^{k} \left(\sum_{i=1}^{n} u_{ic} w_i \frac{\gamma}{w_i} - \frac{{\mathbf{u}_c}^T A \mathbf{u}_c}{{\mathbf{u}_c}^T W \mathbf{u}_c} \right) \\ &= \min_U \left(\gamma(1+\alpha)n - \sum_{c=1}^{k} \frac{{\mathbf{u}_c}^T A \mathbf{u}_c}{{\mathbf{u}_c}^T W \mathbf{u}_c} \right) \\ &= \max_U \sum_{c=1}^{k} \frac{{\mathbf{u}_c}^T A \mathbf{u}_c}{{\mathbf{u}_c}^T W \mathbf{u}_c} \end{aligned} \tag{3.10}$$

Now, in (3.10), let us define the weight matrix as $W \equiv D$. Notice that $U = Y$ in (3.8). Putting these together, we can see that the weighted kernel NEO-K-Means objective (3.5) is equivalent to the extended normalized cut objective (3.8).

3.2.4 Algorithm

The equivalence between (3.8) and (3.5) implies that we can optimize the non-exhaustive, overlapping graph clustering objectives using the weighted kernel

NEO-K-Means algorithm. The difference between standard k-means and weighted kernel k-means is how to compute the distance between a data point and clusters. Using our definitions of kernel and weights, the distance between a vertex v_i and cluster $\mathcal{C}_j$ can be quantified as follows:

$$\texttt{dist}(v_i, \mathcal{C}_j) = -\frac{2\,\texttt{links}(v_i, \mathcal{C}_j)}{\texttt{deg}(v_i)\,\texttt{deg}(\mathcal{C}_j)} + \frac{\texttt{links}(\mathcal{C}_j, \mathcal{C}_j)}{\texttt{deg}(\mathcal{C}_j)^2} + \frac{\gamma}{\texttt{deg}(v_i)} - \frac{\gamma}{\texttt{deg}(\mathcal{C}_j)}, \qquad (3.11)$$

where $\texttt{deg}(v_i)$ denotes the degree of vertex v_i, and $\texttt{deg}(\mathcal{C}_j)$ denotes the sum of edge weights of vertices in $\mathcal{C}_j$. Then, Algorithm 5 can be applied to graph data by computing the distances using (3.11).

Popular software for graph partitioning, e.g., Graclus [32] and Metis [46], employs a multilevel approach. In this framework, an input graph is coarsened by merging nodes level by level. As a result, a series of smaller graphs are created. Once the input graph is coarsened into a small enough graph, an initial partitioning is performed. The clustering result of the coarsest level graph is first projected onto the graph at the level above it. Many different heuristics can be used at any of these coarsening or projection stages in order to improve the overall performance. The clustering is then refined through a refinement algorithm which plays the most important role in optimizing an objective function.

We also exploit the multilevel framework for non-exhaustive, overlapping graph clustering. While we use similar heuristics for coarsening and initial partitioning phases as in [32], we implement the weighted kernel NEO-K-Means algorithm for the refinement phase. Besides the multilevel refinement, any reasonable initialization can be directly given, and we can apply the weighted kernel NEO-K-Means algorithm to optimize the non-exhaustive, overlapping graph clustering objective.

3.3 Related Work

Both the aspects of overlap and non-exhaustiveness in clustering have been studied before, albeit rarely considered together in a unified manner as we do. We recognize that [26] also considers both overlap and non-exhaustiveness by modifying the traditional k-medoid algorithm, but their methodologies include complicated heuristics. A few recent papers study the clustering problem with outlier detection. In particular, [25] have proposed the k-means-- algorithm, which discovers clusters and outliers in a unified fashion; however it does not find overlapping clusters.

We focus our discussion on overlapping clustering as that literature is the most closely related to our contribution. Soft clustering methods, such as fuzzy k-means [17], relax the binary assignment constraint and replace it by a probabilistic assignment. Thresholding these probabilities may result in both overlapping assignments to clusters and non-exhaustive partitions, although it is difficult to control these effects. There have been many attempts to extend k-means to overlapping clustering. For example, [29] defines OKM. However, it has been recognized that OKM tends to yield large overlap among the clusters, which the restricted OKM method [14] should address. Separately, [57] also has reformulated the OKM objective function by adding a sparsity constraint. On the other hand, from the Bayesian perspective, [11] proposed a generative model, called MOC, where each data point is assumed to be generated from an exponential family.

In the context of graph clustering, many different types of overlapping graph clustering, or overlapping community detection methods, have been presented. Compared to our seed-and-grow algorithm which we present in Chapter 2, the NEO-K-Means algorithm adopts a more principled approach. Among the existing methods, the scalable alternatives include *demon* [30] and *bigclam* methods [85]—which we compare against. We also compare with *oslom* [50] which detects outliers and produces statistically significant overlapping communities. Many other methods are

Table 3.1: Vector datasets.

	n	dim.	$\lvert\bar{\mathcal{C}}\rvert$	outliers	k
synth1	5,000	2	2,750	0	2
synth2	1,000	2	550	5	2
synth3	6,000	2	3,600	6	2
yeast	2,417	103	731.5	0	14
music	593	72	184.7	0	6
scene	2,407	294	430.8	0	6

discussed in a recent survey [83], although the majority of successful approaches tend to suffer scalability issues on large networks like those we consider. Our derivation of the relationship between NEO-K-Means and overlapping community detection is inspired by [32] which showed the connection between k-means and graph partitioning.

3.4 Experimental Results

We show the experimental results of NEO-K-means on both vector data and graph data.

3.4.1 Vector Data

We compare NEO-K-Means with fuzzy k-means [17] (denoted by *fuzzy*), MOC [11] (denoted by *moc*), OKM [29] (denoted by *okm*), restricted OKM [14] (denoted by *rokm*), and explicit/implicit sparsity constrained clustering [57] (denoted by *esp* and *isp*, respectively). For NEO-K-Means, we use our methodologies for estimating α and β (discussed in Chapter 3.1.5). We initialize all the methods using k-means with exactly the same centroids, run each of the algorithms 5 times, and pick the assignment matrix which leads to the best objective function value of each method. If an algorithm happens to return empty clusters or clusters that contain all the data points, we exclude these clusters when we compute F_1 score. Table 3.1

Table 3.2: F_1 scores on vector datasets. NEO-K-Means (the last column) achieves the highest F_1 score across all the datasets while the performance of other existing algorithms is not consistent across all the datasets.

	moc	*fuzzy*	*esp*	*isp*	*okm*	*rokm*	NEO
synth1	0.833	0.959	0.977	0.985	0.989	0.969	**0.996**
synth2	0.836	0.957	0.952	0.973	0.967	0.975	**0.996**
synth3	0.547	0.919	0.968	0.952	0.970	0.928	**0.996**
yeast	N/A	0.308	0.289	0.203	0.311	0.203	**0.366**
music	0.534	0.533	0.527	0.508	0.527	0.454	**0.550**
scene	0.467	0.431	0.572	0.586	0.571	0.593	**0.626**

shows a summary of our vector datasets. 'dim.' denotes the dimension of the data points, and $|\bar{\mathcal{C}}|$ denotes the average cluster size.

Synthetic Data.

Three synthetic datasets are generated from the Gaussian distribution. We first fix the cluster centroid of each cluster, and then draw $n - \beta n$ data points from the Gaussian distribution whose mean is the cluster centroid and covariance matrix is the identity. To create the ground-truth clusters, we first assign all the data points to its closest cluster, and then make additional assignments by taking minimum distances such that the total number of non-zeros of the ground-truth cluster matrix is equal to $n + \alpha n$. Finally, βn outliers are added to the data matrix, and these data points are not assigned to any cluster in the ground-truth cluster matrix.

The first three rows of Table 3.2 shows the F_1 scores on the synthetic datasets. Because the datasets are simple, almost all the methods achieve high F_1 scores (above 0.9) except *moc*. On synth3, one of the clusters produced by *moc* contains all the data points in the cluster. As a result, *moc* gets a particularly low F_1 score on this dataset. We can see that NEO-K-Means achieves the highest F_1 score on all of the datasets. While synth1 does not contain outliers, synth2 and synth3 contain

five and six outliers, respectively. We observe that NEO-K-Means finds the correct number of outliers, and perfectly finds all the outliers. On the other hand, all the other baseline methods do not have the functionality of non-exhaustive clustering, so they assign the outliers to some clusters.

Real-world Data.

We use three real-world multi-label datasets from [1], which are presented in Table 3.1: 'yeast', 'music', and 'scene'. The 'music' dataset [76] consists of a set of feature vectors extracted from 593 different music songs. In this dataset, each song is labelled by emotions presented in the song, e.g., happy, surprised, relaxing, etc. Since several different emotions can be expressed in a song, a song can have more than one label. The 'scene' dataset [19] is a set of scene image feature vectors. Each image can be labelled by their scenes, e.g., beach, sunset, mountain, and an image can contain more than one scene. The 'yeast' dataset [35] is from a biology domain. This dataset is a set of feature vectors constructed based on micro-array expression data and phylogenetic profiles of genes. Each gene belongs to multiple functional classes, so each gene can have multiple labels. On these datasets, we treat each label as a ground-truth cluster.

The F_1 scores are presented in the last three rows of Table 3.2. On the 'yeast' dataset, among 14 clusters, *moc* returns 13 empty clusters and one cluster that contains all the data points. Thus, we cannot report the F_1 score of *moc*. We can see that NEO-K-Means always shows the best F_1 score while the algorithmic performance of the other methods varies. For instance, *rokm* is the worst for 'music', but is the second best for the 'scene' dataset.

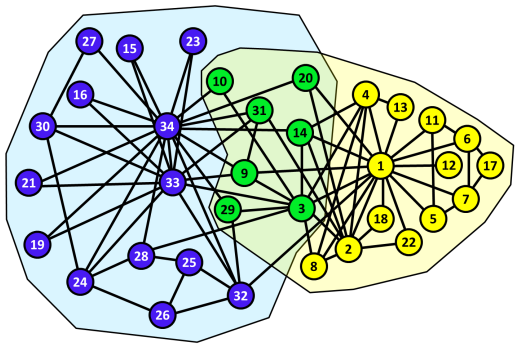


Figure 3.2: Clustering result of NEO-K-Means on Karate Club network. NEO-K-Means is able to reveal the natural underlying overlapping structure of the network.

3.4.2 Community Detection in Graph Data

Karate Club Network.

As an illustration of the method, we first apply our NEO-K-Means algorithm on Zachary's Karate Club network, which is a classical example for testing clustering algorithms. This network represents friendship relationships between 34 members in a karate club at a US university in 1970. In this network, node 1 and node 34 are known to be the instructor and the student founder of the club, respectively. These two nodes are central in the network forming two natural clusters around them. We run the NEO-K-Means with α=0.2, and β=0, so in this setting, the algorithm will make 41 assignments in total, i.e., 7 nodes can belong to both clusters (note that the number of common friends of node 1 and node 34 is four, so we are looking for something a little less than twice the obvious overlap). Figure 3.2 shows the clustering result of NEO-K-Means. We can see that the nodes that are assigned to both clusters have strong interactions with both of the underlying clusters.

Large Real-world Networks.

For comparisons on large real-world networks, we use four real-world networks from [4], which are presented in Table 3.3. We compare the weighted ker-

Table 3.3: Graph datasets

	No. of vertices	No. of edges
Amazon	334,863	925,872
DBLP	317,080	1,049,866
Flickr	1,994,422	21,445,057
LiveJournal	1,757,326	42,183,338

Table 3.4: Average normalized cut of each algorithm on large real-world networks. Lower normalized cut indicates better clustering. NEO-K-Means achieves the lowest normalized cut on all the datasets.

	demon	*oslom*	*bigclam*	NISE	NEO
Amazon	0.555	0.221	0.392	0.116	**0.105**
DBLP	0.606	0.355	0.617	0.204	**0.188**
Flickr	N/A	N/A	0.596	0.515	**0.331**
LiveJournal	N/A	N/A	0.912	0.643	**0.373**

nel NEO-K-Means algorithm with state-of-the-art overlapping community detection methods: NISE which is our personalized PageRank-based method presented in Chapter 2, *bigclam* [85], *demon* [30], and *oslom* [50]. For the comparison with NISE, we use 'graclus centers' seeding method because it produces better average normalized cut values than 'spread hubs' seeding method. As we did in Chapter 2, we set k to 15,000 for Flickr and LiveJournal, and 25,000 for DBLP and Amazon. For NEO-K-Means, we set α=10, β=0 for Flickr and LiveJournal, and α=40, β=0.0001 for DBLP and Amazon. We choose small values of β and large values of α because (i) we expect that there are graph-based pre-processing techniques that remove obvious outliers (for example, connected components analysis) and (ii) real-world networks have vertices in many clusters.

We first compare the methods in terms of the average normalized cut. Recall that the normalized cut of a graph clustering is computed by (3.6). A lower normalized cut should indicate a better clustering. We compute the average normalized cut by dividing the total normalized cut by the returned number of clusters. Table 3.4

Table 3.5: F_1 score of each algorithm on Amazon and DBLP. NEO-K-Means shows the highest F_1 score on Amazon, and comparable F_1 score with NISE on DBLP.

	demon	*oslom*	*bigclam*	NISE	NEO
Amazon	0.165	0.318	0.269	0.467	**0.490**
DBLP	0.137	0.132	0.151	**0.176**	0.174

Table 3.6: Average normalized cut and F_1 score of NEO-K-Means with different α and β on Amazon dataset.

	α=30, β=0	α=35, β=0	α=45, β=0	α=30, β=0.0001	α=35, β=0.0001	α=45, β=0.0001
ncut	0.107	0.104	0.104	0.106	0.104	0.104
F_1	0.488	0.490	0.490	0.488	0.490	0.490

shows the average normalized cut of each algorithm. The *demon* and *oslom* programs fail on the Flickr and LiveJournal networks. We can see that NEO-K-Means achieves the lowest normalized cut value across all the networks. This indicates that the weighted kernel NEO-K-Means is effective in optimizing the normalized cut objective. In Amazon and DBLP, the ground-truth communities are known. Table 3.5 shows F_1 scores on these networks. The NEO-K-Means method shows the best F_1 score on Amazon, but is slightly outperformed by NISE on DBLP. However, we note that NISE is a highly tuned, complicated heuristic, whereas the NEO-K-means algorithm is a more principled method. Table 3.6 shows the results of NEO-K-Means with different α and β on the Amazon dataset. We can see that the results are not too sensitive to the particular α and β picked.

Chapter 4

Low-Rank Semidefinite Programming for Non-exhaustive, Overlapping Clustering

The iterative NEO-K-Means algorithm (presented in Chapter 3) is fast but suffers from the classic problem that iterative algorithms for k-means fall into local minimizers given poor initialization. For a more accurate solution, we continue our study of the NEO-K-Means objective function by proposing a convex relaxation (Chapter 4.2). This convex problem can be globally optimized in time and memory that is polynomial in the input size. The relaxed solution can then be rounded to a discrete assignment solution. Our experimental results with this algorithm

The materials presented in this chapter have been published in [43]. Joyce and Yangyang developed and implemented the algorithms, and also conducted experiments. Professor Gleich and Professor Dhillon supervised the work.

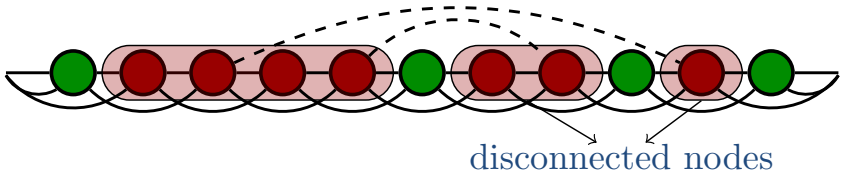
show that it results in better objective function values than the previous iterative algorithm, albeit at a substantial computational cost.

The convex formulation is not without problems. When the NEO-K-Means problem is relaxed to a convex semi-definite program (SDP), the number of variables is *quadratic* in the number of data points. Off-the-shelf SDP solvers such as CVX [40, 39] can then only solve problems with fewer than 100 data points (this is due to a variety of complexities that arise when our SDP is converted into a standard form for existing convex solvers). Even small modern datasets have a few thousand points, and thus a different approach is required.

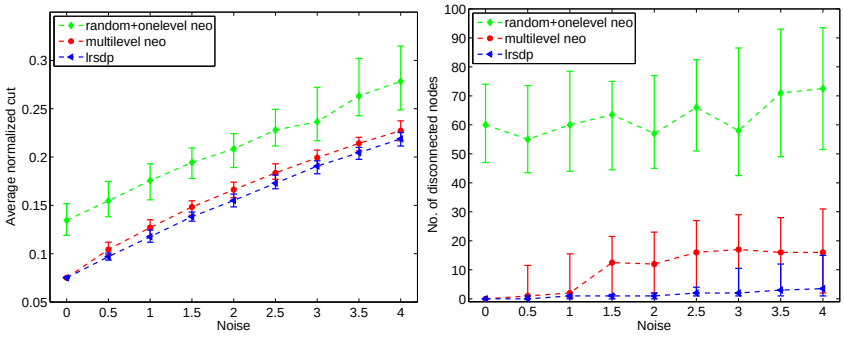
Consequently, we propose optimizing a low-rank factorization of the SDP solution matrix (Chapter 4.3). This is a standard technique to tackle large-scale SDPs [21]. The resulting optimization problem is a quadratically constrained problem with a quadratic objective that can no longer be globally optimized. An augmented Lagrangian procedure, for instance, will only converge to a local minimizer. Nevertheless, when this approach has been used in the past with high-quality optimization methods, it frequently generates solutions that are *as good as the global optimal* from convex solvers [21], a fact which has some theoretical justification [22]. Furthermore, similar ideas yielded stability improvements to convex relaxations of the classic k-means objective [49].

Our new LRSDP algorithm to optimize this non-linear problem can handle problems with tens of thousands of data points, providing an order of magnitude increase in scalability over the convex solver. On the problems where we can compare with the convex formulations, we achieve globally optimal objective values. It also consistently outperforms the iterative algorithm for NEO-K-Means in terms of objective function value.

Our goal with the new procedure is to produce more accurate and reliable clusterings than the previous iterative algorithm in the regime of medium-scale



(a) The disconnected nodes error measure counts the number of nodes that are disconnected from the largest connected component. These nodes are illustrated for the cluster in red. (Green nodes are not in that cluster.)



(b) Avg. normalized cut (c) No. of disconnected points

Figure 4.1: A synthetic study of overlapping community detection on a Watts-Strogatz cycle graph where each point should be assigned to two clusters: (a) an illustration of a portion of the cycle with dashed 'noise' edges showing the disconnected points measure (which is 3); (b) & (c) the results of normalized cut and the number of disconnected points on graphs with 100 nodes returned by our new LRSDP procedure compared with two variations of our previous "neo" iterative algorithms.

problems. This regime is ideal because the new method is more computationally expensive than the iterative algorithm, which was an efficient procedure designed for problems with millions of data points. To see the difference between these methods, we study the behavior on a synthetic problem with community detection on a cycle graph. The graph is a Watts-Strogatz random graph where each node has edges to five neighbors on each side of the cycle. We also add random edges based on an Erdös-Rényi graph with expected degree d, which we consider as noise edges. When the noise is low, clusterings should respect the cycle structure and be continuous, connected regions. Hence, we compute an error measure for each cluster based on the

number of points disconnected from the largest connected component in the cycle; this measure is illustrated in Figure 4.1(a). We compare three methods: the straightforward iterative NEO-K-Means method with random initialization, a multilevel variation on that method, and our LRSDP with random initialization. We run 100 trials and plot the the median, 25th and 75th percentiles of the normalized cut scores and the number of disconnected nodes by varying the noise level. Figure 4.1(b) & Figure 4.1(c) show the results. Our LRSDP method achieves the best performance in terms of both the normalized cut and the number of disconnected nodes. We observe that our LRSDP method often produces 0 disconnected points even as the noise increases whereas the faster iterative method starts to introduce many disconnected points with only a modest amount of error.

We now summarize the contributions of our work:

- We propose NEO-SDP: a convex relaxation of a k-means-like objective that handles non-exhaustive, overlapping clustering problems (Chapter 4.2).
- We formulate the scalable NEO-LR objective and an LRSDP algorithm to optimize a low-rank factorization of the NEO-SDP solution (Chapter 4.3).
- We also propose a series of initialization and rounding strategies that accelerate the convergence of our optimization procedures (Chapter 4.3.3).
- We evaluate LRSDP on real-world data clustering problems and find it achieves the best F_1 performance with respect to ground-truth clusters (Chapter 4.5.3).
- For graph clustering problems, LRSDP produces the best quality communities among all clustering algorithms on real-world networks (Chapter 4.5.4).

4.1 Low-Rank Factorizations of SDPs

Semidefinite programs (SDPs) are one of the most general classes of tractable convex optimization problems. The canonical form and low-rank variation are:

$\mathbf{x}_i$	the data points for k-means	§3.1
k	the number of clusters	
α	the overlap parameter (0 means no overlap)	
β	the outlier parameter (0 means no outliers)	
$\boldsymbol{U}$	the assignment matrix for a solution	§3.1
$\boldsymbol{Z}$	the co-occurence matrix for the SDP relaxation	§4.2
$\boldsymbol{K}$	the kernel matrix for NEO-K-Means	
$\boldsymbol{W}$	a diagonal weight matrix for weighted problems	
$\mathbf{d}$	a specialized weight vector for the SDP relaxation	
$\mathbf{f}$	the cluster count variable for the SDP relaxation	
$\mathbf{g}$	the outlier indicator for the SDP relaxation	
$\boldsymbol{Y}$	the low-rank approximation of $\boldsymbol{Z}$ in NEO-LR	§4.3

Table 4.1: A summary of the notation used in the NEO-K-Means problem, the final assignment, and the SDP and low-rank approximations.

Canonical SDP	**Low-rank SDP**
maximize $\operatorname{trace}(\boldsymbol{CX})$	maximize $\operatorname{trace}(\boldsymbol{CYY}^T)$
subject to $\boldsymbol{X} \succeq 0, \boldsymbol{X} = \boldsymbol{X}^T$,	subject to $\boldsymbol{Y} : n \times k$
$\operatorname{trace}(\boldsymbol{A}_i\boldsymbol{X}) = b_i$	$\operatorname{trace}(\boldsymbol{A}_i\boldsymbol{YY}^T) = b_i$
$i = 1, \ldots, m$	$i = 1, \ldots, m$

Notice the low-rank form drops the positive semidefinite ($\boldsymbol{X} \succeq 0$) and symmetry constraints ($\boldsymbol{X} = \boldsymbol{X}^T$) but replaces $\boldsymbol{X} = \boldsymbol{YY}^T$, which automatically satisfies these constraints. Canonical SDPs can be optimized by a variety of solvers such as CVX [40], [39]. Low-rank SDP factorizations are non-convex and can be locally optimized via an augmented Lagrangian method [21].

4.2 An SDP for NEO-K-Means

We begin by stating an exact SDP-like program for the weighted kernel NEO-K-Means objective and then describe how to relax it to an SDP. We use the same notation as the previous chapter and summarize our common notation in Table 4.1. The essential idea with the SDP-like version is that we replace the assignment matrix

$\boldsymbol{U}$ with a normalized cluster co-occurrence matrix $\boldsymbol{Z}$:

$$\boldsymbol{Z} = \sum_{c=1}^{k} \frac{\boldsymbol{W}\mathbf{u}_c(\boldsymbol{W}\mathbf{u}_c)^T}{s_c}$$

where $\boldsymbol{W}$ is a diagonal matrix with the data point weights w_i on the diagonal, $\mathbf{u}_c$ is the c-th column of matrix $\boldsymbol{U}$ and $s_c = \mathbf{u}_c^T \boldsymbol{W}\mathbf{u}_c$. When $\boldsymbol{Z}$ is defined from an assignment matrix $\boldsymbol{U}$, then values of Z_{ij} are non-zero when items co-occur in a cluster. With appropriate constraints on the matrix $\boldsymbol{Z}$, it serves as a direct replacement for the assignment matrix $\boldsymbol{U}$.

To state the problem, let $\boldsymbol{K}$ denote the kernel matrix of the data points, e.g., if $\boldsymbol{X}$ is the data matrix whose rows correspond to data vectors, then $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{X}^T$ is just the simple linear kernel matrix. Let $\mathbf{d}$ be a vector where $d_i = w_i K_{ii}$, i.e., a weighted diagonal from $\boldsymbol{K}$. We need two new types of variables as well:

- Let $\mathbf{f}$ denote a vector of length n such that the i-th entry indicates the number of clusters that data point i belongs to.
- Similarly, let $\mathbf{g}$ denote a vector of length n such that the i-th entry is one if that data point i belongs to any clusters, and zero if the data point does not belong to any cluster.

Finally, we denote by $\mathbf{e}$ the vector of all 1s.

The following program is equivalent to the NEO-K-Means objective with a discrete assignment matrix:

$$
\begin{aligned}
\underset{\boldsymbol{Z},\mathbf{f},\mathbf{g}}{\text{maximize}} \quad & \operatorname{trace}(\boldsymbol{K}\boldsymbol{Z}) - \mathbf{f}^T\mathbf{d} \\
\text{subject to} \quad & \operatorname{trace}(\boldsymbol{W}^{-1}\boldsymbol{Z}) = k, && (a) \\
& Z_{ij} \geq 0, && (b) \\
& \boldsymbol{Z} \succeq 0, \boldsymbol{Z} = \boldsymbol{Z}^T && (c) \\
& \boldsymbol{Z}\mathbf{e} = \boldsymbol{W}\mathbf{f}, && (d) \\
& \mathbf{e}^T\mathbf{f} = (1+\alpha)n, && (e) \\
& \mathbf{e}^T\mathbf{g} \geq (1-\beta)n, && (f) \\
& \mathbf{f} \geq \mathbf{g}, && (g) \\
& \operatorname{rank}(\boldsymbol{Z}) = k, && (h) \\
& \mathbf{f} \in \mathcal{Z}^n_{\geq 0}, \mathbf{g} \in \{0,1\}^n. && (i)
\end{aligned}
\tag{4.1}
$$

We omit the verification that this is actually equivalent to the NEO-K-Means objective as it is not informative for our discussion. Constraints $(a), (b), (c)$, and (h) encode the fact that $\boldsymbol{Z}$ must arise from an assignment matrix. Constraints $(d), (e), (f), (g)$, and (i) are new to our NEO-K-Means formulation that express the amount of overlap and non-exhaustiveness in the solution. This is a mixed-integer, rank constrained SDP. As such, it is combinatorially hard to optimize just like the original NEO-K-Means objective.

The constraints that make this a combinatorial problem are (h) and (i). If we relax these constraints:

$$
\begin{aligned}
\underset{\boldsymbol{Z},\mathbf{f},\mathbf{g}}{\text{maximize}} \quad & \operatorname{trace}(\boldsymbol{K}\boldsymbol{Z}) - \mathbf{f}^T\mathbf{d} \\
\text{subject to} \quad & (a), (b), (c), (d), (e), (f), (g) \\
& 0 \leq \mathbf{g} \leq 1
\end{aligned}
\tag{4.2}
$$

then we arrive at a convex problem. Thus, any local optimal solution of (4.2) must be a global solution.

Solving (4.2) requires a black-box sdp solver such as CVX. As it converts this

problem into a standard form for such problems, the number of variables becomes $\mathcal{O}(n^2)$ and the resulting complexity is worse than $\mathcal{O}(n^3)$ in most cases, and can be as bad as $\mathcal{O}(n^6)$. These solvers are further limited by the delicate numerical precision issues that arise as they approach a solution. The combination of these features means that off-the-shelf procedures struggle to solve problems with more than 100 data points. We now describe a means to enable us to solve larger problems.

4.3 A Low-Rank SDP for NEO-K-Means

In the SDP formulation of the NEO-K-Means objective (4.2), the matrix $\boldsymbol{Z}$ should only be rank k. By applying the low-rank factorization idea, $\boldsymbol{Z}$ becomes $\boldsymbol{Y}\boldsymbol{Y}^T$ where $\boldsymbol{Y}$ is $n \times k$ and non-negative. Thus, the following optimization program is a low-rank SDP for (4.2) (we have chosen to write it in the standard form of a minimization problem with explicit slack variables $\mathbf{s}, r$ to convert the inequality constraints into equality and bound constraints).

$$
\begin{aligned}
\underset{\boldsymbol{Y},\mathbf{f},\mathbf{g},\mathbf{s},r}{\text{minimize}} \quad & \mathbf{f}^T\mathbf{d} - \text{trace}(\boldsymbol{Y}^T\boldsymbol{K}\boldsymbol{Y}) & \\
\text{subject to} \quad & k = \text{trace}(\boldsymbol{Y}^T\boldsymbol{W}^{-1}\boldsymbol{Y}) & (s) \\
& 0 = \boldsymbol{Y}\boldsymbol{Y}^T\mathbf{e} - \boldsymbol{W}\mathbf{f} & (t) \\
& 0 = \mathbf{e}^T\mathbf{f} - (1+\alpha)n & (u) \\
& 0 = \mathbf{f} - \mathbf{g} - \mathbf{s} & (v) \\
& 0 = \mathbf{e}^T\mathbf{g} - (1-\beta)n - r & (w) \\
& Y_{ij} \geq 0, \mathbf{s} \geq 0, r \geq 0 & \\
& 0 \leq \mathbf{f} \leq k\mathbf{e}, 0 \leq \mathbf{g} \leq 1 &
\end{aligned}
\tag{4.3}
$$

Here we also replaced the constraint $\boldsymbol{Y}\boldsymbol{Y}^T \geq 0$ with the stronger constraint $\boldsymbol{Y} \geq 0$. This problem is a quadratic programming problem with quadratic constraints, and we will discuss how to solve it in the next chapter. We call the problem NEO-LR and

the solution procedure LRSDP. Even though now we lose convexity by formulating the low rank SDP, this nonlinear programming problem only requires $\mathcal{O}(nk)$ memory and existing nonlinear programming techniques allow us to scale to large problems.

After we get a solution, the solution $\boldsymbol{Y}$ can be regarded as the normalized assignment matrix

$$\boldsymbol{Y} = \boldsymbol{W}\hat{\boldsymbol{U}}$$

where $\hat{\boldsymbol{U}} = [\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_k]$, and $\hat{\mathbf{u}}_c = \mathbf{u}_c/\sqrt{s_c}$ for any $c = 1, \ldots, k$.

4.3.1 Solving the NEO-K-Means Low-Rank SDP

To solve the NEO-LR problem (4.3), we use an augmented Lagrangian framework. This is an iterative strategy where each step consists of minimizing an augmented Lagrangian of the problem that includes a current estimate of the Lagrange multipliers for the constraints as well as a penalty term that drives the solution towards the feasible set. Augmented Lagrangian techniques have been successful in previous studies of low-rank SDP approximations [21].

Let $\lambda = [\lambda_1; \lambda_2; \lambda_3]$ be the Lagrange multipliers associated with the three scalar constraints $(s), (u), (w)$, and $\boldsymbol{\mu}$ and $\boldsymbol{\gamma}$ be the Lagrange multipliers associated with the vector constraints (t) and (v), respectively. Let $\sigma \geq 0$ be a penalty

parameter. The augmented Lagrangian for (4.3) is:

$$\begin{aligned}
&\mathcal{L}_A(Y, \mathbf{f}, \mathbf{g}, \mathbf{s}, r; \lambda, \boldsymbol{\mu}, \boldsymbol{\gamma}, \sigma) = \\
&\quad \underbrace{\mathbf{f}^T\mathbf{d} - \text{trace}(\boldsymbol{Y}^T\boldsymbol{K}\boldsymbol{Y})}_{\text{the objective}} \\
&\quad - \lambda_1(\text{trace}(\boldsymbol{Y}^T\boldsymbol{W}^{-1}\boldsymbol{Y}) - k) \\
&\qquad + \frac{\sigma}{2}(\text{trace}(\boldsymbol{Y}^T\boldsymbol{W}^{-1}\boldsymbol{Y}) - k)^2 \\
&\quad - \boldsymbol{\mu}^T(\boldsymbol{Y}\boldsymbol{Y}^T\mathbf{e} - \boldsymbol{W}\mathbf{f}) \\
&\qquad + \frac{\sigma}{2}(\boldsymbol{Y}\boldsymbol{Y}^T\mathbf{e} - \boldsymbol{W}\mathbf{f})^T(\boldsymbol{Y}\boldsymbol{Y}^T\mathbf{e} - \boldsymbol{W}\mathbf{f}) \\
&\quad - \lambda_2(\mathbf{e}^T\mathbf{f} - (1+\alpha)n) + \frac{\sigma}{2}(\mathbf{e}^T\mathbf{f} - (1+\alpha)n)^2 \\
&\quad - \boldsymbol{\gamma}^T(\mathbf{f} - \mathbf{g} - \mathbf{s}) + \frac{\sigma}{2}(\mathbf{f} - \mathbf{g} - \mathbf{s})^T(\mathbf{f} - \mathbf{g} - \mathbf{s}) \\
&\quad - \lambda_3(\mathbf{e}^T\mathbf{g} - (1-\beta)n - r) \\
&\qquad + \frac{\sigma}{2}(\mathbf{e}^T\mathbf{g} - (1-\beta)n - r)^2
\end{aligned} \tag{4.4}$$

At each step in the augmented Lagrangian solution framework, we solve the following subproblem:

$$\begin{aligned}
\text{minimize} \quad & \mathcal{L}_A(\boldsymbol{Y}, \mathbf{f}, \mathbf{g}, \mathbf{s}, r; \lambda, \boldsymbol{\mu}, \boldsymbol{\gamma}, \sigma) \\
\text{subject to} \quad & Y_{ij} \geq 0, \mathbf{s} \geq 0, r \geq 0, \\
& 0 \leq \mathbf{f} \leq k\mathbf{e}, 0 \leq \mathbf{g} \leq 1.
\end{aligned} \tag{4.5}$$

We use a limited-memory BFGS with bound constraints algorithm [24] to minimize the subproblem with respect to the variables $\boldsymbol{Y}$, $\mathbf{f}$, $\mathbf{g}$, $\mathbf{s}$ and r. This requires computation of the gradient of $\mathcal{L}_A$ with respect to the variables. We determine and validate an analytic form for the gradient in [43]. In Chapter 4.5.1, we provide evidence that our optimization procedure is correctly implemented. Those experiments also show that we achieve the same objective function values as the convex formulation (4.2) in a small fraction of the time.

Algorithm 6 Rounding $\boldsymbol{Y}$ to a binary matrix $\boldsymbol{U}$

Input: $\boldsymbol{Y}$, $\boldsymbol{W}$, $\mathbf{f}$, $\mathbf{g}$, α, β
Output: $\boldsymbol{U}$

Update $\boldsymbol{Y} = \boldsymbol{W}^{-1}\boldsymbol{Y}$
Set $\mathcal{D}$ to be the largest $(n - \beta n)$ coordinates of $\mathbf{g}$
for each entry i in $\mathcal{D}$ **do**
    Set $\mathcal{S}$ to be the top $\lfloor f_i \rfloor$ entries in $Y(i, :)$
    Set $U(i, \mathcal{S}) = 1$ /* Assign i to $\mathcal{S}$ */
end for
Set $\bar{\mathbf{f}} = \mathbf{f} - \lfloor \mathbf{f} \rfloor$
Set $\mathcal{R}$ to be the largest entries in $\bar{\mathbf{f}}$
for each entry i in $\mathcal{R}$ **do**
    Pick a cluster ℓ where $Y(i, \ell)$ is the maximun over all clusters where i is not currently assigned
    Set $U(i, \ell) = 1$
end for

4.3.2 Rounding Procedure

Solutions from the the LRSDP method are real-valued. We need to convert $\boldsymbol{Y}$ into a binary assignment matrix $\boldsymbol{U}$ through a rounding procedure. Both the vectors $\mathbf{f}$ and $\mathbf{g}$ provide important information about the solution. Namely, $\mathbf{f}$ gives us a good approximation to the number of clusters each data point is assigned to, and $\mathbf{g}$ indicates which data points are not assigned to any cluster.

The procedure we use for rounding solutions $\boldsymbol{Y}$ that arise when we run LRSDP on a unweighted kernel matrix $\boldsymbol{K}$ is given by Algorithm 6. It uses the largest $n - \beta n$ entries of the vector $\mathbf{g}$ to determine the set of nodes to assign first. Each data point i is assigned to $\lfloor f_i \rfloor$ clusters based on the values in the ith row of $\boldsymbol{Y}$. The remaining assignments are all based on the largest residual elements in $\mathbf{f} - \lfloor \mathbf{f} \rfloor$.

For our experiments with overlapping community detection, we found the following simple alternative rounding strategy more successful. Select the top $(1 + \alpha)n$ entries in $\boldsymbol{W}^{-1}\boldsymbol{Y}$ as the clustering assignment.

4.3.3 Practical Improvements

Finally, we describe a set of practical improvements for our method. These are designed to accelerate the convergence of the augmented Lagrangian framework by moving it closer to a point that satisfies the constraints and is nearly optimal. They are designed based on commonly used strategies in the relax-and-round approach to discrete optimization problems.

Final rounding. At the conclusion of our rounding procedure, we have an assignment of points to clusters. We then use that as the initial cluster assignments for the iterative NEO-K-Means procedure. Since that procedure has monotone convergence behavior, this can only improve the solution.

Initialization. We run the iterative NEO-K-Means algorithm multiple times and use the result with the best objective function value as the initialization to LRSDP. For problems over a few hundred data points, this procedure results in faster convergence and better final solutions.

Sampling. For vector datasets without feature maps, we found that first using LRSDP on sampling 10% of the data points, then using this LRSDP solution as an initialization of the iterative algorithm produces similarly good results as using LRSDP on all the data points while taking significantly less time.

Hierarchical results. For overlapping community detection on large graph data (e.g., the HepPh and AstroPh datasets we show later), we apply a two-level hierarchical clustering. In the first level, we use LRSDP with $k' = \sqrt{k}$, $\alpha' = \sqrt{1+\alpha}-1$ and unchanged β, then in the second level, we run LRSDP with k', α' and $\beta' = 0$ for each cluster at level 1. These parameter settings produce a final assignment result with a total of $(1+\alpha)n$ assignments in k clusters.

4.4 Related Work

This work is most strongly related to convex relaxations of the k-means objective [49] and related SDP formulations of k-means [67], [68]. For instance, [49] employs the same general strategy of using a low-rank factorization of the SDP for k-means in concert with an augmented Lagrangian solver for the resulting nonlinear optimization problem. Even more generally, our work fits into the broad setting of convex relaxations of clustering problems including normalized cut objectives [84].

Recently, there was a proposal for a different type of convex clustering method [54], [41] which is also based on k-means. The key difference is that these relaxations model a centroid point for each data point and then attempt to penalize differences among the centroids. It is related to the lasso and the fused lasso procedures. As a convex optimization problem, it suffers the same issues as the existing SDP relaxations of k-means, namely, a quadratic number of variables to optimize.

Using augmented Lagrangian methods to solve low-rank factorizations of SDP solutions has a long history of delivering successful performance when the data arise from graphs. For instance, [21] originally proposed this idea for the MAX-CUT and minimum bisection SDPs. Later, similar ideas were used to address key weaknesses in spectral clustering [51] on power-law graphs.

4.5 Experimental Results

We begin by validating our implementation and comparing our solutions against the global optima from the CVX program. We then show the effectiveness of LRSDP as an initialization method of the iterative NEO-K-Means algorithm which is a simple greedy algorithm designed for optimizing the NEO-K-Means objective function. Finally, we show experimental results on vector and graph clustering problems by comparison with state-of-the-art clustering and community detection

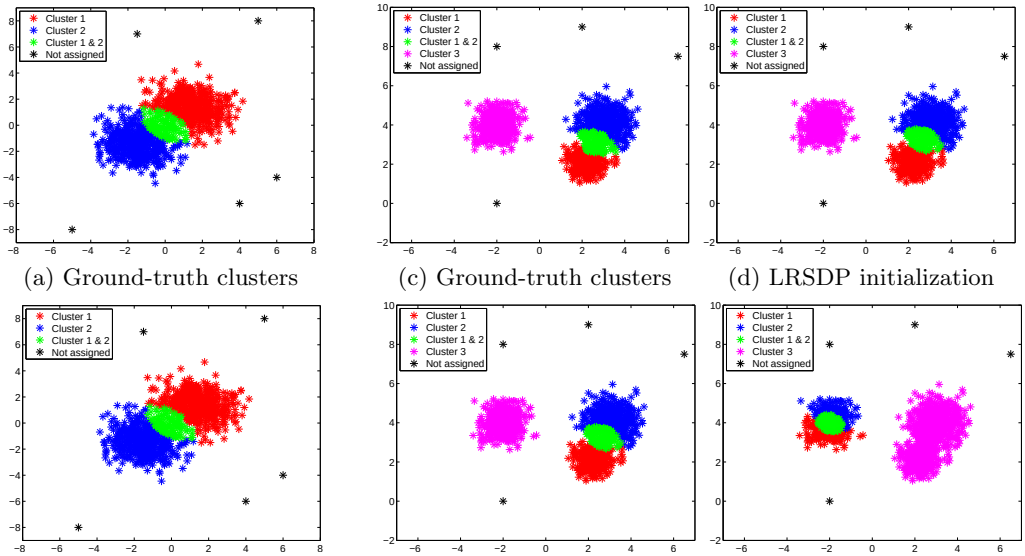
Table 4.2: Comparison of SDP and LRSDP (objective value and run time). The small differences between the objective values are the result of differences in solution tolerances and precision in the sub-problems.

		Objective value		Run time	
		SDP	LRSDP	SDP	LRSDP
dolphins	k=2, α=0.2, β=0	-1.968893	-1.968329	107.03 secs.	2.55 secs.
	k=2, α=0.2, β=0.05	-1.969080	-1.968128	56.99 secs.	2.96 secs.
	k=3, α=0.3, β=0	-2.913601	-2.915384	160.57 secs.	5.39 secs.
	k=3, α=0.3, β=0.05	-2.921634	-2.922252	71.83 secs.	8.39 secs.
les miserables	k=2, α=0.2, β=0	-1.937268	-1.935365	453.96 secs.	7.10 secs.
	k=2, α=0.3, β=0	-1.949212	-1.945632	447.20 secs.	10.24 secs.
	k=3, α=0.2, β=0.05	-2.845720	-2.845070	261.64 secs.	13.53 secs.
	k=3, α=0.3, β=0.05	-2.859959	-2.859565	267.07 secs.	19.31 secs.

methods.

4.5.1 Algorithmic Validation

We measure the objective function values produced by LRSDP compared with the convex formulation of the problem and solved by CVX. We consider two graph clustering problems using the 'dolphins' [58] and 'les miserables' [48] datasets. The 'dolphins' network represents frequent associations between 62 dolphins (there are 159 undirected edges in the network), and 'les miserables' network represents the co-appearance of characters in the novel Les Miserables (there are 77 nodes and 254 edges). Table 4.2 shows the results. We try a set of different configurations with k, α, and β. We compare the run time of CVX solver and LRSDP and find that LRSDP is roughly an order of magnitude faster than CVX. In Table 4.2, we report the objective values before the relaxed solution is rounded to a discrete assignment solution to precisely measure how much our solution is different from the solution returned by CVX. We can see that the objective values returned from CVX and returned from our LRSDP solver are essentially identical—they are different in light of the solution tolerances given by the methods. *Therefore, in these cases, we are successful in finding a globally optimal solution.*



(a) Ground-truth clusters (c) Ground-truth clusters (d) LRSDP initialization

(b) k-means/LRSDP initialization (e) Success of k-means initialization (f) Failure of k-means initialization

Figure 4.2: The output of NEO-K-Means algorithm with two different initialization methods on two synthetic datasets. (a) & (b) On a simple dataset, NEO-K-Means can easily recover the ground-truth clusters with k-means or LRSDP initialization. (c)–(f) LRSDP initialization allows the NEO-K-Means algorithm to consistently produce a reasonable clustering structure whereas k-means initialization sometimes (4 times out of 10 trials) leads to a failure in recovering the underlying clustering structure.

4.5.2 Motivating Example

Now, we show how we can exploit the benefit of LRSDP by using it as an initialization of the simple iterative NEO-K-Means algorithm. We consider two synthetic datasets shown in Figure 4.2(a) & Figure 4.2(c). In these datasets, green data points indicate the overlapped region between clusters, and black data points indicate outliers which are not supposed to belong to any cluster. The first dataset was considered in Chapter 3. We run the iterative NEO-K-Means algorithm on these datasets with two different initialization methods: k-means and LRSDP. On a simpler dataset, Figure 4.2(a), we observe that the NEO-K-Means can always recover the underlying clustering structure regardless of the initialization methods.

Table 4.3: Real-world vector datasets.

	n	dim.	$\lvert\bar{\mathcal{C}}\rvert$	k
yeast	2,417	103	731.5	14
music	593	72	184.7	6
scene	2,407	294	430.8	6

However, on Figure 4.2(c), we observe the advantages of LRSDP over the k-means initialization. When we use the LRSDP initialization, the NEO-K-Means always yields a similar clustering structure as the ground-truth clusters as shown in Figure 4.2(d). On the other hand, when the k-means initialization is used, the NEO-K-Means fails to recover the underlying clustering structure 4 times out of 10 trials as shown in Figure 4.2(f). Thus, we see that on more complicated datasets, the dangers of bad initialization and being stuck in local minima become clearer, and LRSDP provides a more stable initialization, which enables the NEO-K-Means algorithm to consistently produce a reasonable clustering.

4.5.3 Data Clustering

We show some experimental results on real-world vector datasets. We use three multi-label datasets which we get from [1]. Table 4.3 presents some basic statistics of these datasets ('dim.' denotes the dimensionality of the vectors and $|\bar{\mathcal{C}}|$ denotes the average size of the ground-truth clusters). As we did in Chapter 3, on these datasets, we treat each label as a ground-truth cluster.

To see the effectiveness of our LRSDP method, we compare LRSDP using a final iterative NEO-K-Means improvement step. This method is denoted by 'lrsdp+neo'. Also, we used the sampling method with 10% of the data points. This method is denoted by 'slrsdp+neo'. We compare these LRSDP approaches with the iterative NEO-K-Means initialized by the traditional k-means (denoted by 'kmeans+neo'). We run each method five times, and Table 4.4 shows the best, worst, average, and the standard deviation of the NEO-K-Means objective function values.

Table 4.4: Comparison of NEO-K-Means objective function values.

		worst	best	avg.±std.
YEAST	kmeans+neo	9611	9495	9549 ± 51
	lrsdp+neo	**9440**	9280	9364 ± 60
	slrsdp+neo	9471	**9231**	9367 ± 90
MUSIC	kmeans+neo	87779	70158	77015 ± 7658
	lrsdp+neo	**82323**	**70157**	75923 ± 5936
	slrsdp+neo	82336	70159	75926 ± 5940
SCENE	kmeans+neo	18905	**18745**	18806 ± 66
	lrsdp+neo	18904	18759	18811 ± 58
	slrsdp+neo	**18895**	18760	18810 ± 55

Within all these methods, α and β values are automatically detected as described in Chapter 3. A lower objective value indicates a better clustering. We can see that there is a significant difference in the objective value between 'kmeans+neo' and LRSDP methods ('lrsdp+neo' and 'slrsdp+neo') on 'yeast' and 'music' datasets. By using the LRSDP solution as the initialization of the iterative algorithm, we can achieve a better objective function value for two of the datasets. This implies that LRSDP is effective in optimizing the NEO-K-Means objective, and thus provides a good initialization of the iterative algorithm. We note that the benefit of LRSDP on 'scene' dataset is not significant, but we also note that on this dataset, the average behavior of all methods is roughly the same. In this case, the overlaps among the ground-truth clusters are very small (the ground-truth α is 0.074) which implies that the traditional k-means should be a highly accurate initialization.

We also compare the clustering performance with other state-of-the-art clustering methods including model-based overlapping clustering [11], denoted by *moc*, explicit/implicit sparsity constrained clustering [57], denoted by *esp*, and *isp*, respectively, and overlapping k-means [29], denoted by *okm*. All these clustering methods are initialized by k-means, and executed five times. To see the clustering performance, we compute the F_1 score which measures the matching between algorithmic solutions and the ground-truth clusters. Higher F_1 scores indicate improved

Table 4.5: F_1 scores on real-world vector datasets.

		worst	best	avg.±std.
YEAST	*moc*	N/A	N/A	N/A
	esp	0.274	0.289	0.284±0.006
	isp	0.232	0.256	0.248±0.010
	okm	0.311	0.323	0.317±0.004
	kmeans+neo	0.356	0.366	0.360±0.004
	lrsdp+neo	**0.390**	**0.391**	0.391±0.001
	slrsdp+neo	0.369	**0.391**	0.382±0.011
MUSIC	*moc*	0.530	0.544	0.538±0.006
	esp	0.514	0.539	0.526±0.011
	isp	0.506	0.539	0.517±0.013
	okm	0.524	0.531	0.527±0.003
	kmeans+neo	0.526	0.551	0.543±0.011
	lrsdp+neo	0.537	**0.552**	0.545±0.008
	slrsdp+neo	**0.541**	**0.552**	0.547±0.005
SCENE	*moc*	0.466	0.470	0.467±0.002
	esp	0.569	0.582	0.575±0.005
	isp	0.586	0.609	0.598±0.010
	okm	0.571	0.576	0.573±0.002
	kmeans+neo	0.597	**0.627**	0.610±0.015
	lrsdp+neo	**0.610**	0.614	0.613±0.002
	slrsdp+neo	0.605	0.625	0.613±0.008

matches with the ground-truth clusters. Table 4.5 shows F_1 scores of each algorithm on the real-world datasets. On the 'yeast' dataset, *moc* produces 13 empty clusters and one cluster which contains all the data points, so we cannot report F_1 score of *moc* on this dataset. We first note that the NEO-K-Means-based methods ('kmeans+neo', 'lrsdp+neo', and 'slrsdp+neo') are consistently better than the other clustering methods; *and, the LRSDP methods are able to achieve better F_1 scores than the other methods.*

4.5.4 Overlapping Community Detection

The iterative NEO-K-Means method and our new LRSDP method can both be used for overlapping community detection because optimizing the NEO-K-Means

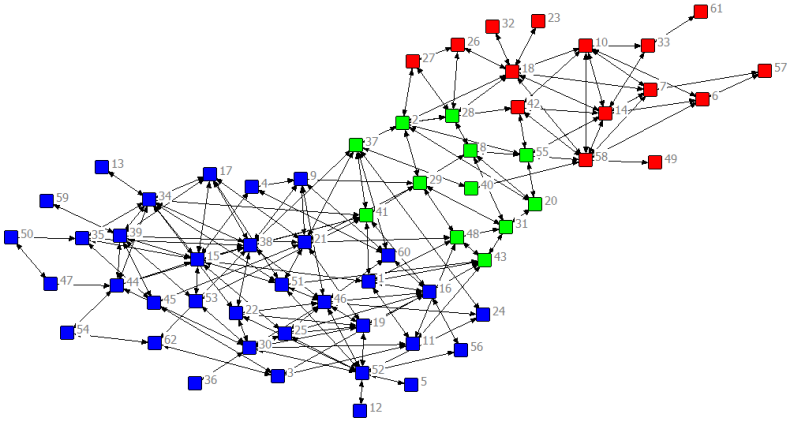


Figure 4.3: Visualization of the clustering result of LRSDP on 'dolphins' network. Blue nodes only belong to cluster 1, red nodes only belong to cluster 2, and green nodes belong to both of the clusters.

Table 4.6: Real-world network datasets.

	No. of vertices	No. of edges
Facebook1	348	2,866
Facebook2	756	30,780
HepPh	11,204	117,619
AstroPh	17,903	196,972

objective function corresponds to optimizing an extended version of normalized cut. To see whether LRSDP produces a reasonable clustering structure on graphs, we visualize the clustering result of LRSDP (k=2, α=0.2, β=0) on the 'dolphins' network [58] in Figure 4.3. There are two clusters where green nodes indicate the overlapped region (blue and green nodes form one cluster, and red and green nodes form the other cluster). Notice that the green nodes have many interactions with both of the clusters, which shows that LRSDP produces a plausible solution aligned with an intuitive clustering structure.

Next, we consider real-world networks from [4]. We use four different networks which are summarized in Table 4.6. Facebook1 and Facebook2 are social

Table 4.7: Average normalized cut of the iterative multilevel NEO-K-Means and LRSDP

	multilevel neo	LRSDP
Facebook1	0.371	**0.279**
Facebook2	0.331	**0.223**
HepPh	0.185	**0.169**
AstroPh	0.240	**0.201**

networks, and HepPh and AstroPh are collaboration networks. To run LRSDP on the two large networks, HepPh and AstroPh, we use a hierarchical clustering which we discussed in Chapter 4.3.3. Table 4.7 shows the comparison of the average normalized cut between the multilevel NEO-K-Means algorithm and LRSDP. The multilevel NEO-K-Means (denoted by 'multilevel neo' or 'm-neo') is a variation of the iterative NEO-K-Means algorithm where the graph clustering problem is solved at multiple scales. We also use the multilevel NEO-K-Means as the final improvement step of LRSDP as we briefly discussed in Chapter 4.3.3. We see that LRSDP achieves the lower normalized cut than the multilevel NEO-K-Means, which indicates that LRSDP is beneficial to optimizing the objective function. Within these methods, we set k=32, α=3, β=0 on Facebook networks. On large networks, we determine α and β values based on the statistics of the output of NISE method.

We also compare with other state-of-the-art overlapping community detection methods including *demon* [30], *bigclam* [85], *oslom* [50], and NISE (presented in Chapter 2). Let us first note that the runtime of LRSDP is competitive with other state-of-the-art approaches. For example, on the HepPh network with k=100, LRSDP took 18 minutes whereas *oslom* method took 19 minutes and *bigclam* method took 11 minutes. On the other hand, the multilevel NEO-K-Means algorithm completed in less than 10 seconds. Thus, our approaches and algorithms would be more suitable for applications where getting a high-quality clustering is more important than getting faster results. This is the case, for instance, in modern biology and

Table 4.8: AUC of conductance-vs-graph coverage

	Facebook1	Facebook2	HepPh	AstroPh
bigclam	0.830	0.640	0.625	0.645
demon	0.495	0.318	0.503	0.570
oslom	0.319	0.445	0.465	0.580
NISE	0.297	0.293	0.102	0.153
m-neo	0.285	0.269	0.206	0.190
LRSDP	**0.222**	**0.148**	**0.091**	**0.137**

neuroscience data. A recently collected network of the rat brain required "more than 4,000 hours to compile" [3]. On this time scale, the quality of the final results is paramount.

We evaluate the quality of communities based on the conductance score which is one of the most commonly used metrics to evaluate the cohesiveness of communities. In particular, we compute the area under the curve (AUC) in a plot of conductance-vs-graph coverage as we did in Chapter 2. A lower AUC score indicates a better clustering. Table 4.8 shows the results. We can see that LRSDP achieves the lowest AUC score across all the datasets, which implies that it produces the most coherent communities.

4.6 Further Extension: Fast Multiplier Methods

There have been recent developments in fast alternating methods and proximal methods for convex and nearly convex objectives that arise in machine learning. We recently propose two variations on the low-rank approximation (4.3) that can utilize some of these techniques for even more scalability. In particular, we propose two fast multiplier methods to accelerate the convergence of an augmented Lagrangian scheme: a proximal method of multipliers and an alternating direction method of multipliers (ADMM). For the proximal augmented Lagrangian or proximal method of multipliers, we show a convergence result for the non-convex case with bound-

constrained subproblems. These methods are up to 13 times faster—with no change in quality—compared with a standard augmented Lagrangian method on problems with over 10,000 variables bringing runtimes down from over an hour to around 5 minutes. This work is currently under review [42].

Chapter 5

Non-exhaustive, Overlapping Co-clustering

We now discuss how we can extend our NEO-K-Means ideas to co-clustering problems. The goal of co-clustering is to simultaneously identify a clustering of the rows as well as the columns of a data matrix. Indeed, the co-clustering problem can be thought as a clustering of a bipartite graph. For example, in recommender systems, users have ratings on items and this can be represented by a bipartite graph where users and items are denoted by two different types of nodes, and the ratings are denoted by weighted edges between the users and the items. In this setting, traditional co-clustering algorithms find a disjoint and exhaustive clustering of each type of the nodes (e.g., clustering of users and clustering of items), and the clusterings of the two different types of the nodes are produced simultaneously.

As we described the NEO-K-Means problem in the previous chapters, in real-world datasets, the clusters may overlap with each other, and there are often outliers that should not belong to any cluster. To find overlapping clusters as well as outliers on both of the two different types of the nodes in the bipartite graph, we formulate the Non-Exhaustive, Overlapping Co-Clustering (NEO-Co-Clustering)

problem. To encode this problem mathematically, we extend our NEO-K-Means objective function to the co-clustering setting. In our NEO-Co-Clustering objective function, the main idea is to minimize the distance between each entry in a two-dimensional data matrix and the mean of its co-clusters.

To optimize the new objective function, we also develop a simple iterative algorithm we call the NEO-Co-Clustering algorithm (NEO-CC in short). The NEO-CC algorithm adopts an alternating minimization strategy, and we prove that it monotonically decreases the NEO-Co-Clustering objective function. Finally, we conduct experiments on micro-array gene expression data and user-movie ratings data and observe that our NEO-CC algorithm is able to effectively capture the underlying co-clustering structure of the data, which results in boosting the performance of a standard one-dimensional clustering.

5.1 Preliminaries

We first describe the minimum sum-squared residue co-clustering objective function [28] which is an intuitive co-clustering objective function for the traditional disjoint and exhaustive co-clustering. Also, we briefly review the NEO-K-Means objective function.

5.1.1 Minimum Sum-Squared Residue Co-Clustering

Let us consider a two-dimensional data matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$. In the traditional co-clustering setting, the problem is to partition $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ into k row clusters and l column clusters. To denote each dimension of the data matrix, let $\mathcal{X}^r$ denote the set of data points for row clustering, and $\mathcal{X}^c$ denote the set of data points for column clustering. Then, the co-clustering problem is to cluster $\mathcal{X}^r = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ into $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$, and cluster $\mathcal{X}^c = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$ into $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$ where the clusters are pairwise disjoint and every data point is assigned to some cluster. Let

us define the following functions:

$$f : \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\} \rightarrow \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k\}$$

$$g : \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \rightarrow \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_l\}$$

where $f(\mathbf{x}_i) = \mathcal{C}_p$ indicates a data point $\mathbf{x}_i \in \mathcal{X}^r$ belongs to a row cluster $\mathcal{C}_p$; $g(\mathbf{x}_j) = \mathcal{C}_q$ indicates a data point $\mathbf{x}_j \in \mathcal{X}^c$ belongs to a column cluster $\mathcal{C}_q$.

The minimum sum-squared residue (MSSR) co-clustering [28] objective function considers the difference between each entry in the co-cluster and the mean of the co-cluster, which can be represented by

$$x_{ij} - \frac{\sum_{\mathbf{x}_p \in f(\mathbf{x}_i), \mathbf{x}_q \in g(\mathbf{x}_j)} x_{pq}}{|f(\mathbf{x}_i)||g(\mathbf{x}_j)|},$$

where x_{ij} denotes the i-th row and the j-th column entry in $\boldsymbol{X}$. Let $\boldsymbol{U} = [u_{ij}]_{n \times k}$ denote the assignment matrix for row clustering, and $\boldsymbol{V} = [v_{ij}]_{m \times l}$ denote the assignment matrix for column clustering. Also, let $\hat{\boldsymbol{U}} = [\frac{\mathbf{u}_1}{\sqrt{n_1}}, \cdots, \frac{\mathbf{u}_k}{\sqrt{n_k}}]$ denote a normalized assignment matrix where $\mathbf{u}_c$ is the c-th column of $\boldsymbol{U}$ and n_c is the size of cluster c. Then, the MSSR co-clustering objective function is defined to be

$$\underset{\boldsymbol{U}, \boldsymbol{V}}{\text{minimize}} \quad \|\boldsymbol{X} - \hat{\boldsymbol{U}}\hat{\boldsymbol{U}}^T \boldsymbol{X} \hat{\boldsymbol{V}}\hat{\boldsymbol{V}}^T\|_F^2. \tag{5.1}$$

This MSSR co-clustering objective function has a close relationship with the standard k-means objective function in the sense that if each column vector (i.e., $\mathbf{x}_j \in \mathcal{X}^c$) is in a cluster by itself, (5.1) is the k-means objective function for the clustering of $\mathcal{X}^r$.

5.1.2 Revisit of the NEO-K-Means Objective

Let us briefly review the NEO-K-Means objective function presented in Chapter 3. Note that the NEO-K-Means objective function is intended to encode one-sided clustering. The NEO-K-Means objective function is defined as follows:

$$\begin{aligned} &\underset{\boldsymbol{U}}{\text{minimize}} \sum_{j=1}^{k}\sum_{i=1}^{n} u_{ij}\|\mathbf{x}_i - \mathbf{m}_j\|^2, \text{ where } \mathbf{m}_j = \frac{\sum_{i=1}^{n} u_{ij}\mathbf{x}_i}{\sum_{i=1}^{n} u_{ij}} \\ &\text{subject to } \text{trace}(\boldsymbol{U}^T\boldsymbol{U}) = (1+\alpha)n, \sum_{i=1}^{n} \mathbb{I}\{(\boldsymbol{U}\mathbf{1})_i = 0\} \leq \beta n, \end{aligned} \tag{5.2}$$

where there are n data points and k clusters, $\mathbf{x}_i$ denotes the i-th data point, $\boldsymbol{U} = [u_{ij}]_{n\times k}$ such that $u_{ij} = 1$ if $\mathbf{x}_i$ belongs to cluster j; $u_{ij} = 0$ otherwise, and $\mathbb{I}\{exp\} = 1$ if exp is true; 0 otherwise. The parameter α controls the amount of overlap, and β controls the degree of non-exhaustiveness. The parameters imply that the total number of assignments in $\boldsymbol{U}$ is equal to $(1+\alpha)n$ and at least $(1-\beta)n$ data points should belong to some clusters (i.e., there can be at most βn outliers). Thus, $(1+\alpha)n$ should be greater than or equal to $(1-\beta)n$. This leads to $\alpha + \beta \geq 0$.

5.2 The NEO-Co-Clustering Objective

We now discuss how we can design a reasonable objective function for the non-exhaustive, overlapping co-clustering problem where the clusters are allowed to overlap with each other, and outliers are not allowed to be assigned to any cluster. Formally, $\exists i \neq j$ such that $\mathcal{C}_i \cap \mathcal{C}_j \neq \emptyset$, $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_k \subseteq \mathcal{X}^r$, and $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_l \subseteq \mathcal{X}^c$.

Given an element x_{ij} in $\boldsymbol{X}$, let $f(\mathbf{x}_i)$ denote the set of row clusters that $\mathbf{x}_i$ belongs to $(i = 1, \ldots, n)$, and let $g(\mathbf{x}_j)$ denote the set of column clusters that $\mathbf{x}_j$ belongs to $(j = 1, \ldots, m)$. Then, similar to the MSSR co-clustering objective function, we consider the difference between x_{ij} and the mean of its co-clusters as

follows:

$$\sum_{\mathcal{C}_q \in g(\mathbf{x}_j)} \sum_{\mathcal{C}_p \in f(\mathbf{x}_i)} \big(x_{ij} - \sum_{\mathbf{x}_t \in \mathcal{C}_q} \sum_{\mathbf{x}_s \in \mathcal{C}_p} \frac{x_{st}}{|\mathcal{C}_q||\mathcal{C}_p|}\big)^2 \tag{5.3}$$

for x_{ij} such that $f(\mathbf{x}_i) \neq \emptyset$ and $g(\mathbf{x}_j) \neq \emptyset$. Note that since each data point can belong to multiple row and column clusters, we need to consider all the combinations of these row and column clusters when we compute the mean of the co-clusters.

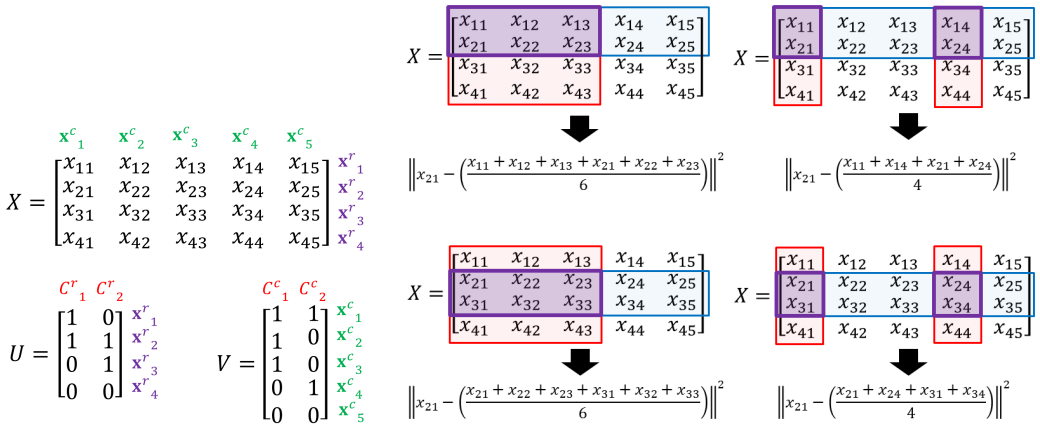
Now, let us represent the idea of (5.3) using vectors and matrices. Given a vector $\mathbf{y} \in \mathbb{R}^m$, let us define $D(\mathbf{y}) = [d_{ij}]_{m \times m}$ as the diagonal matrix with $d_{ii} = y_i$ $(i = 1, \ldots, m)$. Then, our NEO-Co-Clustering Objective is defined as follows:

$$\begin{aligned} \underset{\boldsymbol{U},\boldsymbol{V}}{\text{minimize}} \quad & \sum_{i=1}^{k} \sum_{j=1}^{l} \|D(\mathbf{u}_i)\boldsymbol{X}D(\mathbf{v}_j) - \hat{\mathbf{u}}_i \hat{\mathbf{u}}_i^T \boldsymbol{X} \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T\|_F^2 \\ \text{subject to} \quad & \text{trace}(\boldsymbol{U}^T\boldsymbol{U}) = (1+\alpha_r)n, \\ & \textstyle\sum_{i=1}^{n} \mathbb{I}\{(\boldsymbol{U}\mathbf{1})_i = 0\} \leq \beta_r n, \\ & \text{trace}(\boldsymbol{V}^T\boldsymbol{V}) = (1+\alpha_c)m, \\ & \textstyle\sum_{i=1}^{m} \mathbb{I}\{(\boldsymbol{V}\mathbf{1})_i = 0\} \leq \beta_c m, \end{aligned} \tag{5.4}$$

where α_r and β_r are the parameters for row clustering, and α_c and β_c are the parameters for column clustering.

To explain the implication of (5.4), we show an illustrative example in Figure 5.1. Let us consider a small data matrix $\boldsymbol{X} \in \mathbb{R}^{4\times5}$ and the assignment matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ as shown in Figure 5.1 (a). For an entry x_{21}, Figure 5.1 (b) shows the contribution of the entry x_{21} to the NEO-Co-Clustering objective in (5.4) when $\mathbf{x}_2^r \in \mathcal{C}_1^r$, $\mathbf{x}_2^r \in \mathcal{C}_2^r$ $(\mathbf{x}_2^r \in \mathbb{R}^5)$, $\mathbf{x}_1^c \in \mathcal{C}_1^c$, $\mathbf{x}_1^c \in \mathcal{C}_2^c$ $(\mathbf{x}_1^c \in \mathbb{R}^4)$.

The NEO-Co-Clustering objective function seamlessly generalizes the NEO-K-Means objective function and the MSSR objective function. If $\boldsymbol{V} = \boldsymbol{I}$, $\alpha_c = 0$, $\beta_c = 0$, then (5.4) is equivalent to the NEO-K-Means objective (5.2). If $\alpha_r = 0$, $\alpha_c = 0$, $\beta_r = 0$, $\beta_c = 0$, then (5.4) is equivalent to the MSSR objective (5.1).



(a) Data matrix $\boldsymbol{X}$, row clustering $\boldsymbol{U}$, and column clustering $\boldsymbol{V}$

(b) The contribution of x_{21} to the NEO-Co-Clustering objective

Figure 5.1: Given a small data matrix $\boldsymbol{X} \in \mathbb{R}^{4\times 5}$ and the assignment matrices for row & column clusterings $\boldsymbol{U}$ & $\boldsymbol{V}$, the contribution of an entry x_{21} to the NEO-Co-Clustering objective in (5.4) is determined by $f(\mathbf{x}_2^r)$ and $g(\mathbf{x}_1^c)$. Note that $\mathbf{x}_2^r \in \mathcal{C}_1^r$, $\mathbf{x}_2^r \in \mathcal{C}_2^r$ ($\mathbf{x}_2^r \in \mathbb{R}^5$), $\mathbf{x}_1^c \in \mathcal{C}_1^c$, $\mathbf{x}_1^c \in \mathcal{C}_2^c$ ($\mathbf{x}_1^c \in \mathbb{R}^4$).

Now we conduct a small case study on the following data matrix $\boldsymbol{X}$:

$$\boldsymbol{X} = \begin{bmatrix} 0.05 & 0.05 & 0.05 & 0 & 0 & 0 \\ 0.05 & 0.05 & 0.05 & 0 & 0 & 0 \\ 0.04 & 0.04 & 0.04 & 0 & 0.04 & 0.04 \\ 0.04 & 0.04 & 0 & 0.04 & 0.04 & 0.04 \\ 0 & 0 & 0 & 0.05 & 0.05 & 0.05 \\ 0 & 0 & 0 & 0.05 & 0.05 & 0.05 \\ 0 & 0 & 0.3 & 0 & 0 & 0 \end{bmatrix} \tag{5.5}$$

On $\boldsymbol{X}$, we consider four different co-clustering results and investigate how the NEO-Co-Clustering objective function values are changed. Figure 5.2 shows the NEO-Co-Clustering objective values for the four different configurations. In Figure 5.2 (a) & (b), we consider exhaustive, disjoint clusterings with different k. Notice that on these configurations, $nnz(\boldsymbol{U}) + nnz(\boldsymbol{V}) = 13$. We observe that

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

(a) Objective value: 0.0720

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

(b) Objective value: 0.0677

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

(c) Objective value: 0.0137

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

(d) Objective value: 0.0102

Figure 5.2: NEO-Co-Clustering objective values defined in (5.4) for four different co-clustering results

(b) yields a slightly small objective value than (a). In Figure 5.2 (c), $\mathbf{x}_3^r$ and $\mathbf{x}_4^r$ belong to both of the row clusters, and $\mathbf{x}_7^r$ does not belong to any cluster. We see that (c) achieves a much smaller objective function value than (a) & (b). Indeed, when we look at the data matrix $\boldsymbol{X}$, we can realize that (c) is a more desirable co-clustering result than (a) & (b). On the other hand, the NEO-Co-Clustering objective value is proportional to the number of assignments in $\boldsymbol{U}$ and $\boldsymbol{U}$. In (c), we notice that $nnz(\boldsymbol{U}) + nnz(\boldsymbol{V}) = 14$ ($[n\alpha_r] = 1$, $[n\beta_r] = 1$, $\alpha_c = 0$, $\beta_c = 0$). Thus, we also consider (d) where where $nnz(\boldsymbol{U}) + nnz(\boldsymbol{V}) = 13$. In this case, the column clustering becomes a disjoint and non-exhaustive clustering by setting $[n\alpha_r] = 1$, $[n\beta_r] = 1$, $[m\alpha_c] = -1$, $[m\beta_c] = 1$. Notice that $\mathbf{x}_3^c$ is considered to be an outlier, so it does not belong to any cluster. We can see that (d) achieves the smallest objective value and also (d) might be the optimal co-clustering for $\boldsymbol{X}$. On this simple case study, we observe that a smaller NEO-Co-Clustering objective function value leads to a more desirable co-clustering result. This indicates that by optimizing our NEO-Co-Clustering objective, we might be able to capture the natural co-clustering structures from the underlying data.

5.3 The NEO-Co-Clustering Algorithm

To optimize the NEO-Co-Clustering objective function defined in (5.4), we develop a simple iterative algorithm which we call the NEO-Co-Clustering (NEO-CC) algorithm. Algorithm 7 shows the NEO-CC algorithm. The algorithm consists of two main parts – updating row clustering (lines 4–20) and updating column clustering (lines 22–38).

Let us first describe how $\boldsymbol{U}$ is updated. To update $\boldsymbol{U}$, we need to compute distances between every data point in $\mathcal{X}^r$ and the clusters $\mathcal{C}_q^r$ for $q = 1, \cdots, k$ (lines 4–8). Let $[d_{pq}]_{n \times k}$ denote these distances. The distance between a data point $\mathbf{x}_p$ and a cluster $\mathcal{C}_q^r$ is computed by

$$d_{pq} = \sum_{j=1}^{l} \left\| (\boldsymbol{I}_{p\cdot}) \boldsymbol{X} D(\mathbf{v}_j) - \frac{1}{\sqrt{\|\mathbf{u}_q\|_1}} \hat{\mathbf{u}}_q^T \boldsymbol{X} \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T \right\|^2$$

where $\boldsymbol{I}_{p\cdot}$ denotes the p-th row of the identity matrix of size n.

For each data point $\mathbf{x}_p \in \mathcal{X}^r$ $(p = 1, \cdots, n)$, we record the closest cluster and that distance. Then, the data points are sorted in ascending order by the distance to its closest cluster. To satisfy the non-exhaustiveness constraint, we assign the first $(1 - \beta_r)n$ data points to their closest clusters. Then, we make $\alpha_r n + \beta_r n$ more assignments by taking $\alpha_r n + \beta_r n$ minimum distances among $[d_{pq}]_{n \times k}$. In this way, row clustering is updated. Similarly, we can also update column clustering. The row and column clusterings are repeatedly updated until the change in the objective function value is sufficiently small or the maximum number of iterations is reached.

The NEO-CC algorithm also has close connections to the NEO-K-Means algorithm and the MSSR algorithm. If $\boldsymbol{V} = \boldsymbol{I}$, $\alpha_c = 0$, $\beta_c = 0$, then Algorithm 7 is identical to the NEO-K-Means algorithm. If $\alpha_r = 0$, $\alpha_c = 0$, $\beta_r = 0$, $\beta_c = 0$, then Algorithm 7 is identical to the MSSR algorithm.

Algorithm 7 NEO-Co-Clustering Algorithm

Input: $\boldsymbol{X} \in \mathbb{R}^{n \times m}$, k, l, α_r, α_c, β_r, β_c, t_{max}
Output: Row clustering $\boldsymbol{U} \in \{0,1\}^{n \times k}$, Column clustering $\boldsymbol{V} \in \{0,1\}^{m \times l}$

Initialize $\boldsymbol{U}$, $\boldsymbol{V}$, and $t = 0$.
while not converged and $t < t_{max}$ **do**
  /* Update Row Clustering */
  for each $\mathbf{x}_p \in \mathcal{X}^r$ **do**
    for $q = 1, \cdots, k$ **do**
      $d_{pq} = \sum_{j=1}^{l} \left\| (\boldsymbol{I}_{p\cdot})\boldsymbol{X} D(\mathbf{v}_j) - \frac{1}{\sqrt{\|\mathbf{u}_q\|_1}} \hat{\mathbf{u}}_q^T \boldsymbol{X} \hat{\mathbf{v}}_j \hat{\mathbf{v}}_j^T \right\|^2$
    end for
  end for
  Initialize $w = 0$, $\mathcal{T} = \emptyset$, $\mathcal{S} = \emptyset$, and $\bar{\mathcal{C}}_i^r = \emptyset, \hat{\mathcal{C}}_i^r = \emptyset \ \forall i \ (i = 1, \cdots, k)$.
  while $w < (n + \alpha_r n)$ **do**
    if $w < (n - \beta_r n)$ **then**
      Assign $\mathbf{x}_{i^*}^r$ to $\bar{\mathcal{C}}_{j^*}^r$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $, \{(i,j)\} \notin \mathcal{T}, i \notin \mathcal{S}$.
      $\mathcal{S} = \mathcal{S} \cup \{i^*\}$.
    else
      Assign $\mathbf{x}_{i^*}^r$ to $\hat{\mathcal{C}}_{j^*}^r$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}$.
    end if
    $\mathcal{T} = \{(i^*, j^*)\} \cup \mathcal{T}$.
    $w = w + 1$.
  end while
  Update clusters $\mathcal{C}_i^r = \bar{\mathcal{C}}_i^r \cup \hat{\mathcal{C}}_i^r \ \forall i \ (i = 1, \cdots, k)$.
  /* Update Column Clustering */
  for each $\mathbf{x}_p \in \mathcal{X}^c$ **do**
    for $q = 1, \cdots, l$ **do**
      $d_{pq} = \sum_{i=1}^{k} \left\| D(\mathbf{u}_i)\boldsymbol{X}(\boldsymbol{I}_{\cdot p}) - \frac{1}{\sqrt{\|\mathbf{v}_q\|_1}} \hat{\mathbf{u}}_i \hat{\mathbf{u}}_i^T \boldsymbol{X} \hat{\mathbf{v}}_q \right\|^2$
    end for
  end for
  Initialize $w = 0$, $\mathcal{T} = \emptyset$, $\mathcal{S} = \emptyset$, and $\bar{\mathcal{C}}_j^c = \emptyset, \hat{\mathcal{C}}_j^c = \emptyset \ \forall j \ (j = 1, \cdots, l)$.
  while $w < (m + \alpha_c m)$ **do**
    if $w < (m - \beta_c m)$ **then**
      Assign $\mathbf{x}_{i^*}^c$ to $\bar{\mathcal{C}}_{j^*}^c$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}, i \notin \mathcal{S}$.
      $\mathcal{S} = \mathcal{S} \cup \{i^*\}$.
    else
      Assign $\mathbf{x}_{i^*}^c$ to $\hat{\mathcal{C}}_{j^*}^c$ such that $(i^*, j^*) = \underset{i,j}{\operatorname{argmin}}\ d_{ij}$ where $\{(i,j)\} \notin \mathcal{T}$.
    end if
    $\mathcal{T} = \{(i^*, j^*)\} \cup \mathcal{T}$.
    $w = w + 1$.
  end while
  Update clusters $\mathcal{C}_j^c = \bar{\mathcal{C}}_j^c \cup \hat{\mathcal{C}}_j^c \ \forall j \ (j = 1, \cdots, l)$.
  $t = t + 1$.
end while

5.3.1 Convergence Analysis

We can show that Algorithm 7 monotonically decreases the NEO-Co-Clustering objective function. To show the convergence, we need the following lemma.

Lemma 1. *Let us consider the following function*

$$h(\mathbf{z}) = \sum_i \pi_i \left\| \mathbf{a}_i - c\mathbf{z}\boldsymbol{M} \right\|_2^2$$

where $\mathbf{a}_i \in \mathbb{R}^{1\times m}$, $\mathbf{z} \in \mathbb{R}^{1\times m}$, $\pi_i > 0\ \forall i$, $c = \dfrac{1}{\sqrt{\sum_i \pi_i}}$, *and* $\boldsymbol{M} \in \mathbb{R}^{m\times m}$ *such that* $\boldsymbol{M}\boldsymbol{M}^T = \boldsymbol{M}$. *Let* $\mathbf{z}^*$ *denote the minimizer of* $h(\mathbf{z})$. *Then,* $\mathbf{z}^*$ *is given by*

$$\Big(\sqrt{\sum_i \pi_i}\Big)\boldsymbol{M}\mathbf{z}^{*T} = \boldsymbol{M}\Big(\sum_i \pi_i \mathbf{a}_i^T\Big).$$

Proof. We can express $h(\mathbf{z})$ as follows:

$$h(\mathbf{z}) = \sum_i \pi_i \big(\mathbf{a}_i\mathbf{a}_i^T - 2c\mathbf{z}\boldsymbol{M}\mathbf{a}_i^T + c^2\mathbf{z}\boldsymbol{M}\boldsymbol{M}^T\mathbf{z}^T\big),$$

and the gradient is given by

$$\frac{\partial h(\mathbf{z})}{\partial \mathbf{z}} = \sum_i \pi_i \big(- 2c\boldsymbol{M}\mathbf{a}_i^T + 2c^2\boldsymbol{M}\boldsymbol{M}^T\mathbf{z}^T\big).$$

By setting the gradient to zero, we get

$$\begin{aligned}\sum_i \pi_i \boldsymbol{M}\mathbf{a}_i^T &= c\Big(\sum_i \pi_i\Big)\boldsymbol{M}\boldsymbol{M}^T\mathbf{z}^{*T} \\ &= c\Big(\sum_i \pi_i\Big)\boldsymbol{M}\mathbf{z}^{*T} \quad \text{since } \boldsymbol{M}\boldsymbol{M}^T = \boldsymbol{M}\end{aligned}$$

By setting $c = \dfrac{1}{\sqrt{\sum_i \pi_i}}$, we get

$$\Big(\sqrt{\sum_i \pi_i}\Big)\boldsymbol{M}\mathbf{z}^{*T} = \boldsymbol{M}\Big(\sum_i \pi_i \mathbf{a}_i^T\Big).$$

□

Now, Theorem 4 shows the convergence of Algorithm 7.

Theorem 4. *Algorithm 7 monotonically decreases the NEO-Co-Clustering objective (5.4).*

Proof. Let $J^{(t)}$ denote the NEO-Co-Clustering objective (5.4) at t-th iteration. Let $\boldsymbol{U}$ denote the assignment matrix of the current row clustering $\mathcal{C}$, and $\boldsymbol{U}^*$ denote assignment matrix of the updated row clustering $\mathcal{C}^*$ obtained by line 20 in Algorithm 7.

$$\begin{aligned}
J^{(t)} &= \sum_{i=1}^{k}\sum_{j=1}^{l} \left\| D(\mathbf{u}_i)\boldsymbol{X}D(\mathbf{v}_j) - \hat{\mathbf{u}}_i\hat{\mathbf{u}}_i^T\boldsymbol{X}\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T \right\|_F^2 \\
&= \sum_{i=1}^{k}\sum_{j=1}^{l}\sum_{\mathbf{x}_p\in\mathcal{C}_i} \left\| (\boldsymbol{I}_{p\cdot})\boldsymbol{X}D(\mathbf{v}_j) - \frac{1}{\sqrt{\|\mathbf{u}_i\|_1}}\hat{\mathbf{u}}_i^T\boldsymbol{X}\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T \right\|_2^2 \\
&\geq \sum_{i=1}^{k}\sum_{j=1}^{l}\sum_{\mathbf{x}_p\in\mathcal{C}_i^*} \left\| (\boldsymbol{I}_{p\cdot})\boldsymbol{X}D(\mathbf{v}_j) - \frac{1}{\sqrt{\|\mathbf{u}_i\|_1}}\hat{\mathbf{u}}_i^T\boldsymbol{X}\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T \right\|_2^2 \\
&\geq \sum_{i=1}^{k}\sum_{j=1}^{l}\sum_{\mathbf{x}_p\in\mathcal{C}_i^*} \left\| (\boldsymbol{I}_{p\cdot})\boldsymbol{X}D(\mathbf{v}_j) - \frac{1}{\sqrt{\|\mathbf{u}^*{}_i\|_1}}\hat{\mathbf{u}^*}_i^T\boldsymbol{X}\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T \right\|_2^2 \\
&= \sum_{i=1}^{k}\sum_{j=1}^{l} \left\| D(\mathbf{u}^*{}_i)\boldsymbol{X}D(\mathbf{v}_j) - \hat{\mathbf{u}^*}_i\hat{\mathbf{u}^*}_i^T\boldsymbol{X}\hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T \right\|_F^2 \\
&\geq \sum_{i=1}^{k}\sum_{j=1}^{l} \left\| D(\mathbf{u}^*{}_i)\boldsymbol{X}D(\mathbf{v}^*{}_j) - \hat{\mathbf{u}^*}_i\hat{\mathbf{u}^*}_i^T\boldsymbol{X}\hat{\mathbf{v}^*}_j\hat{\mathbf{v}^*}_j^T \right\|_F^2 \\
&= J^{(t+1)}
\end{aligned}$$

$$\boldsymbol{U} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \Rightarrow$$

(a) Initial $\boldsymbol{U}$, $\boldsymbol{V}$ (obj: 0.159242)

$$\boldsymbol{U} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \Rightarrow$$

(b) obj: 0.018142

$$\boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow$$

(c) obj: 0.013596

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \Rightarrow$$

(d) obj: 0.010233

$$\boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix} \Rightarrow$$

(e) obj: 0.010233

$$\boldsymbol{U} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 1 \\ 1 & 1 \\ 0 & 1 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{V} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{bmatrix}$$

(f) Final $\boldsymbol{U}$, $\boldsymbol{V}$ (obj: 0.010233)

Figure 5.3: The progress of the NEO-Co-Clustering algorithm on the small $\boldsymbol{X}$ presented in (5.5) of Chapter 5.2.

The first inequality holds because we make assignments by line 12 & line 15, and the second inequality holds by Lemma 1 with $\mathbf{a}_i^T = (\boldsymbol{I}_{p\cdot})\boldsymbol{X}D(\mathbf{v}_j)$, $\boldsymbol{M} = \hat{\mathbf{v}}_j\hat{\mathbf{v}}_j^T$, $\mathbf{z}^* = \hat{\mathbf{u}^*}_i^T\boldsymbol{X}$, and $\sqrt{\sum_i \pi_i} = \sqrt{\|\mathbf{u}^*{}_i\|_1}$. The last inequality indicates that we can similarly show the decrease from $\boldsymbol{V}$ to $\boldsymbol{V}^*$. □

5.3.2 Illustrative Example

We now provide an illustrative example to show how Algorithm 7 proceeds on the small $\boldsymbol{X}$ presented in (5.5) of Chapter 5.2. In Figure 5.3, the assignment matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ are initialized with arbitrary matrices as shown in Figure 5.3(a). In Figure 5.3(b)–(e), $\boldsymbol{U}$ and $\boldsymbol{V}$ are updated alternatively, and during this process, the NEO-Co-Clustering objective function monotonically decreases. The final output of the NEO-CC algorithm is shown in Figure 5.3(f). Notice that the final assignment matrices produced by the NEO-CC algorithm are the optimal assignment matrices for $\boldsymbol{X}$ as we described in Chapter 5.2.

5.4 Related Work

There are various approaches to solve the co-clustering problem. Among these, we focus on the MSSR [28] formulations because the MSSR objective function has a close relationship with the standard k-means objective, which enables us to find a seamless extension of our NEO-K-Means objective.

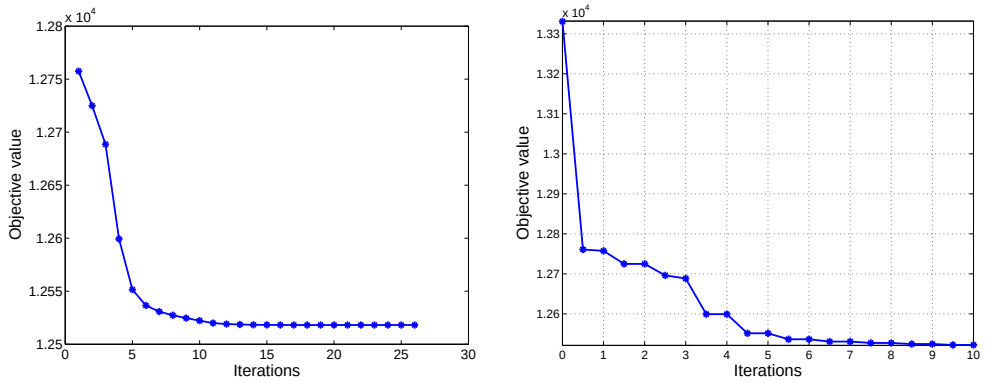
Information-theoretic co-clustering [33] is one of the most well-known co-clustering algorithms. In the information-theoretic approach, a contingency table is viewed as a joint probability distribution between two discrete random variables, and the co-clustering problem is formed as an optimization problem of maximizing the mutual information between the clustered random variables. Also, [10] provides a more general and flexible co-clustering framework where the MSSR [28] and the information-theoretic co-clustering [33] can be viewed as special cases of the framework.

In [31], a Robust Overlapping Co-Clustering (ROCC) has been proposed which is able to find arbitrarily positioned, overlapping co-clusters from noisy datasets. The ROCC algorithm is also intended to find non-exhaustive, overlapping co-clustering. However, the ROCC algorithm includes complicated heuristics while our NEO-CC algorithm is a more principled method. We also notice that [9] proposes an overlapping co-clustering algorithm.

Co-clustering has also been studied in the field of bioinformatics or computational biology since it has been known that a co-clustering technique is useful for finding meaningful structures in gene expression data [70], [27].

5.5 Experimental Results

We show some experimental results on real-world datasets. The first dataset is the YEAST dataset from [35]. This dataset contains micro-array expression data



(a) The NEO-Co-Clustering objective value vs. Iterations

(b) The NEO-Co-Clustering objective values during the first 10 iterations showing both row and column clustering updates

Figure 5.4: The NEO-CC algorithm monotonically decreases the NEO-Co-Clustering objective values.

for genes. Each gene belongs to multiple functional classes, and we treat each functional class as a ground-truth cluster. There are 2,417 genes, 103 features, and 14 functional classes.

In Figure 5.4, we show the change of the NEO-Co-Clustering objective function values as the NEO-CC algorithm progresses. In the plots, the x-axis represents the number of iterations, and the y-axis is the objective function value. We see that the NEO-CC algorithm monotonically decreases the NEO-Co-Clustering objective function values. Figure 5.4(a) shows the change of the objective values until the NEO-CC algorithm terminates on the YEAST dataset while Figure 5.4(b) shows the first 10 iterations. In Figure 5.4(b), we show the objective value after the update of row clustering and the update of column clustering. We see that both row and column clustering updates lead to a decrease in the objective value.

Now, we compare the clustering performance by comparing the algorithmic clusters and the ground-truth clusters. We compare with three different methods: MSSR algorithm [28], the iterative NEO-K-Means algorithm presented in Chapter 3 (denoted by NEO-iter), the low-rank SDP NEO-K-Means presented in Chapter 4

Table 5.1: F_1 scores on YEAST dataset.

	MSSR	NEO-iter	NEO-lrsdp	NEO-CC
Trial 1	0.171	0.359	0.390	**0.406**
Trial 2	0.170	0.366	0.391	**0.406**
Trial 3	0.165	0.360	0.390	**0.406**
Trial 4	0.164	0.360	**0.391**	0.362
Trial 5	0.174	0.356	0.391	**0.407**

Table 5.2: MovieLens datasets

	No. of users	No. of movies	Genres
ML1	718	2,902	Action & Romance
ML2	718	1,913	Mystery & Thriller
ML3	718	1,569	Adventure & Sci-Fi

(denoted by NEO-lrsdp). In our NEO-CC algorithm, we initialize $\boldsymbol{U}$ using the iterative NEO-K-Means, and initialize $\boldsymbol{V}$ by running the iterative NEO-K-Means on the transpose of the original data matrix. The parameters α and β (and α_r, β_r, α_c, and β_c) are estimated using the strategies presented in Chapter 3.

We repeated the experiments for 5 times, and Table 5.1 shows the F_1 scores on YEAST dataset. When we compare the F_1 scores of NEO-iter and NEO-CC, we see that NEO-CC noticeably improves clustering performance. It is interesting to see that the performance of NEO-CC is even slightly better than NEO-lrsdp. Note that NEO-lrsdp involves much more expensive operations than NEO-CC which is a simple iterative algorithm. We also note that the performance of MSSR algorithm is not good. This is because the dataset contains a large overlap among the clusters while the MSSR algorithm only produces disjoint clusters.

The second dataset is a set of user-movie ratings in MovieLens from [2]. In this dataset, the data matrix is represented by users by movies, and non-zero entries represent ratings. In this dataset, the movie genres are treated as the ground-truth clusters. Table 5.2 shows the three different datasets we use. In Table 5.3, we show the best, the worst, and the average F_1 scores for 5 runs. We see that the NEO-CC

Table 5.3: F_1 scores on MovieLens datasets.

		MSSR	NEO-iter	NEO-lrsdp	NEO-CC
ML1	best	0.438	0.563	0.564	**0.581**
	worst	0.438	0.563	0.564	**0.581**
	avg.	0.438	0.563	0.564	**0.581**
ML2	best	0.506	0.568	0.568	**0.588**
	worst	0.502	0.568	0.568	**0.581**
	avg.	0.505	0.568	0.568	**0.584**
ML3	best	0.474	0.591	0.593	**0.598**
	worst	0.471	0.570	0.568	**0.584**
	avg.	0.473	0.574	0.573	**0.588**

algorithm shows the highest F_1 scores.

5.6 Future Work

We plan to extend our NEO-Co-Clustering objective and the NEO-CC algorithm to clustering with other Bregman divergences [12] so that we can extend our methods to Bregman co-clustering [10]. Also, we intend to investigate a low-rank semidefinite programming for the NEO-Co-Clustering problem.

Chapter 6

Design of Parallel Data-driven Algorithms: Case Study with Scalable Data-driven PageRank

We study the design of parallel data-driven algorithms, which enables us to further increase the scalability of our overlapping community detection algorithms. Large-scale graph analysis has received considerable attention in both the machine learning and parallel programming communities. In machine learning, many different types of task-specific algorithms have been developed to deal with massive networks. In parallel computing, many different parallel programming models and systems have been proposed for both shared memory and distributed memory settings to ease implementation and manage parallel programs.

Recent research has observed that distributed graph analytics can have a significant slowdown over shared-memory implementations, that is, the increase in communication costs are not easily made up for by increase in aggregate processing

The materials presented in this chapter have been published in [81]. Joyce designed and implemented the algorithms, and also conducted experiments. Andrew implemented the algorithms. Professor Dhillon and Professor Pingali supervised the work.

power or memory bandwidth. Furthermore, a remarkable number of "large" graphs fit in the main memory of a shared memory machine; it is easy to fit graphs with tens of billions of edges on a large workstation-class machine. Given these factors, it is worth understanding how to efficiently parallelize graph analytics on shared-memory machines. A better understanding of how to implement fast shared-memory analytics both greatly reduces the costs and enables richer applications on commodity systems. Better implementation strategies also help distributed implementations, as they tend to use shared-memory abstractions within a host.

Many graph mining techniques usually involve iterative algorithms where local computations are repeatedly done at a set of nodes until a convergence criterion is satisfied. Let us define *active nodes* to be a set of nodes where computations should be performed. Based on how the active nodes are processed, we can broadly classify these iterative graph algorithms from three different points of view: work activation, data access pattern, and scheduling. In this work [81], we present general approaches for designing scalable data-driven graph algorithms using a case study of the PageRank algorithm. In particular, using the three different algorithm design axes (i.e., work activation, data access pattern, and scheduling), we present eight different formulations and in-memory parallel implementations of PageRank algorithm. We show that by considering data-driven formulations, we can have more flexibility in processing the active nodes, which enables us to develop work-efficient algorithms. We focus our analysis on PageRank in this chapter, but our approaches and formulations can be easily extended to other graph mining algorithms.

6.1 Work Activation

We first classify algorithms into two groups based on work activation: topology-driven and data-driven algorithms. In a topology-driven algorithm, active nodes are defined solely by the structure of a graph. For example, an algorithm which re-

quires processing all the nodes at each iteration is referred to as a topology-driven algorithm. On the other hand, in a data-driven algorithm, the nodes are dynamically activated by their neighbors, i.e., the nodes become active or inactive in an unpredictable way. In many applications, data-driven algorithms can be more work-efficient than topology-driven algorithms because the former allow us to concentrate more on "hot spots" in a graph where more frequent updates are needed.

6.1.1 Topology-driven PageRank

To explain the concepts in more detail, we now focus our discussion on PageRank which is a key technique in web mining [20]. Given a graph $G = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$, let $\mathbf{x}$ denote a PageRank vector of size $|\mathcal{V}|$. Also, let us define $\mathcal{S}_v$ to be the set of incoming neighbors of node v, and $\mathcal{T}_v$ to be the set of outgoing neighbors of node v. Then, node v's PageRank, denoted by x_v, is iteratively computed by

$$x_v^{(k+1)} = \alpha \sum_{w \in \mathcal{S}_v} \frac{x_w^{(k)}}{|\mathcal{T}_w|} + (1 - \alpha)$$

where $x_v^{(k)}$ denotes the k-th iterate, and α is a teleportation parameter ($0 < \alpha < 1$). Algorithm 6.1 presents this iteration, which is the traditional Power method that can be used to compute PageRank. Given a user defined tolerance ϵ, the PageRank vector $\mathbf{x}$ is initialized to be $\mathbf{x} = (1 - \alpha)\mathbf{e}$ where $\mathbf{e}$ denotes the vector of all 1's. The PageRank values are repeatedly computed until the difference between $x_v^{(k)}$ and $x_v^{(k+1)}$ is smaller than ϵ for all the nodes. Since the Power method requires processing all the nodes at each round, it is a topology-driven algorithm.

Algorithm 6.1: Topology-driven PageRank

Input: graph $G = (\mathcal{V}, \mathcal{E})$, α, ϵ
Output: PageRank $\mathbf{x}$

Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}$
while true **do**
  for $v \in \mathcal{V}$ **do**
    $x_v^{(k+1)} = \alpha \sum_{w \in \mathcal{S}_v} \frac{x_w^{(k)}}{|\mathcal{T}_w|} + (1-\alpha)$
    $\delta_v = |x_v^{(k+1)} - x_v^{(k)}|$
  end for
  if $\|\boldsymbol{\delta}\|_\infty < \epsilon$ **then**
    break;
  end if
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

Algorithm 6.2: Data-driven PageRank

Input: graph $G = (\mathcal{V}, \mathcal{E})$, α, ϵ
Output: PageRank $\mathbf{x}$

Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}$
for $v \in \mathcal{V}$ **do**
  `worklist.push(`v`)`
end for
while `!worklist.isEmpty` **do**
  $v =$ `worklist.pop()`
  $x_v^{new} = \alpha \sum_{w \in \mathcal{S}_v} \frac{x_w}{|\mathcal{T}_w|} + (1-\alpha)$
  if $|x_v^{new} - x_v| \geq \epsilon$ **then**
    $x_v = x_v^{new}$
    for $w \in \mathcal{T}_v$ **do**
      if w is not in `worklist` **then**
        `worklist.push(`w`)`
      end if
    end for
  end if
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

6.1.2 Basic Data-driven PageRank

Instead of processing all the nodes in rounds, we can think of an algorithm which dynamically maintains a working set. Algorithm 6.2 shows a basic data-driven PageRank. Initially, the worklist is set to be the entire vertex set. The algorithm proceeds by picking a node from the worklist, computing the node's PageRank, and adding its outgoing neighbors to the worklist.

To examine convergence of the data-driven PageRank, let us rewrite the problem in the form of a linear system. We define a row-stochastic matrix $\boldsymbol{P}$ to be $\boldsymbol{P} \equiv \boldsymbol{D}^{-1}\boldsymbol{A}$ where $\boldsymbol{A}$ is an adjacency matrix and $\boldsymbol{D}$ is the degree diagonal matrix. We assume that there is no self-loop in the graph, i.e., the diagonal elements of $\boldsymbol{A}$ are all zeros. Then, the PageRank computation can be written as follows:

$$\mathbf{x} = \alpha \boldsymbol{P}^T \mathbf{x} + (1-\alpha)\mathbf{e}.$$

This is the linear system of

$$(\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{x} = (1-\alpha)\mathbf{e},$$

and the residual is defined to be

$$\mathbf{r} = (1-\alpha)\mathbf{e} - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{x} = \alpha \boldsymbol{P}^T\mathbf{x} + (1-\alpha)\mathbf{e} - \mathbf{x}.$$

In this setting, it has been shown in [60] that each local computation in Algorithm 6.2 decreases the residual. Indeed, when a node v's PageRank is updated, its residual r_v becomes zero, and it can be shown that $\alpha r_v/|\mathcal{T}_v|$ is added to each of its outgoing neighbors' residuals. Theorem 5 formally describes this.

Theorem 5. *In Algorithm 6.2, when $x_v^{(k)}$ is updated to $x_v^{(k+1)}$, the total residual is decreased at least by $r_v(1-\alpha)$. More specifically,*

$$\mathbf{e}^T\mathbf{r}^{(k+1)} = \begin{cases} \mathbf{e}^T\mathbf{r}^{(k)} - r_v^{(k)}(1-\alpha) & : \mathcal{T}_v \neq \emptyset \\ \mathbf{e}^T\mathbf{r}^{(k)} - r_v^{(k)} & : \mathcal{T}_v = \emptyset \end{cases}$$

Proof. Let $x_v^{(k)}$ denote the k-th update of x_v. In Algorithm 6.2, we initialize $\mathbf{x}$ to be $\mathbf{x}^{(0)} = (1-\alpha)\mathbf{e}$. Thus, the initial residual $\mathbf{r}^{(0)}$ can be written as follows:

$$\mathbf{r}^{(0)} = (1-\alpha)\mathbf{e} - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)(1-\alpha)\mathbf{e} = (1-\alpha)\alpha \boldsymbol{P}^T\mathbf{e} \geq 0. \tag{6.1}$$

We can see that the initial residual is positive. In Algorithm 6.2, for each active node v, we update its PageRank as follows:

$$x_v^{(k+1)} = (1-\alpha) + \alpha \sum_{w \in \mathcal{S}_v} \frac{x_w^{(k)}}{|\mathcal{T}_w|}.$$

Indeed, this is equivalent to:

$$x_v^{(k+1)} = x_v^{(k)} + r_v^{(k)}. \tag{6.2}$$

because:

$$x_v^{(k+1)} = x_v^{(k)} + r_v^{(k)} = x_v^{(k)} + \underbrace{(1-\alpha) - x_v^{(k)} + \alpha[\boldsymbol{P}^T \mathbf{x}^{(k)}]_v}_{r_v^{(k)}}.$$

Also, after such an update, we can show that $\mathbf{r}^{(k+1)} \geq 0$. Let $\gamma = \mathbf{r}_v^{(k)}$. Then,

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \gamma \mathbf{e}_v$$

$$\mathbf{r}^{(k+1)} = (1-\alpha)\mathbf{e} - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{x}^{(k+1)}$$

$$\mathbf{r}^{(k+1)} = (1-\alpha)\mathbf{e} - (\boldsymbol{I} - \alpha \boldsymbol{P}^T)(\mathbf{x}^{(k)} + \gamma \mathbf{e}_v)$$

$$\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \gamma(\boldsymbol{I} - \alpha \boldsymbol{P}^T)\mathbf{e}_v \tag{6.3}$$

Note that the v-th component of $\mathbf{r}^{(k+1)}$ goes to zero, and we only add positive values to the other components. Recall that the initial residual is positive shown in (6.1). Thus, by induction, we can see that $\mathbf{r}^{(k+1)} \geq 0$.

Now, by multiplying $\mathbf{e}^T$ in (6.3), we get:

$$\mathbf{e}^T \mathbf{r}^{(k+1)} = \begin{cases} \mathbf{e}^T \mathbf{r}^{(k)} - r_v^{(k)}(1-\alpha) & : \mathcal{T}_v \neq \emptyset \\ \mathbf{e}^T \mathbf{r}^{(k)} - r_v^{(k)} & : \mathcal{T}_v = \emptyset \end{cases}$$

Thus, any step decreases the residual by at least $\gamma(1-\alpha)$, and moves $\mathbf{x}$ closer to the solution. □

Followed from this, we can also show Theorem 6.

Theorem 6. *Algorithm 6.2 guarantees $\|\mathbf{r}\|_\infty < \epsilon$ when it is converged.*

Proof. Whenever a node's PageRank is updated, the residual of each of its outgoing neighbors is increased. Thus, if we ever change a node's PageRank, we need to add its outgoing neighbors to the worklist to verify that their residual is sufficiently small. This is what Algorithm 6.2 does. □

Thus, we showed that Algorithm 6.2 converges, and on termination, it is guaranteed that the residual $\|\mathbf{r}\|_\infty < \epsilon$. From the next chapter, we will focus on the data-driven formulation of PageRank, and build up various variations.

6.2 Data Access Pattern

Data access pattern (or memory access pattern) is an important factor in designing a scalable graph algorithm. When an active node is processed, there can be a particular data access pattern. For example, some algorithms require reading a value of an active node and updating its outgoing neighbors, whereas some algorithms require reading values from incoming neighbors of an active node and updating the active node's value. Based on these data access patterns, we can classify algorithms into three categories: pull-based, pull-push-based, and push-based algorithms.

6.2.1 Pull-based PageRank

In *pull-based* algorithms, an active node *pulls* (reads) its neighbors' values and updates its own value. Note that pull-based algorithms require more *read* operations than *write* operations in general because the *write* operation is only performed on the active node. In the PageRank example, Algorithms 6.1 and 6.2 are both pull-based algorithms because an active node pulls (reads) its incoming neighbors' PageRank values and updates its own PageRank.

6.2.2 Pull-Push-based PageRank

In *pull-push-based* algorithms, an active node *pulls* (reads) its neighbors' values and also *pushes* (updates) its neighbors' values. When we consider the cost for processing an active node, pull-push-based algorithms might be more expensive than pull-based algorithms as they require both *read* and *write* operations on neighbors. However, in terms of information propagation, pull-push-based algorithms can have advantages because in pull-push-based algorithms, an active node can propagate information to its neighbors whereas in pull-based algorithms, an active node passively receives information from its neighbors.

Now, we transform the basic data-driven PageRank into a pull-push-based algorithm. In Algorithm 6.2, whenever a node's PageRank is updated, the residuals of its outgoing neighbors are increased. Thus, to guarantee that the maximum residual is smaller than ϵ, all the outgoing neighbors of an active node should be added to the worklist. However, if we explicitly compute and maintain the residuals, we do not need to add all the outgoing neighbors of an active node, instead, we only need to add the outgoing neighbors whose residuals become greater than or equal to ϵ. In this way, we can filter out some work in the worklist. In Algorithm 6.3, the initial residual $\mathbf{r}^{(0)}$ is computed by $\mathbf{r}^{(0)} = (1-\alpha)\alpha \boldsymbol{P}^T \mathbf{e}$ (lines 3–8). Each active node pulls its incoming neighbors' PageRank values (line 14), and pushes residuals to its outgoing neighbors (line 17). Then, an outgoing neighbor w of the active node v is added to the worklist only if the updated residual r_w is greater than or equal to ϵ and its old residual is less than ϵ. The second condition allows us to avoid having duplicates in the worklist (i.e., we add a node to the worklist only when its residual crosses ϵ). In this algorithm, there is a trade-off between overhead for residual computations and filtering out work in the worklist. We empirically observe that in many cases, the benefit of filtering overcomes the overhead for residual computations.

Algorithm 6.3: Pull-Push-based PageRank

Input: graph $G = (\mathcal{V}, \mathcal{E})$, α, ϵ
Output: PageRank $\mathbf{x}$

Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}$
Initialize $\mathbf{r} = \mathbf{0}$
for $v \in \mathcal{V}$ **do**
  for $w \in \mathcal{S}_v$ **do**
    $r_v = r_v + \frac{1}{|\mathcal{T}_w|}$
  end for
  $r_v = (1-\alpha)\alpha r_v$
end for
for $v \in \mathcal{V}$ **do**
  `worklist.push(`v`)`
end for
while `!worklist.isEmpty` **do**
  $v =$ `worklist.pop()`
  $x_v = \alpha \sum_{w \in \mathcal{S}_v} \frac{x_w}{|\mathcal{T}_w|} + (1-\alpha)$
  for $w \in \mathcal{T}_v$ **do**
    $r_w^{old} = r_w$
    $r_w = r_w + \frac{r_v \alpha}{|\mathcal{T}_v|}$
    if $r_w \geq \epsilon$ and $r_w^{old} < \epsilon$ **then**
      `worklist.push(`w`)`
    end if
  end for
  $r_v = 0$
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

Algorithm 6.4: Push-based PageRank

Input: graph $G = (\mathcal{V}, \mathcal{E})$, α, ϵ
Output: PageRank $\mathbf{x}$

Initialize $\mathbf{x} = (1-\alpha)\mathbf{e}$
Initialize $\mathbf{r} = \mathbf{0}$
for $v \in \mathcal{V}$ **do**
  for $w \in \mathcal{S}_v$ **do**
    $r_v = r_v + \frac{1}{|\mathcal{T}_w|}$
  end for
  $r_v = (1-\alpha)\alpha r_v$
end for
for $v \in \mathcal{V}$ **do**
  `worklist.push(`v`)`
end for
while `!worklist.isEmpty` **do**
  $v =$ `worklist.pop()`
  $x_v^{new} = x_v + r_v$
  for $w \in \mathcal{T}_v$ **do**
    $r_w^{old} = r_w$
    $r_w = r_w + \frac{r_v \alpha}{|\mathcal{T}_v|}$
    if $r_w \geq \epsilon$ and $r_w^{old} < \epsilon$ **then**
      `worklist.push(`w`)`
    end if
  end for
  $r_v = 0$
end while
$\mathbf{x} = \frac{\mathbf{x}}{\|\mathbf{x}\|_1}$

6.2.3 Push-based PageRank

In *push-based* algorithms, an active node updates its own value, and only *pushes* (updates) its neighbors' values. Compared to pull-based algorithms, push-based algorithms can be more costly in the sense that they require more *write* operations. However, push-based algorithms invoke more frequent updates, which might be helpful to achieve a faster information propagation over the network. Compared to pull-push-based algorithms, push-based algorithms can be more efficient because they only require *write* operations instead of *read* & *write* operations. To design a push-based PageRank, we need to notice that the (k+1)-st PageRank update of node v is equivalent to the sum of the k-th PageRank of v and its k-th residual. This can be derived from the linear system formulation discussed in Chapter 6.1.2. Thus, we can formulate a push-based PageRank as follows: for each active node

v, its PageRank is updated by $x_v^{(k+1)} = x_v^{(k)} + r_v^{(k)}$. Algorithm 6.4 shows the full procedure. Note that the only difference between Algorithm 6.3 and Algorithm 6.4 is line 14. In Algorithm 6.4, an active node updates its own PageRank and the residuals of its outgoing neighbors.

6.3 Scheduling

Task scheduling, the order in which tasks are executed, can be very important to graph algorithms [63]. For example, in data-driven PageRank, we see that whenever a node v has residual r_v, and its PageRank is then updated, the total residual is decreased "at least" or "exactly" by $r_v(1-\alpha)$. This suggests that if we process "large residual" nodes first, the algorithm might converge faster. Thus, we can define a node v's priority p_v to be the residual per unit work, i.e., $p_v = r_v/d_v$ where $d_v = |\mathcal{S}_v| + |\mathcal{T}_v|$ for the pull-push-based PageRank, and $d_v = |\mathcal{T}_v|$ in the push-based algorithm. Realizing the potential benefits in convergence requires priority scheduling. In priority scheduling, each task is assigned a value, the priority, and scheduled in increasing (or decreasing) order. More sophisticated schedulers allow modifying the priority of existing tasks, but this is an expensive operation not commonly supported in parallel systems. Practical priority schedulers have to trade off several factors: efficiency, communication (and thus scaling), priority fidelity, and set-semantics (here, the "set-semantics" means that there are no duplicate work items in the worklist). In general, both priority fidelity and set-semantics require significant global knowledge and communication, thus are not scalable. To investigate the sensitivity of PageRank to different design choices in a priority scheduler, we use two different designs: one which favors priority fidelity but gives up set-semantics and one which preserves set-semantics at the expense of priority fidelity. We compare these with scalable non-priority schedulers to see if the improved convergence outweighs the increased cost of priority scheduling.

The first scheduler we use is the scalable, NUMA-aware OBIM (ordered-by-integer-metric) priority scheduler [52]. This scheduler uses an approximate consensus protocol to inform a per-thread choice to search for stealable high-priority work or to operate on local near-high-priority work. Various underlying data structures and stealing patterns are aware of the machine's memory topology and optimized to maximize information propagation while minimizing cache coherence cost. OBIM favors keeping all threads operating on high priority work and does not support either set-semantics or updating the priority of existing tasks. To handle this, tasks are created for PageRank every time a node's priority changes, potentially generating duplicate tasks in the scheduler. Tasks with outdated priorities are quickly filtered out at execution time (a process which consumes only a few instructions).

The second scheduler we use is a bulk-synchronous priority scheduler. This scheduler operates in rounds. Each round, all items with priority above a threshold are executed. Generated tasks and unexecuted items are placed in the next round. The range and mean are computed for the tasks, allowing the threshold to be chosen for each round based on the distribution of priorities observed for that round. This organization makes priority updates simple; priorities are recomputed every round. Further, set-semantics may be trivially maintained. However, to minimize the overhead of bulk-synchronous execution, each round must have sufficient work to amortize the barrier synchronization. This produces a schedule of tasks which may deviate noticeably from the user requested order.

We also consider FIFO- and LIFO-like schedules (parallel schedulers cannot both scale and preserve exact FIFO and LIFO order). It is obvious that a LIFO scheduler is generally bad for PageRank. Processing nodes after a single neighbor is visited will process the node once for each in-neighbor. FIFO schedulers provide time for a node to accumulate pending changes from many neighbors before being processed. We use a NUMA-aware scheduler, similar to that from Galois and

QThreads, to do scalable, fast FIFO-like scheduling.

6.4 Related Work

Our approaches of considering three different algorithm design axes are mainly motivated by the Tao analysis [69] where the concepts of topology-driven and data-driven algorithms have been studied in the context of amorphous data-parallelism. While Tao analysis has been proposed for a general parallel programming framework, our analysis is geared more towards designing new scalable data mining algorithms.

For scalable parallel computing, many different types of parallel programming models have been proposed, e.g., Galois [62], Ligra [74], GraphLab [56], Priter [87], and Maiter [88]. Since PageRank is a popular benchmark for parallel programming models, various versions of PageRank have been implemented in different parallel platforms in a rather ad hoc manner. Also, in data mining communities, PageRank has been extensively studied, and many different approximate algorithms (e.g., [7], [45]) have been developed over the years [16]. The Gauss–Seidel style update of PageRank is studied in [60], and parallel distributed PageRank also has been developed [38]. Our PageRank formulations can be considered as variations of these previous studies. Our contribution in this work is to systematically analyze and discuss various PageRank implementations with the perspective of designing scalable graph mining methodologies.

Even though we have focused our discussion on PageRank in this chapter, our approaches can be easily extended to other data mining algorithms. For example, in semi-supervised learning, label propagation is a well-known method [15] which involves fairly similar computations as PageRank. We expect that our data-driven formulations can be applied to the label propagation method. Also, as we mentioned in Chapter 2, there is a close relationship between personalized PageRank and community detection [7]. So, parallel data-driven community detection can be

Table 6.1: Summary of algorithm design choices

Algorithm	Activation	Access	Schedule
dd-push	Data-driven	Push	FIFOs w/ Stealing
dd-push-prs	Data-driven	Push	Bulk-sync Priority
dd-push-prt	Data-driven	Push	Async Priority
dd-pp-rsd	Data-driven	Pull-Push	FIFOs w/ Stealing
dd-pp-prs	Data-driven	Pull-Push	Bulk-sync Priority
dd-pp-prt	Data-driven	Pull-Push	Async Priority
dd-basic	Data-driven	Pull	FIFOs w/ Stealing
power-iter	Topology	Pull	Load Balancer

another interesting application of our analysis.

6.5 Experimental Results

6.5.1 Experimental Setup

To see the performance and scaling sensitivity of PageRank to the design considerations in this chapter, we implement a variety of PageRank algorithms with different scheduling and data access patterns. All implementations are written using the Galois System [62]. Table 6.1 summarizes the design choices for each implementation. Pseudo-code and more detailed discussions of each appear in previous chapters. We also compare our results to a third-party baseline, namely GraphLab, varying such parameters as are available in that implementation. For all experiments, we use $\alpha = 0.85$, $\epsilon = 0.01$. We use a 4 socket Xeon E7-4860 running at 2.27GHz with 10 cores per socket and 128GB RAM. GraphLab was run in multi-threaded mode.

6.5.2 Datasets

We use four real-world networks, given in Table 6.2. Twitter and Friendster are social networks, and pld and sd1 are hyperlink graphs. These graphs range from about 600 million edges to 3.6 billion edges. These range in size for in-memory

Table 6.2: Input Graphs

	# nodes	# edges	CSR size	source
pld	39M	623M	2.7G	webdatacommons.org/hyperlinkgraph/
sd1	83M	1,937M	7.9G	webdatacommons.org/hyperlinkgraph/
Twitter	51M	3,228M	13G	twitter.mpi-sws.org/
Friendster	67M	3,623M	14G	archive.org/details/friendster-dataset-201107

compressed sparse row representations from 2.7GB to 14GB for the directed graph. Most of the algorithms require tracking both in-edges and out-edges, making the effective in-memory size approximately twice as large.

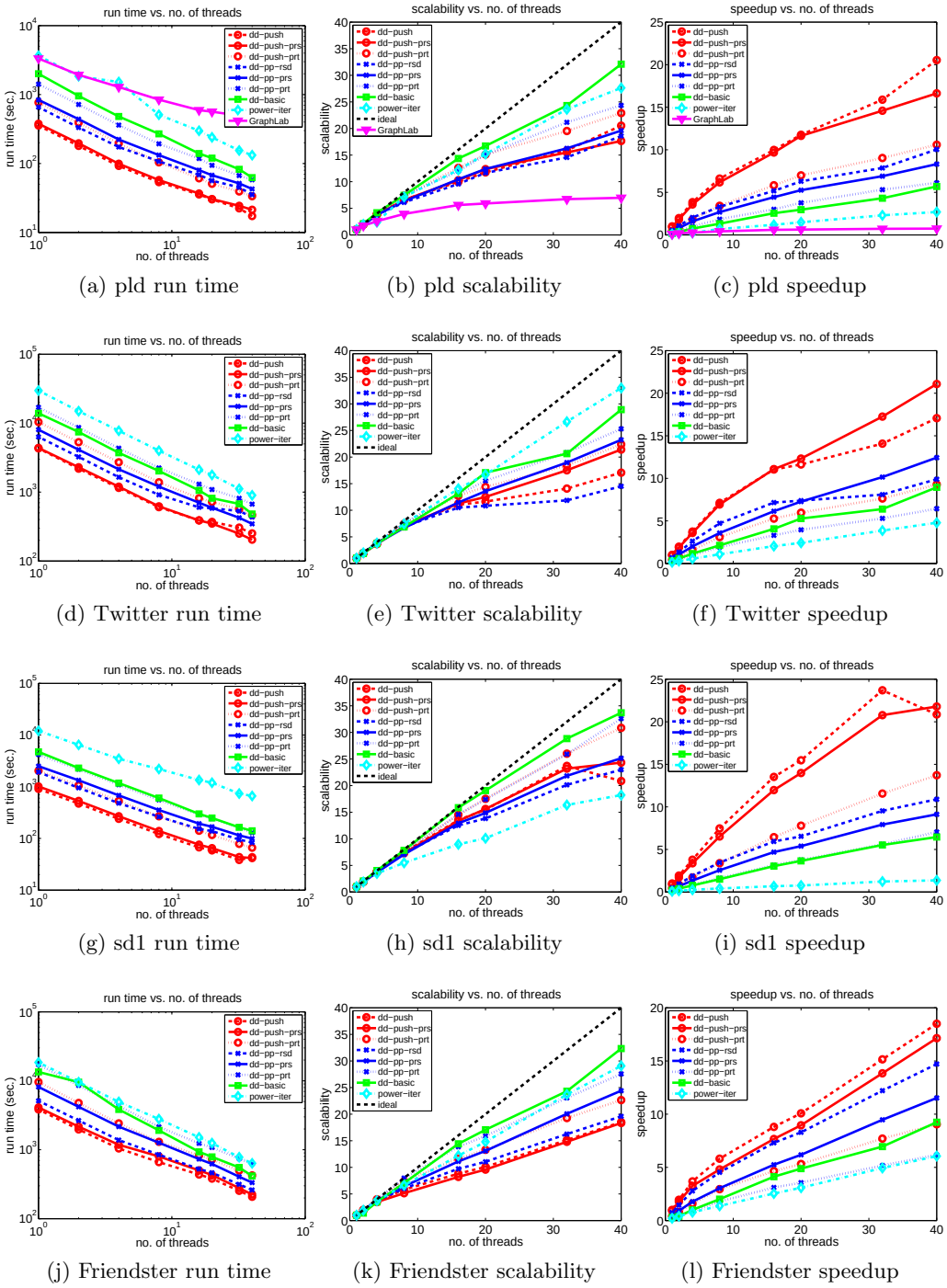
6.5.3 Results

Figure 6.1 shows run time, self-relative scalability, and speedup against the best single-threaded algorithm. Specifically, for each input, we present:

- Run time: run time of each method as a function of threads
- Scalability: $\frac{\text{run time of method } m \text{ with a single thread}}{\text{run time of method } m \text{ with } t \text{ threads}}$
- Speedup: $\frac{\text{run time of the fastest single-thread method}}{\text{run time of method } m \text{ with } t \text{ threads}}$.

We note that GraphLab ran out of memory for all but the smallest (pld) input. On pld, the serial GraphLab performance was approximately the same as the closest Galois implementation, power-iter, but GraphLab scaled significantly worse. Several broad patterns can be seen in the results. First, all data-driven implementations outperform topology implementation. The best data-driven PageRank implementation is 28x faster than GraphLab, and 10-20x faster than Galois power-iter, depending on the thread count. Second, push-only implementations outperform pull-push implementations which outperform a pure pull-based version. Finally, priority-scheduled versions scale better but perform worse than a fast, non-priority scheduler.

One surprising result is that pulling to compute PageRank and pushing residuals outperforms a pure pull-based version (dd-pp-* vs. dd-basic). The read-mostly



(a) pld run time (b) pld scalability (c) pld speedup

(d) Twitter run time (e) Twitter scalability (f) Twitter speedup

(g) sd1 run time (h) sd1 scalability (i) sd1 speedup

(j) Friendster run time (k) Friendster scalability (l) Friendster speedup

Figure 6.1: Run time, scalability and speedup. Our data-driven, push-based PageRank achieves the best speedup.

Table 6.3: The number of completed tasks (unit: 10^6)

	pld		sd1		twitter		friendster	
threads	1	40	1	40	1	40	1	40
dd-push	134	133	282	273	393	417	476	581
dd-push-prs	330	319	758	740	888	850	1076	1069
dd-push-prt	246	244	538	535	395	418	504	619
dd-pp-rsd	131	130	279	271	386	410	473	540
dd-pp-prs	311	303	712	716	963	835	1239	1212
dd-pp-prt	138	136	289	286	394	419	489	611
dd-basic	655	536	1029	896	1629	1526	1482	1356
power-iter	2606	2606	6716	6716	4297	4297	3104	3104

nature of pull-based algorithms are generally more cache friendly. Push-based algorithms have a much larger write-set per iteration, and writes to common locations fundamentally do not scale. The extra cost of the pushes, however, is made up by a reduction in the number of tasks. Table 6.3 shows the number of completed tasks for each algorithm, and we see that pull-push methods (dd-pp-rsd) lead to 70-80% reduction in the number of tasks executed (compared to dd-basic). The pushing of residual allows a node to selectively activate a neighbor, and thus greatly reduces the total work performed (effectively, PageRanks are only computed when they are needed). On the other hand, the basic pull algorithm must unconditionally generate tasks for each of a node's neighbors when the node is updated. It is more understandable that the push-only version outperforms all others. The pushing of residual is equivalent to the computation of PageRank deltas; thus, the pull can be eliminated without any extra cost. This both reduces the number of edges inspected for every node (from in and out to just out) and reduces the total computation (instructions). Serially, a deterministic scheduler processes the same nodes, thus it does not save on the total number of tasks, as can be seen in the Table 6.3 rows for dd-push and dd-pp-rsd. The variation in those rows is due to the variation in scheduling order (especially at higher thread counts) though the variation is relatively minor.

In Table 6.3, all reported numbers include all tasks (nodes) considered to make scheduling decisions. For *-prt methods, this includes the nodes which are duplicates in the worklist. For *-prs methods, this includes each round's examination of all the nodes in the worklist to pick the priority threshold. Priority scheduling favoring priority order, denoted *-prt, shows the high cost of duplicate items in the worklist. This priority scheduler must insert duplicate tasks every time a node moves to a new priority bin. This means that many tasks are useless; they discover as their first action that there is nothing to do and complete. Figure 6.1 shows that this has a distinct time cost. Although filtering out duplicates is not expensive, the total work for filtering is significant. Priority scheduling favoring set semantics, denoted *-prs, also must examine a significant number of nodes to determine which tasks to pick at each scheduling round. We observe that the total number of nodes in the worklist decreases rapidly, making the working set after several rounds significantly smaller than the entire graph. This boost in locality helps offset the extra data accesses.

It is interesting to see that optimizing for cache behavior (pull-based) may not always be as effective as optimizing for pushing maximum information quickly (push-based). The push-only PageRank (dd-push-*) is entirely read-write access, while the pull-only version (dd-basic) does one write per node processed. In general, read-mostly access patterns are significantly more cache and coherence friendly. From this perspective, the pull-push versions, denoted dd-pp-*, should be the worst as they have the read set of the pull versions and the write set of the push versions. The extra writes are not just an alternate implementation of the PageRank update, but rather influence the scheduling of tasks. The extra writes weigh nodes, allowing nodes to only be processed when profitable. This improved scheduling makes up for the increased write load. Given the scheduling benefits of the residual push, it is easy to see that the push-only version is superior to the pull-push version as it

Table 6.4: Run time of different PageRank implementations on the pld dataset

System	Method	Threads				
		40	32	16	8	1
GraphLab	sync	478 secs.	496 secs.	594 secs.	845 secs.	3,332 secs.
	async-fifo	500 secs.	580 secs.	618 secs.	898 secs.	5,194 secs.
	async-qfifo	788 secs.	804 secs.	970 secs.	1,292 secs.	5,098 secs.
	async-sweep	4,186 secs.	5,162 secs.	9,156 secs.	$>$ 4 hrs.	$>$ 4 hrs.
	async-prt	$>$ 4 hrs.	$>$ 4 hrs.	$>$ 4 hrs.	$>$ 4 hrs.	$>$ 4 hrs.
Galois	power-iter	132 secs.	155 secs.	299 secs.	510 secs.	3,650 secs.
	dd-basic	62 secs.	82 secs.	140 secs.	269 secs.	2,004 secs.
	dd-pp-prt	58 secs.	67 secs.	118 secs.	193 secs.	1,415 secs.
	dd-push	17 secs.	22 secs.	36 secs.	53 secs.	355 secs.

reduces the memory load and work per iteration. We do note that when looking at the self-relative scalability of the implementations, the read-mostly algorithms, while slower, have better scalability than the push and pull-push variants.

Table 6.4 shows a comparison between our data-driven PageRank algorithms (implemented using Galois) and GraphLab's PageRank implementations when varying the scheduling on the pld dataset. GraphLab supports different schedulers, but we find the simple synchronous one to be the best. We note that the GraphLab's asynchronous method refers to a Gauss–Seidel style solver, which still is a bulk-synchronous, topology-driven approach. The power-iter version (in Galois) is actually a classic synchronous implementation in this sense, but still notably faster. While GraphLab's topology-driven synchronous implementation has similar single threaded performance to the Galois topology-driven synchronous implementation, power-iter scales much better than GraphLab. Also, all the data-driven implementations (dd-*) are much faster than GraphLab's PageRank implementations. We see that with 40 threads the fastest GraphLab's method takes 478 seconds whereas our push-based PageRank takes 17 seconds.

6.6 Discussion

Priority scheduling needs some algorithmic margin to be competitive as it is more costly. While it is not surprising that priority scheduling is slower than simple scalable scheduling, this has some important consequences. First, the benefit is dependent on both algorithmic factors and input characteristics. When scheduling changes the asymptotic complexity of an algorithm, there can be huge margins available. In PageRank, there is a theoretical margin available, but it is relatively small. This limits the extra computation that can be spent on scheduling overhead without hurting performance. Second, the margin available depends on input characteristics. For many analytic algorithms, scheduling increases in importance as the diameter of the graph increases. Since PageRank is often run on power-law style graphs with low diameter, we expect a small margin available from priority scheduling.

Good priority schedulers can scale competitively with general purpose schedulers. We observe that multiple priority scheduler implementations scale well. We implement two very different styles of priority schedulers which pick different points in the design and feature space. This is encouraging as it leads us to believe that such richer semantic building blocks can be used by algorithm designers. PageRank updates priorities often—a use case which is hard to support efficiently and scalably. Even many high-performance, serial priority queues do not support this operation. Constructing a concurrent, scalable priority scheduler which maintains set semantics by adjusting priorities for existing items in the scheduler is an open question. The reason is simply one of global knowledge. Knowing whether to insert an item or whether it is already scheduled and thus only needs its priority adjusted requires global knowledge of the system. Maintaining and updating global knowledge concurrently in a NUMA system is rarely scalable. For scalability, practical implementations will contain multiple queues, meaning that not only does one need

to track whether a task is scheduled, but on which queue the task is scheduled. The scheduler we produced for *-prs stores set semantics information by marking nodes in the graph and periodically rechecks priority. This essentially introduces latency between updating a priority and having the scheduler see the new priority. The amount of latency depends on how many iterations proceed before rechecking. This number determines the overhead of the scheduler.

6.7 Future Work

Although PageRank is a simple graph analytic algorithm, there are many interesting implementation details one needs to consider to achieve a high-performance implementation. We show that data-driven implementations are significantly faster than traditional power iteration methods. PageRank has a simple vertex update equation. However, this update can be mapped to the graph in several ways by changing how and when information flows through the graph. These algorithmic changes can significantly affect the performance. Within this space, one can also profitably consider the order in which updates occur to maximize convergence speed. While we investigate these implementation variants for PageRank—seeing performance improvements of 28x over standard power iterations—these considerations can apply to many other convergence-based graph analytic algorithms.

Based on the lessons learned from the case study with scalable data-driven PageRank, we expect that we can develop parallel data-driven algorithms for the methods we presented in the previous chapters. We plan to develop parallel data-driven community detection and clustering algorithms.

Chapter 7

Conclusions

In this thesis, we have proposed scalable overlapping community detection and clustering algorithms that effectively identify high quality overlapping communities and clusters in various real-world datasets.

We started our discussion with a personalized PageRank-based overlapping community detection method, which we call NISE. The key idea of NISE is to find good seeds and then greedily expand these seeds using a personalized PageRank clustering scheme. By developing NISE, we suggest new ideas in the prototypical "seed-and-grow" meta-algorithm for overlapping communities. Experimental results show that NISE significantly outperforms other state-of-the-art overlapping community detection methods in terms of run time, cohesiveness of communities, and ground-truth accuracy on large-scale real-world networks.

By exploiting the connection between community detection and clustering, we were able to develop a more principled algorithm. We present a novel extension of the k-means formulation that simultaneously considers non-exhaustive and overlapping clustering called NEO-K-Means. The underlying objective and algorithm seamlessly generalize the classic k-means approach and enable a new class of applications. When we evaluate this new method on synthetic and real-world data,

it shows the best performance in terms of finding the ground-truth clusters among a large class of state-of-the-art methods. We show that a weighted kernel k-means variation of NEO-K-Means provides a principled way to find a set of overlapping communities in large-scale networks. We conclude that NEO-K-Means is a useful algorithm to analyze much of the complex data emerging in current data-centric applications.

We further improved the performance of NEO-K-Means by studying a semidefinite programming technique. The iterative NEO-K-Means algorithm is fast but suffers from the classic problem that iterative algorithms for k-means fall into local minimizers given poor initialization. To get a more reliable and accurate solution, we study a convex relaxation of the NEO-K-Means objective and also formulate a low-rank SDP for NEO-K-Means. Our new convex and low-rank objective functions provide a new, principled framework to cluster vector and graph data. When our non-convex low-rank method is optimized through an augmented Lagrangian method, it produces state-of-the-art quality results for both data clustering as well as for overlapping community detection.

We extended our NEO-K-Means ideas to co-clustering, which enables us to identify overlapping communities in bipartite graphs. We formulate the non-exhaustive, overlapping co-clustering problem and propose the NEO-Co-Clustering objective function. To optimize the objective, we develop a simple iterative algorithm which we call NEO-CC algorithm. We prove that the NEO-CC algorithm monotonically decreases the NEO-Co-Clustering objective function. Experimental results show that our NEO-CC algorithm is able to effectively capture the underlying co-clustering structure of the data and discover high quality overlapping communities on real-world bipartite graphs.

Finally, we studied the design of parallel data-driven algorithms, which enables us to further increase the scalability of our overlapping community detection

algorithms. Using PageRank as a model problem, we investigate the impact of different algorithm design choices. The design choices affect both single-threaded performance as well as parallel scalability. The implementation lessons not only guide efficient implementations of many graph mining algorithms but also provide a framework for designing new scalable algorithms, especially for large-scale community detection.

Bibliography

[1] A Java Library for Multi-Label Learning. `http://mulan.sourceforge.net/datasets.html`.

[2] GroupLens Research. `http://grouplens.org/datasets/movielens/`.

[3] Rat brains are basically wired up like miniature internets. `www.engadget.com/`.

[4] Stanford Network Analysis Project. `http://snap.stanford.edu/`.

[5] B. Abrahao, S. Soundarajan, J. Hopcroft, and R. Kleinberg. On the separability of structural classes of communities. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data mining*, pages 624–632, 2012.

[6] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466:761–764, 2010.

[7] R. Andersen, F. Chung, and K. Lang. Local graph partitioning using PageRank vectors. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 475–486, 2006.

[8] R. Andersen and K. J. Lang. Communities from seed sets. In *Proceedings of the 15th International Conference on World Wide Web*, pages 223–232, 2006.

[9] D. Baier, W. Gaul, and M. Schader. Two-mode overlapping clustering with applications to simultaneous benefit segmentation and market structuring. *Classification and Knowledge Organization*, pages 557–566, 1997.

[10] A. Banerjee, I. S. Dhillon, J. Ghosh, S. Merugu, and D. S. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximations. *Journal of Machine Learning Research*, 8:1919–1986, 2007.

[11] A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu, and R. J. Mooney. Model-based overlapping clustering. In *Proceedings of the 11th ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 532–537, 2005.

[12] A. Banerjee, S. Merugu, I. S. Dhillon, and J. Ghosh. Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[13] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[14] C.-E. ben N'Cir, G. Cleuziou, and N. Essoussi. Identification of non-disjoint clusters with small and parameterizable overlaps. In *Proceedings of the International Conference on Computer Applications Technology*, pages 1–6, 2013.

[15] Y. Bengio, O. Delalleau, and N. Le Roux. *Label Propagation and Quadratic Criterion*. MIT Press, 2006.

[16] P. Berkhin. A survey on PageRank computing. *Internet Mathematics*, 2:73–120, 2005.

[17] J. C. Bezdek, R. Ehrlich, and W. Full. FCM: The fuzzy c-means clustering algorithm. 10(2-3):191–203, 1984.

[18] F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V. Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and Katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

[19] M. R. Boutell, J. Luo, X. Shen, and C. M. Brown. Learning multi-label scene classification. *Pattern Recognition*, 37:1757–1771, 2004.

[20] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[21] S. Burer and R. D. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95:329–357, 2003.

[22] S. Burer and R. D. Monteiro. Local minima and convergence in low-rank semidefinite programming. *Mathematical Programming*, 103(3):427–444, 2005.

[23] R. S. Burt. *Structural Holes: The Social Structure of Competition*. Harvard University Press, 1995.

[24] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[25] S. Chawla and A. Gionis. k-means--: a unified approach to clustering and outlier detection. In *Proceedings of the 13th SIAM International Conference on Data Mining*, pages 189–197, 2013.

[26] Y.-L. Chen and H.-L. Hu. An overlapping cluster algorithm to provide non-exhaustive clustering. 173(3):762–780, 2006.

[27] Y. Cheng and G. Church. Biclustering of expression data. In *Proceedings of*

the 8th International Conference on Intelligent Systems for Molecular Biology, pages 93–103, 2000.

[28] H. Cho, I. S. Dhillon, Y. Guan, and S. Sra. Minimum sum-squared residue co-clustering of gene expression data. In *SIAM International Conference on Data Mining*, pages 114–125, 2004.

[29] G. Cleuziou. An extended version of the k-means method for overlapping clustering. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4, 2008.

[30] M. Coscia, G. Rossetti, F. Giannotti, and D. Pedreschi. Demon: a local-first discovery method for overlapping communities. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data mining*, pages 615–623, 2012.

[31] M. Deodhar, G. Gupta, J. Ghosh, H. Cho, and I. Dhillon. A scalable framework for discovering coherent co-clusters in noisy data. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 241–248, 2009.

[32] I. S. Dhillon, Y. Guan, and B. Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[33] I. S. Dhillon, S. Mallela, and D. S. Modha. Information-theoretic co-clustering. In *Proceedings of the 9th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 89–98, 2003.

[34] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets*. Cambridge University Press, 2010.

[35] A. Elisseeff and J. Weston. A kernel method for multi-labelled classification.

In *Proceedings of the Neural Information Processing Systems Conference*, pages 681–687, 2001.

[36] U. Gargi, W. Lu, V. Mirrokni, and S. Yoon. Large-scale community detection on YouTube for topic discovery and exploration. In *Proceedings of the 5th International AAAI Conference on Weblogs and Social Media*, pages 486–489, 2011.

[37] D. F. Gleich and C. Seshadhri. Vertex neighborhoods, low conductance cuts, and good seeds for local community methods. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data mining*, pages 597–605, 2012.

[38] D. F. Gleich, L. Zhukov, and P. Berkhin. Fast parallel PageRank: A linear system approach. Technical Report YRL-2004-038, Yahoo! Research Labs, 2004.

[39] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, March 2014.

[40] M. C. Grant and S. P. Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, volume 371 of *Lecture Notes in Control and Information Sciences*, pages 95–110. 2008.

[41] T. Hocking, J. Vert, A. Joulin, and F. R. Bach. Clusterpath: an algorithm for clustering using convex fusion penalties. In *Proceedings of the 28th International Conference on Machine Learning*, pages 745–752, 2011.

[42] Y. Hou, J. J. Whang, D. F. Gleich, and I. S. Dhillon. Fast multiplier methods to optimize non-exhaustive, overlapping clustering. *Under review*, 2015.

[43] Y. Hou, J. J. Whang, D. F. Gleich, and I. S. Dhillon. Non-exhaustive, overlapping clustering via low-rank semidefinite programming. In *Proceedings of the*

21th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, pages 427–436, 2015.

[44] A. K. Jain. Data clustering: 50 years beyond k-means. 31(8):651–666, 2010.

[45] G. Jeh and J. Widom. Scaling personalized web search. *Proceedings of the 12th international conference on World Wide Web*, pages 271–279, 2003.

[46] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48:96–129, 1998.

[47] I. M. Kloumann and J. M. Kleinberg. Community membership identification from small seed sets. In *Proceedings of the 18th ACM International Conference on Knowledge Discovery and Data mining*, pages 1366–1375, 2014.

[48] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial Computing*. Addison-Wesley, 1993.

[49] B. Kulis, A. C. Surendran, and J. C. Platt. Fast low-rank semidefinite programming for embedding and clustering. In *Proceedings of the International Conference on Artifical Intelligence and Statistics*, pages 235–242, 2007.

[50] A. Lancichinetti, F. Radicchi, J. Ramasco, and S. Fortunato. Finding statistically significant communities in networks. *PLoS ONE*, 6(4), 2011.

[51] K. Lang. Fixing two weaknesses of the spectral method. In *Proceedings of the Neural Information Processing Systems Conference*, pages 715–722, 2005.

[52] A. Lenharth, D. Nguyen, and K. Pingali. Concurrent priority queues are not good priority schedulers. In *Proceedings of the 21th International European Conference on Parallel and Distributed Computing*, pages 209–221, 2015.

[53] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[54] F. Lindsten, H. Ohlsson, and L. Ljung. Just relax and come clustering! a convexification of k-means clustering. Technical report, Linköpings universitet, 2011.

[55] S. P. Lloyd. Least squares quantization in PCM. 28(2):129–137, 1982.

[56] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *The Proceedings of the VLDB Endowment*, pages 716–727, 2012.

[57] H. Lu, Y. Hong, W. N. Street, F. Wang, and H. Tong. Overlapping clustering with sparseness constraints. In *Proceedings of the 12th IEEE International Conference on Data Mining Workshops*, pages 486–494, 2012.

[58] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, and S. M. Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations: Can geographic isolation explain this unique trait? *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[59] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi. A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally. *Journal of Machine Learning Research*, 13(1):2339–2365, 2012.

[60] F. McSherry. A uniform approach to accelerated PageRank computation. *Proceedings of the 14th international conference on World Wide Web*, pages 575–582, 2005.

[61] A. Mislove, H. S. Koppula, K. P. Gummadi, P. Druschel, and B. Bhattacharjee.

Growth of the Flickr social network. In *Proceedings of the 1st Workshop on Online Social Networks*, pages 25–30, 2008.

[62] D. Nguyen, A. Lenharth, and K. Pingali. A lightweight infrastructure for graph analytics. *Proceedings of the 24th ACM Symposium on Operating Systems Principles*, pages 456–471, 2013.

[63] D. Nguyen and K. Pingali. Synthesizing concurrent schedulers for irregular algorithms. *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 333–344, 2011.

[64] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford University, 1999.

[65] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814–818, 2005.

[66] J.-Y. Pan, H.-J. Yang, C. Faloutsos, and P. Duygulu. Automatic multimedia cross-modal correlation discovery. In *Proceedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining*, pages 653–658, 2004.

[67] J. Peng. 0-1 semidefinite programming for spectral clustering: Modeling and approximation. Technical report, Advanced Optimization Laboratory, McMaster University, 2005.

[68] J. Peng and Y. Wei. Approximating k-means-type clustering via semidefinite programming. *SIAM Journal on Optimization*, 18(1):186–205, 2007.

[69] K. Pingali, D. Nguyen, M. Kulkarni, M. Burtscher, M. A. Hassaan, R. Kaleem, T.-H. Lee, A. Lenharth, R. Manevich, M. Mndez-Lojo, D. Prountzos, and

X. Sui. The Tao of parallelism in algorithms. *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 12–25, 2011.

[70] A. Prelić, S. Bleuler, P. Zimmermann, A. Wille, P. Bühlmann, W. Gruissem, L. Hennig, L. Thiele, and E. Zitzler. A systematic comparison and evaluation of biclustering methods for gene expression data. *Bioinformatics*, 22:1122–1129, 2006.

[71] B. S. Rees and K. B. Gallagher. Overlapping community detection by collective friendship group inference. In *International Conference on Advances in Social Networks Analysis and Mining*, pages 375–379, 2010.

[72] H. Shen, X. Cheng, K. Cai, and M.-B. Hu. Detect overlapping and hierarchical community structure in networks. *Physica A*, 388(8):1706–1712, 2009.

[73] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[74] J. Shun and G. E. Blelloch. Ligra: A lightweight graph processing framework for shared memory. *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 135–146, 2013.

[75] H. H. Song, B. Savas, T. W. Cho, V. Dave, Z. Lu, I. S. Dhillon, Y. Zhang, and L. Qiu. Clustered embedding of massive social networks. *ACM SIGMETRICS Performance Evaluation Review*, 40(1):331–342, 2012.

[76] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. P. Vlahavas. Multi-label classification of music into emotions. In *Proceedings of the International Conference on Music Information Retrieval*, pages 325–330, 2008.

[77] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[78] J. J. Whang, I. S. Dhillon, and D. F. Gleich. Non-exhaustive, overlapping k-means. In *Proceedings of the 15th SIAM International Conference on Data Mining*, pages 936–944, 2015.

[79] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, pages 2099–2108, 2013.

[80] J. J. Whang, D. F. Gleich, and I. S. Dhillon. Overlapping community detection using neighborhood-inflated seed expansion. *Under review*, 2015.

[81] J. J. Whang, A. Lenharth, I. S. Dhillon, and K. Pingali. Scalable data-driven pagerank: Algorithms, system issues, and lessons learned. In *Proceedings of the 21th International European Conference on Parallel and Distributed Computing*, pages 438–450, 2015.

[82] J. J. Whang, X. Sui, and I. S. Dhillon. Scalable and memory-efficient clustering of large-scale social networks. In *Proceedings of the 12th IEEE International Conference on Data Mining*, pages 705–714, 2012.

[83] J. Xie, S. Kelley, and B. K. Szymanski. Overlapping community detection in networks: the state of the art and comparative study. *ACM Computing Surveys*, 45(4):43:1–43:35, 2013.

[84] E. P. Xing and M. I. Jordan. On semidefinite relaxations for normalized k-cut and connections to spectral clustering. Technical Report UCB/USD-3-1265, University of California, Berkeley, 2003.

[85] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 587–596, 2013.

[86] S. Zhang, R.-S. Wang, and X.-S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A*, 374(1):483–490, 2007.

[87] Y. Zhang, Q. Gao, L. Gao, and C. Wang. Priter: A distributed framework for prioritizing iterative computations. *IEEE Transactions on Parallel and Distributed Systems*, 24(9):1884–1893, 2013.

[88] Y. Zhang, Q. Gao, L. Gao, and C. Wang. Maiter: An asynchronous graph processing framework for delta-based accumulative iterative computation. *IEEE Transactions on Parallel and Distributed Systems*, 25(8):2091–2100, 2014.