Finding Community Structures in Social Activity Data

Thesis by
Chengbin Peng

In Partial Fulfillment of the Requirements
For the Degree of Computer Science

Doctor of Philosophy

King Abdullah University of Science and Technology, Thuwal,
Kingdom of Saudi Arabia

May, 2015
The thesis of Chengbin Peng is approved by the examination committee

Committee Chairperson: David Keyes
Committee Member: Mikhail Moshkov
Committee Member: Xiangliang Zhang
Committee Member: David Ketcheson
Committee Member: Suojin Wang
Finding Community Structures in Social Activity Data

Chengbin Peng

Social activity data sets are increasing in number and volume. Finding community structure in such data is valuable in many applications. For example, understanding the community structure of social networks may reduce the spread of epidemics or boost advertising revenue; discovering partitions in traffic networks can help to optimize routing and to reduce congestion; finding a group of users with common interests can allow a system to recommend useful items. Among many aspects, quality of inference and efficiency in finding community structures in such data sets are of paramount concern. In this thesis, we propose several approaches to improve community detection in these aspects.

The first approach utilizes the concept of $K$-cores to reduce the size of the problem. The $K$-core of a graph is the largest subgraph within which each node has at least $K$ connections. We propose a framework that accelerates community detection. It first applies a traditional algorithm that is relatively slow to the $K$-core, and then uses a fast heuristic to infer community labels for the remaining nodes.

The second approach is to scale the algorithm to multi-processor systems. We devise a scalable community detection algorithm for large networks based on stochastic block models. It is an alternating iterative algorithm using a maximum likelihood approach. Compared with traditional inference algorithms for stochastic block models,
our algorithm can scale to large networks and run on multi-processor systems. The time complexity is linear in the number of edges of the input network.

The third approach is to improve the quality. We propose a framework for non-negative matrix factorization that allows the imposition of linear or approximately linear constraints on each factor. An example of the applications is to find community structures in bipartite networks, which is useful in recommender systems.

Our algorithms are compared with the results in recent papers and their quality and efficiency are verified by experiments.
ACKNOWLEDGEMENTS

I would like to thank my advisor, my committee members, my family, and my friends for helping me over the past years in Saudi Arabia.

I would like to thank my advisor Professor David Elliot Keyes. He has motivated my academic research at KAUST since before my arrival, and spent significant effort on advising my work. He has also set an example of great personality through his practice. He has made my PhD study not only an advancement in research, but also an evolution in personality.

I would like to thank my committee members, Professor Xiangliang Zhang, Professor Mikhail Moshkov, Professor Suojin Wang, and Professor David Ketcheson. They have given brilliant comments and suggestions for my papers and thesis.

I would like to thank my friends. They have made the life here vivid and enjoyable.

I want to express my special appreciation to my family. They always stay with me physically or mentally, and give me hands whenever needed.

Additionally, I should also thank Saudi’s leaders, and especially King Abdullah, who make this research oasis possible on the red sea coast. Without the far-sighted vision, KAUST would not have been born. I should also thank thousands of workers who make this oasis alive. Without their hard work over day and night, we would not have a huge diversity of plants in the campus, not to say maintenance, water, or electricity.
# TABLE OF CONTENTS

Examination Committee Approval .................................................. 2
Copyright ....................................................................................... 3
Abstract ......................................................................................... 4
Acknowledgements ........................................................................ 6
List of Abbreviations ..................................................................... 10
List of Figures ................................................................................ 11
List of Tables .................................................................................. 12

1 Community Structures in Networks ........................................... 13
   1.1 Motivation and Objectives .................................................. 15
      1.1.1 Problem Formulation ............................................... 15
      1.1.2 Reduce the Problem Size ......................................... 18
      1.1.3 Increase the Scalability ............................................ 21
      1.1.4 Improve the Quality ................................................. 22
   1.2 Summary of Contributions .................................................. 26

2 Related Work .............................................................................. 28
   2.1 Acceleration for Community Detection Algorithms ............... 29
   2.2 Parallel Implementations for Community Detection ............... 30
   2.3 Non-negative Matrix Factorization and Recommender Systems . 31

3 Reduce the Problem Size ............................................................. 34
   3.1 Algorithm ............................................................................ 34
      3.1.1 Find the $K$-core Subgraph ....................................... 35
      3.1.2 Apply a Community Detection Algorithm on the $K$-core . 35
      3.1.3 Recover the Community Structure for the Whole Graph .. 36
3.1.4 Time Complexity ............................................ 40
3.2 Theoretical Analysis ........................................... 40
  3.2.1 Node Remaining Ratios in \( K \)-core ...................... 40
  3.2.2 Selecting an Appropriate \( K \) .............................. 42
  3.2.3 Maximizing the Modularity in \( K \)-core .................... 43
3.3 Experiments ..................................................... 46
  3.3.1 Data ..................................................... 46
  3.3.2 Implementation Details ..................................... 46
  3.3.3 Results and Analysis — Large-scale Networks ............. 49
  3.3.4 Results and Analysis — Multiple Real-world Networks .... 50
3.4 Discussion ....................................................... 58

4 Increase the Scalability 59
  4.1 Stochastic Block Model ......................................... 59
  4.2 Methodology .................................................... 60
    4.2.1 Estimation Algorithm ..................................... 60
    4.2.2 Parallelism ............................................... 66
  4.3 Model Analysis: An Ideal Example .......................... 68
    4.3.1 Self-Assembling Process ................................ 69
    4.3.2 Graphs with Heterogeneous Partitions ..................... 70
    4.3.3 Minimum Gap between \( p \) and \( q \) ......................... 71
    4.3.4 Convergence Rate ......................................... 74
  4.4 Experimental Results .......................................... 74
    4.4.1 The Serial Algorithm ..................................... 74
    4.4.2 The Parallel Algorithm ................................... 75
    4.4.3 Microscopic Example on dblp-Coauthor Network .......... 80
  4.5 Discussion ....................................................... 80

5 Improve the Quality 82
  5.1 General Problem Definition .................................... 82
    5.1.1 Algorithm for Weighted Non-negative Matrix Factorization 84
  5.2 Weighted and Constrained Non-negative Matrix Factorization ... 85
    5.2.1 Algorithm ................................................. 85
    5.2.2 Convergence Analysis of the Multiplicative Algorithms .... 87
  5.3 Sample Implementations ........................................ 93
    5.3.1 Procedure to Calculate \( H \) in Eq. (5.31) .................. 93
    5.3.2 Procedure to Calculate \( W \) in Eq. (5.30) .................. 95
5.4 Experiments ................................................................. 97
  5.4.1 Application to Movielens Rating Data ....................... 98
5.5 Discussion ............................................................... 105

6 Conclusion and Future Work 106
  6.1 Conclusion ............................................................ 106
  6.2 Directions for Future Work ........................................ 107

References 109
# LIST OF ABBREVIATIONS

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-NMF</td>
<td>Constrained Non-negative Matrix Factorization</td>
</tr>
<tr>
<td>ER</td>
<td>Erdős-Rényi</td>
</tr>
<tr>
<td>LFR</td>
<td>Lancichinetti-Fortunato-Radicchi</td>
</tr>
<tr>
<td>MPI</td>
<td>Message Passing Interface</td>
</tr>
<tr>
<td>NMAE</td>
<td>Normalized Mean Absolute Error</td>
</tr>
<tr>
<td>NMF</td>
<td>Non-negative Matrix Factorization</td>
</tr>
<tr>
<td>NMI</td>
<td>Normalized Mutual Information</td>
</tr>
<tr>
<td>RMSE</td>
<td>Root-Mean-Square Error</td>
</tr>
<tr>
<td>SBMs</td>
<td>Stochastic Block Models</td>
</tr>
<tr>
<td>W-NMF</td>
<td>Weighted Non-negative Matrix Factorization</td>
</tr>
<tr>
<td>WC-NMF</td>
<td>Weighted and Constrained Non-negative Matrix Factorization</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1 Effectiveness of $K$-core subgraph community detection ............ 20
1.2 An example of the bipartite graph .................................. 24
2.1 A brief categorization of community detection algorithms related to our contributions ........................................... 29
3.1 Results of $K$-core acceleration on com-Youtube and com-Amazon for community detection ......................... 49
3.2 Relative run time over different $K$’s, Part 1 .......................... 51
3.3 Relative run time over different $K$’s, Part 2 ......................... 52
3.4 Modularity over different $K$’s, Part 1 ................................. 53
3.5 Modularity over different $K$’s, Part 2 ................................. 54
3.6 Normalized cut over different $K$’s, Part 1 ............................. 55
3.7 Normalized cut over different $K$’s, Part 2 ............................. 56
4.1 Comparison to popular algorithms. (a) and (b) represent the variation of average accuracy and running time over different choices of $\mu$. (c) and (d) represent the variation over different choices of graph size $N$. 76
4.2 Comparison on benchmark data ........................................... 77
4.3 Number of communities versus community size ..................... 78
4.4 Subgraph of the dblp-Coauthor network, where colors represent the community labels determined by different algorithms ........... 81
5.1 Tests on data with upper bound applied in C-NMF and WC-NMF . 101
5.2 Tests on data with squeezed lower bound in C-NMF and WC-NMF . 102
5.3 Tests on data with most popular items removed ...................... 103
5.4 Top-N test on data with most popular items removed ............... 104
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Data sets</td>
<td>47</td>
</tr>
<tr>
<td>3.2</td>
<td>Algorithms using $K$-core on com-Amazon network</td>
<td>50</td>
</tr>
<tr>
<td>3.3</td>
<td>Remaining nodes in $K$-core in multiple graphs</td>
<td>57</td>
</tr>
<tr>
<td>4.1</td>
<td>Properties of the test graphs</td>
<td>77</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison on real-world data</td>
<td>79</td>
</tr>
<tr>
<td>5.1</td>
<td>Notation of symbols for Non-negative Matrix Factorization</td>
<td>83</td>
</tr>
<tr>
<td>5.2</td>
<td>List of algorithms for comparison</td>
<td>99</td>
</tr>
</tbody>
</table>
Chapter 1

Community Structures in Networks

A network comprises a set of nodes and a set of edges, and nodes are connected in pairs by edges [1, 2]. Networks are widely used to represent the relationship between objects. For example, in a road network, nodes can be locations and edges can be the roads between those locations [3]. In molecular networks, nodes represent sets of molecules, and edges are conserved molecular interactions [4]. In social networks, nodes can be individuals such as authors, and edges can be the relationship between them such as the co-authorships [5].

The bipartite network is a special case, in which the nodes can be divided into two sets and there are no internal edges between nodes within a set. Some real world networks can be better described by this model. For example, human disease network depicts connections between genetic disorders and disease genes [6]. In consumer product networks, the two sets of nodes are consumers and products respectively, and an edge between two nodes represent a transaction on a product initiated by a consumer [7]. In recommender systems, the rating from users to items forms a bipartite network [8].

Real world networks can be characterized by different aspects. From the edge connectivity perspective, if we define the number of edges connected to a node as
its degree, degrees in a real world network usually follow the power-law distribution\(^1\) [9, 11, 12]. In terms of the shortest path between any pair of nodes, many networks exhibit the “small world” phenomenon indicating that the distances are usually short [13, 14, 15]. The “small world” phenomenon of a network also suggests that neighbors of a randomly selected node are also likely to be connected [13].

In addition to these characteristics, many real-world networks have inherent community structure that is also called clustering [16]. Different definitions of community can be proposed, but an essential notion is that nodes within the same community are highly connected to one another. The community structures usually have practical meanings in real world scenarios. In social networks, each community may contain individuals with similar interest or background [17]. For example, people from a common high school can be considered as a community [18]. In transportation networks, community structure may arise due to multinational interactions [19]. For example, in worldwide air transportation, at an appropriate granularity the Arabian Peninsula countries are in the same community as India and many countries in Northeastern Africa [19]. In biological networks, communities may indicate strong correlations between proteins or genes. For example, in the yeast protein-protein interaction network, proteins within the same community can form a functional module [20].

There are several types of community structures. A non-overlapping community structure indicates that each node in the network only belongs to one community [16]. For example, in the Zachary karate club network, a professional player only belongs to one club [21]. In contrast, in an overlapping community structure, each node can belong to multiple communities [22]. For example, if we consider each conference as a community, a researcher in a collaborative network can show up in multiple conferences.

Identifying community structures has many applications, such as in network dy-

\(^1\)A random variable \( x \) follows the power-law distribution if \( p(x) \propto x^{-\alpha} \), where \( \alpha \) is a constant parameter larger than one [9, 10].
namics where it is used to understand the spreading of a disease [23]. It can find protein complexes in biological networks and topical or disciplinary groups in collaborative networks [24, 25]. Communities in bipartite networks indicate node similarities, and this can be used for recommender systems [8, 26]. Many aspects of community detection problems have also been explored recently, such as the impact of the degree assortativity [27], the contribution of node attributes [28], and the detection for community outliers [29].

Understanding community structure in a graph is a major algorithmic challenge, with a wide spectrum of applications across many disciplines. In this research, we propose community detection algorithms to improve the community detection in three different aspects: speed, scalability, and quality. The speed is improved for general community detection algorithms by providing a framework to reduce the problem size for the main algorithms. The scalability is improved by proposing a linear time algorithm and parallelizing it using Stochastic Block Models (SBMs). The quality is improved by the imposition of additional constraints to an existing algorithm.

1.1 Motivation and Objectives

1.1.1 Problem Formulation

Loosely speaking, detecting communities is equal to finding clusters of nodes such that nodes within each cluster are more frequently connected than nodes from different clusters. However, if we only maximize the edge density\(^2\) for each cluster, then each cluster must be a complete subgraph in which every two nodes are connected. However, in real-world data, the edge information is usually incomplete, and clusters obtained by simple density maximization cannot reflect the real communities. To

\(^2\)Edge density is the number of actual edges connecting a set of nodes divided by the maximum possible number of edges that can be placed.
overcome this issue, three approaches are discussed in this thesis.

One approach is to maximize the modularity. Modularity is obtained by comparing intra-community edge density of the original network to the density of the randomized network with the same degree distribution. It is defined as follows \[30, 31\]

\[
Q = \sum_r (e_r - a_r^2),
\]

(1.1)

where \(e_r\) is the ratio of degrees caused by intra-community edges to the total degrees of nodes in community \(r\), and \(a_r\) is expected value of \(e_r\) when nodes are randomly connected and preserve their original degrees. For community detection, we aim to maximize the modularity. Unfortunately, the problem of finding the global maximum is NP-complete \[32\], while the time complexity of different approximation algorithms varies, for example, \(O(N \log N)\) \[30\] and \(O(N^2 \log N)\) \[33\], where \(N\) is the number of nodes.

A second approach to community detection is by using SBMs \[34, 35\]. There are different definitions of SBMs, but for undirected graphs they typically contain two variables. One is a community indicator represented by \(Z \in \{0, 1\}^{N \times K}\) for \(N\) nodes, where \(K\) is the number of communities. Entry \(Z_{ir} = 1\) means node \(i\) belongs to block \(r\), and each row of \(Z\) contains only one nonzero entry. The other variable is a probability indicator \(B \in [0, 1]^{K \times K}\), where \(B_{r,k}(r \neq k)\) represents the probability of connections between nodes drawn from block \(r\) and \(k\), respectively, and \(B_{r,r}\) is the probability of connections inside the block. Connections are generated independently from these probabilities \[16\]. In this setting, finding community structures is equivalent to finding the most likely variable setting for a given network \(G\)

\[
\max_{Z,B} Pr(Z, B|G).
\]

(1.2)
This optimization is again an NP-complete problem even with a given $B$ [34]. Approximate algorithms also have different time complexities, such as $O(N^2)$ [35] and $O(K^2N^2)$ [36].

We may also find the community structure using the idea of non-negative matrix factorization (NMF) [37, 38]. A basic NMF algorithm factorizes a matrix into two by minimizing the residual

$$
\min_{W \geq 0, H \geq 0} ||V - WH||_F, \quad (1.3)
$$

where $V$, $W$, and $H$ are matrices, $|| \cdot ||$ is a matrix norm, and $\geq$ is an element-wise operator. We may consider a bipartite graph containing two sets of nodes. There are $M$ nodes from Set 1 and $N$ nodes from Set 2. According to the definition of bipartite networks, only pairs of nodes from different sets can be connected. Therefore, if $V \in \mathbb{R}^{M \times N}$ is the adjacency matrix, $V_{ij}$ indicates the weight of the edge connecting node $i$ in Set 1 and node $j$ in Set 2. The sizes of $W$ and $H$ become $M \times K$ and $K \times N$ respectively, where $K$ is the number of communities for each set of nodes. If certain entries in the $k^{th}$ column of $W$ are relatively large, then the corresponding subset of the Set 1 nodes becomes the $k^{th}$ community of Set 1, because nodes in this community are more densely connected to a subset of Set 2 nodes indicated by the $k^{th}$ row of $H$. For some $V$, there is an exact solution of $W$ and $H$ so that the objective function can be zero, which is called the exact NMF [39]. The problem of finding solutions to exact NMF is proved to be NP-hard [39]. For an approximate solution to Eq. (1.3), there are algorithms of time complexity $O(MNK)$ [40, 41].

In this thesis, we improve the three approaches in three different aspects respectively. We reduce the problem size for general modularity maximization algorithms in Chapter 3, enhance the scalability for inferring stochastic block models in Chapter 4, and improve the quality for non-negative matrix factorization algorithms in Chapter
1.1.2 Reduce the Problem Size

The first aspect of improving community detection algorithms is to reduce the running time, and we focus on non-overlapping community detection. There are many algorithms developed for the community detection problem [16]. A typical community detection algorithm accepts adjacency matrices of the graphs as input, and generates the community labels for the nodes automatically. Some algorithms may additionally need the number of communities and other parameters. Examples include Bayesian methods maximizing the likelihood\(^3\) of a stochastic block model [42], spectral algorithms utilizing eigenvalue decomposition [43], and fast algorithms using modularity optimization [44, 45]. These methods all share a common feature: their cost is at least linear in the number of nodes (denoted by \(N\)). For example, the time complexity of traditional spectral clustering [46] and modularity maximization [47] is \(O(N^2)\). Many recent algorithms aim to reduce the time complexity with respect to \(N\), but even for those with complexity \(O(N)\), their actual complexity may be multiplied by large coefficients, for example, number of clusters [48, 49]. Intuitively, these algorithms spend the same efforts on different nodes, while our framework reduces their running time on peripheral nodes. Our goal is to work with a reduced representation that has fewer nodes, and consequently reduce the time complexity even for some linear time algorithms. Unlike typical sampling algorithms that aim to maintain the degree distribution or node diversity in the subgraph [50], our reduced representation can capture the essence of the community structure of the original.

We define \(K\)-core as the largest subgraph in which each node has at least \(K\) edges [51]. The fundamental observation of this research is that community structure

\(^3\)The likelihood function of a stochastic model takes an observed instance and the parameters of the model as inputs. In this thesis, it represents the probability of generating the input graph when the parameters of the stochastic block model are given.
is generally preserved in the $K$-core for reasonable values of $K$. For example, to compute the 4-core: remove all nodes of degree less than 4, update the degrees of all remaining nodes, and repeat until only nodes of degree 4 or higher remain. See Fig. 1.1(a) for an example of a graph and its 4-core. Algorithms for finding the $K$-core are efficient and amenable for parallelization [52]. We propose to run community detection first only on the much smaller $K$-core and then use a fast heuristic to find community labels for the remaining nodes.

In Fig. 1.1(b), we show sample results for the ego-Facebook network\textsuperscript{4} ($4 \times 10^3$ nodes, $8.8 \times 10^4$ edges). This is an interesting example because the network is fairly dense, so community structure is retained for fairly large values of $K$. We use modularity optimization [47] as an exemplar community detection method to illustrate our idea. Modularity is widely used as a criterion to evaluate the quality of detected communities. The modularity optimization algorithm initializes each node as a sole member of one community, and agglomerates communities until the modularity decreases [47]. For $K = 1$, we see the results of applying the community detection method to the full network, and this serves as the baseline. As we increase $K$, the number of nodes and edges decreases. At $K = 20$, we have 50% of nodes and 80% of edges remaining. The number of edges does not decrease as quickly as the number of nodes since we are removing low-degree nodes. Because modularity optimization is an expensive procedure, the time reduces faster even than the fraction of nodes, to less than 20% of the total time for the baseline at $K = 20$. The number of communities is determined automatically, but the highest number of communities found in this example is 14. It is also worth noting that our framework fails when $K$ is too large, for example, at $K = 39$ in Fig. 1.1(b). Therefore, we also propose a method to choose an appropriate $K$ so that we can gain significant acceleration without undermining

\textsuperscript{4} This network is a combination of 11 ego-networks obtained from SNAP (Stanford Network Analysis Platform, \url{http://snap.stanford.edu/}). Each ego network is a sample of a large network (Facebook friendship network in this case), containing one central node and all the neighboring nodes to the central node.
(a) Reduction to 4-core by removing edges and nodes outside the core

(b) Sample results

Figure 1.1: Effectiveness of K-core subgraph community detection
the quality. For this example, by assuming the smallest community size is 100, our method determines $K = 29$, which is a good balance between time reduction and quality preservation. We measure the quality in terms of modularity. In this case, the solution is always at least as good as the baseline.

Related work and details of our proposed algorithms are described in Section 2.1 and Chapter 3 respectively.

### 1.1.3 Increase the Scalability

In the second aspect, we aim to scale a community detection algorithm to parallel systems. Again, we consider non-overlapping community structures. Many community detection algorithms handle such a problem [24, 44, 47, 49, 53, 54, 55, 56, 57]. However, they come along with different limitations for large graphs, for example, in handling small communities within a large graph [58, 59, 60]. Alternatively, model-based methods can produce more reliable and accurate results when the assumptions are in accordance with the real graphs. SBMs are among important probabilistic tools describing the connectivity relationship between pairs of nodes [61], and have received considerable attention both in theoretical [35] and application domains [62]. For large graphs, a linear time inference algorithm for such models is preferable.

On the other hand, we can also utilize parallel computers to detect communities on large graphs. Parallel computing has been proposed for decades [63, 64]. It can decrease the execution time of applications by executing them on multiple processors [65].

Parallel computers are classified according to the programming model natural to their memory structure, whether shared, or distributed, or hybrid. In a shared memory model [66], each processor can access the entire system memory, while each processor in a distributed memory model [67] can access only its local memory and requires an interprocessor network to communicate data to and from other local mem-
ories. In the hybrid model [68], subsets of processors share memory and these subsets are connected by a communication network. The hardware supporting these programming models need not fully resemble them. The convenient global addressing of shared memory can be implemented over distributed memory hardware and the efficient controlled access of distributed memory over shared memory hardware. However, there is a general association of distributed memory protocols with the systems of the largest memory capacities. Therefore, in our algorithm, we choose the message passing model as the most performance-portable to the hardware resources that will be required for graph applications of the future.

In short, we propose a multi-stage likelihood maximization algorithm based on SBMs, which runs in linear time and is scalable to parallel systems. Related work and details of our proposed algorithms are described in Section 2.2 and Chapter 4 respectively.

1.1.4 Improve the Quality

The third aspect is to improve the quality of community detection results via non-negative matrix factorization. We first propose a general framework to improve the performance by allowing additional weights and constraints, and then analyze a specific example on a user-movie bipartite network. If we consider the matrix before factorization as the training data, as it is in this example, the quality is measured by comparing the computed values to the ground truth values that are excluded from the training matrix.

Non-negative Matrix Factorization (NMF) [41, 69] has been proposed for more than a decade. Unlike traditional matrix factorization methods, in this algorithm no entries of the matrices are less than zero. NMF decomposes a non-negative matrix approximately into the product of two lower-rank non-negative matrices. One advantage is that it enables parts-based representation by additive-only combinations [40].
That is, it can represent the whole object as a combination of small parts. Such a representation allows us to interpret the basic patterns of the data matrix for certain applications. For example, in face recognition the data matrix is composed by image pixels [70], and color components for each pixel are represented by non-negative values. The basic patterns of facial images can be different kinds of noses, mouths, eyes, etc., that are also represented by non-negative values. However, without non-negative constraints in the matrix factorization, negative entries will come into the factors and the desired basic patterns cannot be correctly revealed. NMF has better clustering capabilities as well compared to traditional K-means [71]. A typical approach to NMF is iterative: alternatively fix one of the two factors and calculate the other in successive iterations [41, 69]. The simplicity of this approach has led to wide adoption in areas such as face recognition [40, 70], text mining [40, 72, 73], data clustering [71], graph mining [38, 74, 75], privacy protection [76], speech denoising [77], item recommendation [78, 79], etc.

It is also worth noting that identifying communities of a bipartite network using NMF is similar to recommending items to users in a user-item network. For example, to recommend suitable movies for a specific user, we first group movies and users according to their connections, and recommendations for a specific user are picked from his most preferred movie group. The problem can be abstracted as follows. Let $V$ be a matrix where $V(i, j)$ is the rating value of the $i$th user for the $j$th item. In practice, most of the entries in $V$ are unknown, and we want to predict them by some algorithm based on the known entries. There are two stages for algorithm evaluation. In the training session, some known entries of $V$ are removed, and the resulted matrix $V$ is the training set to feed the algorithm as input. In the testing session, the previously removed entries are used to measure the prediction (output) accuracy of the algorithms.

For example, we may have a small bipartite graph whose adjacency matrix $V$ is
Figure 1.2: An example of the bipartite graph

represented by Eq. (1.4). $W$ and $H$ are two factors of $V$, and are obtained from NMF. A subgraph of the bipartite graph is shown in Fig. 1.2, in which $u_1, u_2, u_3, \text{ and } u_4$ represent four users, and $m_1, m_2, ..., \text{ and } m_6$ are six movies. The width of each edge is proportional to the rating of movies from different users. As a result of NMF on the known entries of $V$, we can see that the first two rows of $W$ are the same, meaning that $u_1, u_2, \text{ and } u_3$ are from the same community, sharing common interests. By the columns of $H$, we can also learn that movies are clustered into two communities $(m_1, m_2, m_3) \text{ and } (m_4, m_5, m_6)$, and movies belong to the same community sharing common ratings from the same user. We can use such knowledge on community structure to build a recommender system. In this example, if we want to predict the rating from $u_3$ to $m_2$, because we know that $(m_1, m_2, m_3)$ are in the same community and $u_3$ has high ratings (5) to movies in this community, we may also consider the
missing rating is high (5).

\[
V = \begin{bmatrix}
5 & 5 & 5 & 0 & 0 & 0 \\
5 & 5 & 5 & 0 & 0 & 0 \\
5 & ? & 5 & 0 & 0 & 0 \\
1 & 1 & 1 & 4 & 4 & 4 \\
\vdots & \vdots \\
\end{bmatrix}_{100 \times 6}
\]

\[
W = \begin{bmatrix}
5 & 0 \\
5 & 0 \\
5 & 0 \\
1 & 4 \\
\vdots \\
\end{bmatrix}_{100 \times 2}
\]

\[
H = \begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 \\
\end{bmatrix}_{2 \times 6}
\]

Different application domains usually have different requirements for the NMF algorithm, and the ability to impose specialized constraints is eagerly sought. In this research, we introduce a framework that allows a variety of equality and inequality constraints to be included in the NMF algorithms, with adjustable tightness of these constraints. We prove that if linear constraints are added to the factors, then algorithms following our multiplicative framework can converge. This property can make the adoption of the NMF algorithm attractive in many application fields requiring variations of constraints.

As an example to demonstrate the effectiveness of specialized constraints, in the community detection inspired recommender system research, we devise two special constraints in Weighted and Constrained Non-negative Matrix Factorization (WC-NMF) based on Weighted Non-negative Matrix Factorization (W-NMF) [79] and
Constrained Non-negative Matrix Factorization (C-NMF) based on NMF [69]. These two specifications can enhance the formula, for example, by maintaining the upper bound of the missing entries, as well as approximately orthogonalizing factor matrices. Our algorithm allows flexible weights for these constraints when embedded with the objective function.

We use several metrics to compare the quality of the recommended items that are the consequences of the node clustering by community detection. Our C-NMF and WC-NMF algorithms significantly and stably improve the performance of NMF algorithms in top-N metric, and slightly improve them in terms of Root-Mean-Square Error (RMSE) [80, 81] and Normalized Mean Absolute Error (NMAE) [79, 82] using a squeezed lower-bound strategy. On the MovieLens data without the 6% most popular items, C-NMF achieves a much better result in top-N metric than SVD50 and SVD150, the most predictive method in Cremonesi et al.'s experiment [83].

Related work and details of our proposed algorithms are described in Section 2.3 and Chapter 5 respectively.

1.2 Summary of Contributions

Our work contributes to community detection algorithms in three aspects.

• **Time Reduction for Community Detection Algorithms**: We propose a fairly intuitive but nonetheless novel method for accelerating community detection by focusing the bulk of the effort on the $K$-core. We suggest a fast heuristic for providing labels to nodes outside the $K$-core. We show positive results (similar to Fig. 1.1(b)) for a variety of popular community detection algorithms on various real-world data sets. Overall, we find that we get substantial decreases (up to 90%) in runtime without sacrificing quality, measured in terms of modularity and normalized cut.
• **Parallelism for Inferring a Stochastic Block Model for Large Graphs:**
  We devise our algorithm based on the coordinate descent approach and use appropriate approximations to simplify computations. Our algorithm runs in linear time with respect to the number of edges. We propose a parallel algorithm based on message-passing that can utilize multi-processor systems. We overlap the communication and computation in the algorithm, so that for large graphs, it can achieve significant speedup proportional to the number of processors. To the best of our knowledge, it is the first parallel algorithm for inferring SBMs. Our algorithm can produce high-quality results. It first considers each node as one community in the initialization and then uses a multi-stage iterative strategy to construct larger communities gradually. The accuracy of our algorithm is verified on sparse and dense, small and large graphs as benchmark data. Our algorithm outperforms traditional community detection algorithms empirically as measured by Normalized Mutual Information (NMI) with respect to the ground truth\(^5\).

• **Quality Improvement for Node Clustering on Bipartite Graphs:** We introduce weights and general constraints in non-negative matrix factorization. In a specific application, we can adopt appropriate constraints and weights to improve the factorization quality. We demonstrate the effectiveness through a recommender system example inspired by community detection on bipartite graphs.

\(^5\)The ground truth community structure is usually extracted from the existing groups in the real world data or from the generation of synthetic data.
Chapter 2

Related Work

Generally, there are two kinds of community structures: non-overlapping communities, in which each node belongs to one community, and overlapping communities, in which one node can belong to several communities, and thus the communities are overlapped. Community detection algorithms can be divided into these two categories accordingly as shown in Fig. 2.1.

Non-overlapping community detection algorithms include statistical inference algorithms using stochastic blockmodels [42] and other algorithms, such as modularity maximization [47] and spectral clustering [46]. Our $K$-core acceleration framework is designed for this category of algorithms, and our parallel inference algorithm aims to enhance the scalability for using the stochastic blockmodels.

Overlapping community detection algorithms include statistical inference algorithms as well using mixture membership stochastic blockmodels [84]. Non-negative matrix factorization is also used [37], and we propose a framework that allows constraints for the factorization to improve the quality, illustrated by an example on recommender systems inspired by community detection.
2.1 Acceleration for Community Detection Algorithms

Much research has been devoted to community detection algorithms [16, 85]. However, scalability remains a major issue. One strategy is to split the graph into very small subgraphs and optimize each small problem locally [53, 86], in which case the quality is usually compromised. Another strategy is to include additional phases to optimize the community labels initialized by the aforementioned local optimum [44, 45, 87].

Perhaps the most closely related work to ours focuses on removing unimportant edges [88]. This method preserves all the nodes and at least one edge per node, so it may require input graphs with relatively high average degree to achieve a good performance.

Scalability can also be achieved via high-performance implementations such as multi-threaded algorithms that run on multiple processors [55, 89, 90]. Some other fast algorithms rely on additional information, such as the practical meaning corre-
sponding to each node or edge [91, 92]. Our K-core framework is compatible with these improvements.

Some papers discuss the existence condition of a K-core subgraph [93, 94]. Some study the relationship between the K-core and the bond percolation properties [95]. Bond percolation characterizes the connectivity of nodes due to randomly selected edges and is quite similar to the label propagation in community detection [53]. Recently, K-core is also used to evaluate the robustness of communities, which reflects the collaborative nature in co-authorship graphs [96].

2.2 Parallel Implementations for Community Detection

Many community detection algorithms have been proposed in the past decade, e.g., modularity optimization [47, 97], block modularity maximization [98], spectral clustering [46], and the Louvain method [44]. Nevertheless, they may have some disadvantages. For example, modularity-based algorithms may have resolution limit [59], which means they may be unable to find small communities without modifying their objective function [99]. Some algorithms also suffer from high time complexity, such as spectral clustering with a typical time complexity no less than $O(NK^2)$ [48], where $N$ is the number of nodes and $K$ is the number of communities. Those with high time complexity are not suitable for large graphs or graphs with many communities.

Recent years have seen work on parallel implementations for non-SBM community detection algorithm. For example, Riedy et al. [54], Staudt et al. [57], and Bhowmick et al. [100] proposed algorithms aiming at modularity maximization. Parallel algorithms for spectral clustering are also proposed [49, 101]. However, those implementations are mostly on shared-memory systems, which constrain the maximum speedup due to the hardware limit. In addition, none of these implementations can suitably
address the problem in SBMs.

On the other hand, as discussed in the previous section, using SBMs is theoretically more reliable under particular assumptions. Many algorithms have been proposed to infer the parameters of SBMs, such as Bayesian estimation [42] and nuclear norm minimization [102]. Bayesian estimation defines the prior distributions on latent variables (community labels of nodes) and maximizes the posterior distribution when a graph is given [42]. Nuclear norm minimization of a matrix minimizes the sum of its singular values, and their algorithm is verified on graphs containing one thousand nodes.

Our fast and scalable algorithm fills the gap of processing very large graphs using stochastic block models.

2.3 Non-negative Matrix Factorization and Recommender Systems

Non-negative matrices refer to matrices in which none of the entries are less than zero. Non-negative matrix factorization approximates a non-negative matrix $V$ by the multiplication of two lower rank non-negative matrices $W$ and $H$: $V \approx WH$ [40]. Several variations have been proposed for different applications. Orthogonal non-negative matrix tri-factorization was introduced [78], in which the tightness and the type of the constraint is fixed by the algorithm [71] and consequently the application domain is limited. Liu et al. [103] propose a label-based constraint on the objective function of the factorization algorithm for images.

On the other hand, a weighted non-negative matrix factorization (W-NMF) [79] is devised for data sets with missing values. However, because NMF is an approximate factorization, some entries in $WH$ can be unrealistically large, especially when those entries have weight zero in W-NMF.
In this research, we provide a framework that allows to apply a variety of constraints in both weighted and unweighted NMF. We adopt this framework for a recommender system as an example inspired by community detection. This example utilizes the community structure of bipartite networks that is similar to the overlapping community structure in general networks found by NMF [37, 38], but those traditional NMF methods pay little attention to the upper bounds and other constraints.

Recommender systems suggest items to users [104]. For example, such a system for online shopping can suggest goods to buyers. In this work, we consider a system recommending movies to audiences based on their previous ratings of other movies. We define a data matrix containing each user’s ratings of watched movies, and we want to find the best rated movie for each user among movies that he has not yet watched. To solve this kind of recommendation problem, methods based on matrix factorization are among the most popular techniques.

Matrix factorization can find communities of users and movies on such bipartite graphs. With this finding, it can predict the user ratings on movies not watched yet. The top algorithms [81] for the Netflix Prize\(^1\) in both 2007 and 2008 were based on matrix factorization, although they took many practical factors into account: bias, temporal dynamics, etc. Takács et al. [80] investigate a matrix factorization approach using many parameters for biases and regularization terms. Kurucz et al. [105] discuss how to implement an SVD algorithm on large data sets with missing values. It can also be applied on recommender systems for rating prediction. Principal component analysis (PCA) is also effective in some cases [82]. For performance improvements, some researchers embed the users and items into an Euclidean space [106], while some others emphasize the importance of negative feedback when the data set is very sparse [107]. Some researchers also analyze the algorithm robustness from noisy rating attacks [108].

\(^1\)http://www.netflixprize.com/
For the example of recommender systems, compared to traditional algorithms, our constrained non-negative matrix factorization can apply upper and lower bounds on the unknown ratings to improve the accuracy of the prediction.
Chapter 3

Reduce the Problem Size

3.1 Algorithm

A $K$-core framework is introduced that consists of three steps and uses any one of a set of standard community detection methods in an inner step. The first step is to reduce the whole graph to a $K$-core. The second step uses an existing algorithm to generate community labels for nodes in the $K$-core. The third step is to find community labels for the remainder of the graph via a fast algorithm. Algorithm 1 is the pseudocode of the framework; here, $G$ is the original graph, $K$ is the desired core number, $G_K$ is the $K$-core subgraph, $g_K$ is the community assignment for the $K$-core, and $g$ is the community assignment for all nodes. We discuss the choice of $K$ in Section 3.2.

**Algorithm 1** Accelerated $K$-core Community Detection

1: **input** Graph $G$, Parameter $K$
2: **output** Labels $g$
3: (1) $G_K \leftarrow \text{Kcore}(G, K)$
4: (2) $g_K \leftarrow \text{CommunityDetection}(G_K)$
5: (3) $g \leftarrow \text{Recover}(G, g_K)$
6: **return** $g$
3.1.1 Find the $K$-core Subgraph

There are many algorithms available to find the $K$-core of a graph [52, 93]. A standard approach [93] is to traverse all the nodes and remove those connected by less than $K$ edges as depicted in Fig. 1.1(a). The edges connected to the removed nodes are removed as well. The traversal and removal may be repeated multiple times, until all the remaining nodes have degrees at least $K$. The pseudocode is presented in Algorithm 2. The resulting $K$-core is unique.

Algorithm 2: Algorithm to Find the $K$-core Subgraph

1: **input** Graph $G$, Parameter $K$
2: **output** Subgraph $G_K$
3: $G_K \leftarrow G$
4: **while** $G_K$ is not a $K$-core **do**
5: \hspace{1em} Find all nodes in $G_K$ whose degree is less than $K$
6: \hspace{1em} Remove those nodes and their incident edges
7: \hspace{1em} Update the node degrees for the remaining nodes
8: **end while**
9: **return** $G_K$

3.1.2 Apply a Community Detection Algorithm on the $K$-core

Applying a community detection algorithm to the $K$-core is the main part in our framework, and any existing algorithm can be used here. The biggest performance gains will be for algorithms with high quality and high time complexity.

The main algorithm detects the community structure only for nodes in the $K$-core, which is the main reason for the running time reduction. Here we discuss briefly the effect of our framework on different algorithms. For algorithms with high quality and fairly fast running time, our framework can reduce the running time significantly with the quality preserved. Those algorithms include the Louvain method [44], Martelot’s method [45], and modularity optimization [47]. For very fast algorithms such as label
propagation [53], our framework does not improve the running time except for very large graphs. Some algorithms like the Bayesian method [42] and spectral clustering [46] need to specify potential community numbers before running, but our framework can increase the community number (in the “recover” step) from the potential values for the sake of the quality.

We note that any method may be used for community detection, though each may rely on a different merit function. This is discussed further in the next subsection.

3.1.3 Recover the Community Structure for the Whole Graph

When the community structure $g_K$ of the $K$-core is known, the third step is to recover the node labels for all the nodes in and outside the $K$-core. With the knowledge of $g_K$, it becomes possible to use a simple algorithm to achieve high quality. In fact, the exact details of the recovery algorithm do not have much impact on the overall success of the method. Low degree nodes are generally not critical for community structure.

The recovery step is composed of two stages: inferring community labels for nodes outside the $K$-core and optimizing the structure for the whole graph, as shown in Algorithm 3.

**Algorithm 3** Algorithm to Find the Community Structure of the Whole Graph

1: **input** Graph $G$, Labels $g_K$
2: **output** Labels $g$
3: $s \leftarrow$ set of unlabeled nodes, roughly sorted in descending order according to the proportion of their neighbors inside the $K$-core
4: **repeat**
5:   **for all** nodes $i \in s$ do
6:     $g(i) \leftarrow$ plurality vote of labeled neighbors of $i$
7:   Remove $i$ from $s$
8: **end for**
9: **until** no more nodes can be labeled
10: $g \leftarrow \text{SecondStageAlgorithm}(G, g)$
11: **return** $g$
In the first stage, we sort the unlabeled nodes in a descending order according to the ratio of their labeled neighbors. Sorting is necessary to make sure that the community structure information of the $K$-core is sufficiently utilized, so it runs at the beginning of this stage.

A sorting algorithm typically has a time complexity of $O(n \log n)$. However it can be reduced to $O(n)$ by using a coarse sorting instead. Since the ratio of neighboring nodes is in the interval of $[0, 1]$, in coarse sorting we can divide the interval into a number of evenly-sized bins and sort accordingly. For instance, if we have ten bins for interval $[0, 1]$, we can use Bin 1 for the range of $[0, 0.1]$, Bin 2 for $(0.1, 0.2]$, and Bin 10 for $(0.9, 1.0]$. Then, the nodes with ratios 0.45, 0.6, and 0.85 are put into bins 5, 6, and 8, respectively.

By the order from the sorting above, we assign community labels of the unlabeled nodes according to the most frequent label among their neighbors. Each assignment may impact subsequent labels. Additionally, we repeat the procedure (without resorting) until we have labeled every connected node. Any nodes that are not connected to the $K$-core are unlabeled. Each unlabeled node is put into its own single-node community.

In the second stage, we optimize the community structure for the whole graph. Although it can be fast by repeatedly adjusting the labels of nodes according to their most frequent neighboring labels using label propagation [53], we propose a local modularity optimization algorithm to produce a better quality. This method updates the labels according to the modularity gain by the following analysis.

Modularity is widely used to evaluate the quality of community structures identified by algorithms [30, 33, 44]. The definitions are as follows. First, we use $\delta$ to represent the Kronecker delta function, which returns one only if its two inputs are identical and returns zero otherwise. Then, we use the term degree to represent the number of edges connected to each node. The total degree for a set of nodes is the
summation of degrees of these nodes. Let $A$ be the adjacency matrix of $G$ and $m$ be the number of edges. Thus, $2m$ is the total degree for nodes in $G$. We define

$$e_r = \frac{1}{2m} \sum_{ij} A_{ij} \delta(g(i), r) \delta(g(j), r)$$

to be the fraction of degrees caused by edges that join vertices within community $r$. As we expect dense connections in each community, the modularity definition contains the summation of $e_r$. On the other hand, we also define

$$a_r = \frac{1}{2m} \sum_{ij} A_{ij} \delta(g(j), r)$$

to be the fraction of the degree summation for vertices in community $r$. If nodes are randomly selected but their degree summation is $a_r$, the ratio of edges inside the community is expected to be $a_r^2$. The modularity is defined to subtract the random term to avoid the trivial solution that all the nodes are in one community. Finally, we arrive at the modularity definition

$$Q = \sum_r (e_r - a_r^2). \quad (3.1)$$

Therefore, when a node $i$ is moved from community $r$ to community $s$ ($r \neq s$), the change of modularity is as follows.

$$\Delta Q = Q^{(2)} - Q^{(1)}$$

$$= \sum_i [e_i^{(2)} - (a_i^{(2)})^2] - \sum_i [e_i^{(1)} - (a_i^{(1)})^2]$$

$$= [(e_r^{(1)} - d_r) - (a_r^{(1)} - d) + [(e_s^{(1)} + d_s) - (a_s^{(1)} + d)]$$

$$- [e_r^{(1)} - (a_r^{(1)})^2] - [e_s^{(1)} - (a_s^{(1)})^2]$$

$$= (-d_r + d_s) + 2d(a_r^{(1)} - a_s^{(1)}) - 2d^2. \quad (3.2)$$
where variables with superscript \(^{(1)}\) or \(^{(2)}\) indicate the corresponding values before or after node moving respectively,

\[
d_r = \frac{1}{2m} \left[ \sum_{j \neq i} (A_{ij} + A_{ji})\delta(g(j), r) + A_{ii} \right],
\]

and

\[
d = \frac{1}{2m} \sum_j A_{ij}.
\]

For undirected graphs without self-loops,

\[
d_r = \frac{1}{m} \sum_j A_{ij}\delta(g(j), r).
\]

As \(m\) is the total number of edges in the graph, \(md_r\) is the number of edges in community \(r\) that are connected with node \(i\) and \(2md\) is the total number of edges connecting to node \(i\).

In local modularity optimization, we traverse all the nodes for multiple rounds, and compute the potential \(\Delta Q\) for each node over all the possible community labels. If the maximum potential \(\Delta Q\) for a node is larger than zero, we change the community label of that node correspondingly; otherwise, the label of the node remains unchanged.

There is somewhat of a mix of objectives in the different steps of our method. In Step 2, the community detection optimizes according to the selected method. But in Step 3, the recovery first uses label propagation as a criterion and then switches to optimizing modularity. It is certainly possible to make all the steps consistent, but we have selected very inexpensive heuristics for Step 3. Perhaps surprisingly, our experiments show that quality is not negatively impacted by this mixture of objective functions. It may be due to the fact that they are different ways of measuring the same fundamental structure.
3.1.4 Time Complexity

The running time of the first and the third steps in Algorithm 1 grows linearly with respect to the number of nodes in the graph. Let $b$ be a constant, $N$ be the number of nodes in the graph, and $N_K$ be the number of nodes in the corresponding $K$-core subgraph. If a community detection algorithm has time complexity $O(N^b)$ to find the community structure, the second step of our framework costs only $O(N^b_k)$. Thus, the complexity of Algorithm 1 is $O(N^b_k + N)$. Therefore, when $b > 1$, the speedup is

$$\left(\frac{N}{N_K}\right)^b$$

asymptotically, from which we can see that for community detection algorithms with high time complexity (e.g. $O(N^b)$ with $b = 2$), our framework can reduce the running time significantly.

3.2 Theoretical Analysis

In this section we will discuss the theoretical basis of our proposed method. The essence of our approach is that the dense communities are preserved after a $K$-core reduction, while the size of the graph reduces drastically. Subsequently, we can identify the same communities with much less effort.

3.2.1 Node Remaining Ratios in $K$-core

In this subsection, we define the remaining ratio $R$ as the ratio of the nodes in the $K$-core to all the nodes in the graph, and analyze the relationship between $R$ and $K$.

First, we define several variables: $K$ is the parameter for the $K$-core subgraph; $N$ is the total number of nodes; $E$ is the number of edges; $N_K$ and $E_K$ represent the number of nodes and edges in $K$-core; $\mathcal{V}$ is the set of nodes in a community, and $\mathcal{V}^{(i)}$
is the set of remaining nodes after the $t^{th}$ iteration of the while loop in Algorithm 2; and $y(t)(k)$ is the probability of being degree $k$ for remaining nodes after the $t^{th}$ removal. Hence, $t = 0$ indicates the status of the original graph, without any removal. $k_{\text{max}}$ is the maximum possible degree of any node in $\mathcal{V}$. $R(t)$ is the remaining ratio of nodes after $t^{th}$ removal, and $R(0) = 1$. The remaining degrees are the total degree of the nodes in $K$-core while discarding all the other nodes.

We also use the correlation coefficient to indicate the mixing pattern in connected pairs of nodes [109]. If it is positive, high degree nodes tend to connect to other high degree nodes, and if negative, high degree nodes prefers to connect low degree nodes. When it is zero, the graph is an uncorrelated network such that the node degrees at the end points of any edge are completely independent [110].

**Theorem 1.** For an uncorrelated graph $G$, the ratio of remaining degrees and nodes in $K$-core $G_K$ to those in the original graph $G$ (represented by $E_K$ and $N_K$ respectively) depends only on the degree distribution of the graph $G$.

**Proof.** As defined, $y(0)(k)$ should be the degree distribution of the original graph. In each removal, we define the remaining degree $D_{rn}$ introduced by edges between the remaining nodes after this round removal, and the removed degree $D_{rm}$ which is due to the edges between the remaining nodes and the removed nodes by current removal.

In the $t + 1^{st}$ removal ($t \geq 0$), the total number of remaining degrees is

$$D_{rn}^{(t+1)} = \sum_{k=K}^{k_{\text{max}}} [k \times y(t)(k)] \times N \times \frac{\sum_{k=K}^{k_{\text{max}}} [k \times y(t)(k)]}{\sum_{k=0}^{k_{\text{max}}} [k \times y(t)(k)]} \quad (3.3)$$

where $\sum_{k=K}^{k_{\text{max}}} y(t)(k)$ represents the expected ratio of neighboring nodes inside $\mathcal{V}^{(t+1)}$.

The total number of removed degrees (the degree reduction due to node removals) is

$$D_{rm}^{(t+1)} = \sum_{k=K}^{k_{\text{max}}} [k \times y(t)(k)] \times N \times \frac{\sum_{k=0}^{K-1} [k \times y(t)(k)]}{\sum_{k=0}^{k_{\text{max}}} [k \times y(t)(k)]} \quad (3.4)$$
Then, for nodes in $V^{(t+1)}$, the ratio of remaining degree on average after the $t+1$st removal is

$$r^{(t+1)} = \frac{D_{rn}^{(t+1)}}{D_{rn}^{(t+1)} + D_{rm}^{(t+1)}} = \frac{\sum_{k=K}^{k_{max}} [k \times y^{(t)}(k)]}{\sum_{k=0}^{k_{max}} [k \times y^{(t)}(k)]}$$  \quad (3.5)$$

For a node of degree $k_c$, if edges are randomly kept at probability $r$, then the probability of having $k$ remaining edges is

$$z^{(t+1)}(k, k_c) = f(k, k_c, r^{(t+1)}) = \binom{k_c}{k} (r^{(t+1)})^k (1 - r^{(t+1)})^{k_c-k}$$  \quad (3.6)$$

where $f(k, k_c, r^{(t+1)})$ is the probability of having $k$ edges successfully kept within a total of $k_c$ edges. Function $f$ is essentially a probability density function of a binomial distribution, and will be used several times in the remaining part of this section.

Therefore, the expected degree distribution after one removal is

$$y^{(t+1)}(k) = \sum_{k_c=K}^{k_{max}} z^{(t+1)}(k, k_c) y^{(t)}(k_c)$$  \quad (3.7)$$

Finally, we have the expected remaining ratio after $t+1$th removal.

$$R^{(t+1)} = R^{(t)} \times \sum_{k=K}^{k_{max}} y^{(t)}(k);$$  \quad (3.8)$$

This procedure can run iteratively as $t$ increases and converges asymptotically as $r$ reduces. It is noticeable that $r^{(t+1)}$ in Eq. (3.5) is fully determined by the initial degree distribution $y^{(0)}(k)$, so is $R^{(t+1)}$.

\[\square\]

### 3.2.2 Selecting an Appropriate $K$

We discuss how to select $K$ based on these theoretical analysis. We define $N^{(min)}$ as the number of nodes that belong to the smallest community in the original graph.
The original graph is identical to the 1-core of the graph if there is no isolated node. We also define

\[ R_K = \frac{N_K}{N} \]

as the node remaining ratio in \( K \)-core, and use this to approximate the remaining ratio of nodes in the aforementioned community. As the purpose of using \( K \)-core is to highlight the important nodes and connections, we may assume each node has at least \( K \) neighbors within its own community in the \( K \)-core, so the number of remaining nodes belonging to the aforementioned community in \( K \)-core is at least \( K + 1 \). Then, the following inequality should hold

\[ N^{(\text{min})} R_K \geq K + 1. \]  \hfill (3.9)

In this criterion, \( R_K \) can be obtained from either the theoretical analysis or the empirical result. The best \( K \) can be chosen by selecting the maximum \( K \) satisfying this criterion.

### 3.2.3 Maximizing the Modularity in \( K \)-core

Edges can be either an intra-cluster connection or an inter-cluster connection. We define mixing parameter \( \mu \) as the probability that a randomly selected edge is an intra-cluster edge. If \( \mu \) is large, the connection between different communities becomes stronger. We use \( G_K \) as the \( K \)-core of the original graph. \( g_K \) is a vector representing community labels that can maximize the modularity for nodes in the \( K \)-core.

**Proposition 2.** For an uncorrelated network, we assume the mixing parameter \( \mu \) and the node degree distribution are identical in all the communities. If the modularity for \( G_1 \) is maximized by choosing community labels represented by \( g_1 \) for all the nodes,
then the modularity for $G_K$ is also maximized by choosing the corresponding part of $g_1$, represented by $g_K$, for nodes in the $K$-core.

Proof. Let $\rho$ be the remaining ratio of edges in $G_K$. Because each cluster has the same degree distribution and correlation coefficient, when $K$ is given, $\rho$ is invariant in different clusters.

Using the definition of $m$, $e_r$, and $a_r$ in Section 3.1.3, we define $E_r = 2me_r$ as total degrees introduced by the edges in cluster $r$ and $A_r = 2ma_r$ as the total degrees introduced by all the edges connecting to nodes in cluster $r$. Thus, the modularity can be rewritten as

\[
\text{mod} (G_1, g_1) = \sum_r \left[ \frac{E_r}{2m} - \left( \frac{A_r}{2m} \right)^2 \right],
\]

\[
\text{mod} (G_K, g_K) = \sum_r \left[ \frac{\rho E_r}{2\rho m} - \left( \frac{\rho A_r}{2\rho m} \right)^2 \right] = \text{mod} (G_1, g_1).
\] (3.10)

Thus, using the same configuration, the modularity in $K$-core is the same as in the $1$-core. As $g_1$ maximizes the modularity for Graph $G_1$, any other configuration $g'$ can be considered as a series of relabeling certain nodes from one cluster to the another based on $g_1$, which will reduce the modularity. Without loss of generality, we demonstrate that relabeling $n_\delta$ nodes from their ground truth cluster ‘Cluster 1’ to a new cluster ‘Cluster 2’ will reduce the modularity. $A_\delta$ is the corresponding number of degrees in $K$-core associated with those nodes.

After relabeling, the change of intra-cluster degrees is

\[
- \frac{\mu A_\delta \rho A_1 - A_\delta}{2\rho m} + \frac{(1 - \mu) A_\delta \rho A_2}{2\rho m},
\] (3.11)

where $\mu A_\delta$ is the intra-cluster degree introduced by edges connecting to the relabeling nodes in ground truth Cluster 1, and the first term means the loss is proportional to the remaining degrees in Cluster 1. The second term means the degree increase is
proportional to the degrees in Cluster 2. On the other hand, the change caused by

\[- \left( \frac{\rho A_r}{2\rho m} \right)^2\]

in Eq. (3.10) is

\[- \left[ \left( \frac{\rho A_1 - A_\delta}{2\rho m} \right)^2 + \left( \frac{\rho A_2 + A_\delta}{2\rho m} \right)^2 \right] + \left[ \left( \frac{\rho A_1}{2\rho m} \right)^2 + \left( \frac{\rho A_2}{2\rho m} \right)^2 \right].\]

\[= \frac{A_\delta}{2\rho m} \left[ \frac{\rho A_1 - \rho A_2 - A_\delta}{\rho m} \right].\]  

Therefore, the total change is the summation of Eq. (3.11) and (3.12)

\[
\text{mod} \ (G_K, g'_K) - \text{mod} \ (G_K, g_K) \\
= \frac{A_\delta}{2\rho m} \left[ \frac{\rho A_1 - A_\delta}{\rho A_1} + (1 - \mu) \frac{A_2}{\sum_r A_r} + \frac{\rho A_1 - \rho A_2 - A_\delta}{\rho m} \right] \\
= \frac{A_\delta}{2\rho m} \left[ \frac{A_1 - A_\delta}{A_1} + (1 - \mu) \frac{A_2}{\sum_r A_r} + \frac{A_1 - A_2 - A_\delta}{A_1} \right] \\
= \text{mod} \ (G_1, g'_1) - \text{mod} \ (G_1, g_1) \\
\leq 0
\]

where \(g'_1\) corresponding to the configuration of changing labels of node with \(\frac{A_\delta}{\rho}\) edges from Cluster 1 to Cluster 2.

Thus, the proposition is proved. Therefore, if the Step 2 algorithm can maximize the modularity, the computed community for \(K\)-core is the same as the ground truth. As our Step 3 algorithm also maximizes the modularity for the remaining nodes, the ground truth community structure of the whole graph can be found by our framework.
3.3 Experiments

In this section, we adopt several traditional community detection algorithms into our $K$-core framework, and run them on multiple graphs. Our $K$-core framework can reduce the running time significantly without undermining the quality of the result. By focusing on the $K$-core, our approach also reduces memory usage of the main algorithm. Those features allow it to run on larger graphs.

3.3.1 Data

The real world graphs listed in Table 3.1 are from the Stanford Large Network Dataset Collection at SNAP\textsuperscript{1}. These graphs have been used for evaluating community detection algorithms by many researchers [55, 111, 112, 113]. A brief description about the networks are as follows [114]. Nodes in com-Youtube are users on Youtube.com, and edges indicate the friendships between the users. Nodes in com-Amazon describes the products in Amazon.com, and each edge between two nodes indicates the two products are frequently co-purchased. In Email-Enron network, nodes are email addresses, and an edge between two nodes indicates at least one email is sent from one address to the other. Networks with name prefix ca- are collaboration networks among authors from the e-print arXiv. Each collaboration network covers a subject category in arXiv. Oregon-1 networks contain peering information between autonomous systems inferred from Oregon route-views in different time periods. Peering is an interaction to exchange information.

3.3.2 Implementation Details

All the results in this section are produced in MATLAB R2013b using a 2.66GHz CPU. The results of timing and quality are averaged from five runs.

\textsuperscript{1}SNAP: Stanford Network Analysis Platform, http://snap.stanford.edu/
Table 3.1: Data sets

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>Number of Nodes</th>
<th>Number of Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-Youtube</td>
<td>1,134,890</td>
<td>2,987,624</td>
</tr>
<tr>
<td>com-Amazon</td>
<td>334,863</td>
<td>925,872</td>
</tr>
<tr>
<td>Email-Enron</td>
<td>36,692</td>
<td>367,662</td>
</tr>
<tr>
<td>ca-CondMat</td>
<td>23,133</td>
<td>186,936</td>
</tr>
<tr>
<td>ca-GrQc</td>
<td>5,242</td>
<td>28,980</td>
</tr>
<tr>
<td>ca-HepPh</td>
<td>12,008</td>
<td>237,010</td>
</tr>
<tr>
<td>ca-HepTh</td>
<td>9,877</td>
<td>51,971</td>
</tr>
<tr>
<td>oregon1_010407</td>
<td>10,729</td>
<td>21,999</td>
</tr>
<tr>
<td>oregon1_010421</td>
<td>10,859</td>
<td>22,747</td>
</tr>
<tr>
<td>oregon1_010428</td>
<td>10,886</td>
<td>22,493</td>
</tr>
</tbody>
</table>

To demonstrate the effectiveness and flexibility of our framework, we provide results for a variety of community detection algorithms. The details of the methods and implementations is given below. The methods automatically determine the number of communities except for the last two as noted.

- **Louvain method** [44]: Initializes each node as a community, and shifts the community label of each node according to the modularity gain, until the labels converge. Then, it considers each community as a node to merge some of them again according to the modularity gain. We use the MATLAB implementation called `cluster_jl` by A. Scherrer with default settings\(^2\).

- **Martelot’s method** [45]: A multi-scale algorithm with two phases, optimizing a global and a local criterion respectively. In our experiment, we use the modularity criterion. We use the MATLAB implementation called `fast_mo` by E. Le Martelot\(^3\).

- **Modularity optimization** [47]: An agglomerative method that merges nodes into bigger and bigger communities hierarchically, using the modularity criterion. We use the MATLAB implementation called `fast_newman` by E. Le Martelot\(^3\).

\(^2\)http://perso.uclouvain.be/vincent.blondel/research/Community_BGLL_Matlab.zip

\(^3\)http://www.elemartelot.org/index.php/programming/cd-code
- **Label propagation** [53]: This is a fast method that simply updates the label of a node according to the plurality vote of its neighbors (according to their labels, randomly breaking ties). It runs until the labels cease changing (or 1000 iterations). It is appropriate for large networks, though the quality is usually compromised. Also, the update order impacts the results, so in the experiments run for five times using random orders, and the results are averaged. We implemented this algorithm in MATLAB ourselves.

- **Bayesian method** [42]: A variational approach solves the parameter inference problem in a stochastic block model. We use the MATLAB implementation called vbmod_restart by J. Hofman⁴. In this case, the possible number of communities must be specified by the user, which is a disadvantage but not relevant for our discussion here. We use the results of the prior experiments to come up with reasonable starting values⁵. We set opt.NUM_RESTARTS to be one (default is ten) in order for the code to run relatively quickly.

- **Spectral clustering** [46]: Considers the graph as a similarity matrix, and solves a data clustering problem where each cluster is a community. It contains two phases. First, it maps the data onto a lower dimensional space formed by eigenvectors of the Laplacian matrix; second, it uses k-means to cluster the reduced data. We use the MATLAB implementation SpectralClustering by I. Buerk⁶ with the Jordan/Weiss normalization scheme (type=3), except we have modified it to run the K-means step ten times and choose the solution that minimizes the sum of the within-cluster sums of point-to-centroid distances. The user needs to specify an appropriate number of communities in advance and we use the same values as for the the Bayesian method⁵.

⁴http://vbmod.sourceforge.net/
⁵Specified number of communities: Email-Eron=485, ego-Facebook=10, ca-AstroPh=604, ca-CondMat=1835, ca-GrQc=391, ca-HepPh=608, ca-HepTh=904, oregon1 XXXXXX=30.
⁶http://www.mathworks.com/matlabcentral/fileexchange/34412-fast-and-efficient\ -spectral-clustering/content/files/SpectralClustering.m
In the recovery step of our algorithm (Step 3), the maximum number of iterations for labeling nodes is set to ten and the maximum number of modularity optimization iterations is set to 10.

### 3.3.3 Results and Analysis — Large-scale Networks

Fig. 3.1 demonstrates how our $K$-core framework accelerates the (linear-time) label propagation algorithm [53] on com-Youtube and com-Amazon data. Since label propagation is very fast, we skip the second stage algorithm in Step (3) of our framework.
Table 3.2: Algorithms using $K$-core on com-Amazon network

<table>
<thead>
<tr>
<th>Algorithm Name</th>
<th>$K$</th>
<th>Modularity</th>
<th>Time (Relative to 3-core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Martelot’s method [45]</td>
<td>3</td>
<td>0.923</td>
<td>6.3e4 (1.00)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.917</td>
<td>2.6e4 (0.41)</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>0.890</td>
<td>3.2e3 (0.05)</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>0.788</td>
<td>5.5e2 (0.01)</td>
</tr>
</tbody>
</table>

which uses the similar idea but optimizes the modularity. Any unlabeled nodes are assigned to singleton communities.

The plots on the left of Fig. 3.1 show that the quality (based on modularity) is unchanged as $K$ increases. The plots on the right of Fig. 3.1 show details on the method. In these cases, the label propagation algorithm used in Step 2 is so inexpensive that the cost of Step 3 (even without the modularity optimization) eventually dominates the total time. Nevertheless, the overall run time is significantly reduced even when the original algorithm is only linear!

Louvain method [44] is also a linear time algorithm, but the average running time for each node is much longer than label propagation. Thus, we only run those algorithms with $K \geq 3$ on the com-Amazon network. By Fig. 3.1, the ratio of remaining nodes drops from 76% in 3-core to 50% in 4-core, 17% in 5-core, and 0.2% in 6-core. In Table 3.2, the running time column contains the running time in seconds, and those in the bracket behind are the relative running times with respect to 3-core. We show that by our approach, the modularity computed through both algorithms does not change much from 3-core to 5-core, but the running times have significant reduction. At 6-core, because the remaining nodes are too few, the modularity is no longer satisfactory.

3.3.4 Results and Analysis — Multiple Real-world Networks

In this section, we consider a collection of results on eight more real-world graphs. We use two blue vertical lines to represent the best $K$ selected by our algorithm
Figure 3.2: Relative run time over different $K$’s, Part 1
Figure 3.3: Relative run time over different $K$’s, Part 2
Figure 3.4: Modularity over different $K$’s, Part 1
Figure 3.5: Modularity over different $K$’s, Part 2
Figure 3.6: Normalized cut over different $K$’s, Part 1
Figure 3.7: Normalized cut over different $K$'s, Part 2
with $N^{\text{min}}$ equal to 20 and 100 respectively. In Fig. 3.2-3.7, label propagation [53] is omitted because the quality is usually much worse than other algorithms. Additionally, the Bayesian method [42] and modularity optimization [47] are omitted in two comparisons because they run too slowly and the experiments take more than 24 hours.

In Fig. 3.2 and 3.3, the maximum running time for each algorithm is normalized to one. Generally, the overall run time is substantially reduced. The main exception is that the runtime increases for spectral clustering on the oregon networks (which are quite small and sparse). Otherwise, the running time reduction can be as much as 80% for $K = 2$ and over 95% for larger values of $K$.

In terms of quality, Fig. 3.4 and 3.5 shows the modularity as $K$ increases. Generally, the modularity is little changed and sometimes even improves. For instance, the time to run Modularity Optimization on ca-HepTh is over 1000 seconds on the full graph, but that time reduces by 95% for $K = 4$ with no reduction whatsoever in modularity score.

For another quality measure, we show the normalized cut [85] in Fig. 3.6 and 3.7 (lower is better). Once again, we see almost no degradation in quality even when runtimes are reduced by 50-90% or more.

<table>
<thead>
<tr>
<th>Data Set Name</th>
<th>$K$ (Remaining Ratio)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email-Enron</td>
<td>3 (0.58) 6 (0.25) 9 (0.14)</td>
</tr>
<tr>
<td>ca-CondMat</td>
<td>4 (0.58) 6 (0.35) 8 (0.19)</td>
</tr>
<tr>
<td>ca-GrQc</td>
<td>3 (0.50) 4 (0.30) 5 (0.17)</td>
</tr>
<tr>
<td>ca-HepPh</td>
<td>5 (0.45) 10 (0.23) 20 (0.16)</td>
</tr>
<tr>
<td>ca-HepTh</td>
<td>3 (0.52) 4 (0.32) 5 (0.21)</td>
</tr>
<tr>
<td>oregon1_010407</td>
<td>2 (0.65) 3 (0.20) 4 (0.08)</td>
</tr>
<tr>
<td>oregon1_010421</td>
<td>2 (0.65) 3 (0.21) 4 (0.09)</td>
</tr>
<tr>
<td>oregon1_010428</td>
<td>2 (0.65) 3 (0.20) 4 (0.08)</td>
</tr>
</tbody>
</table>

Table 3.3 lists the ratio of remaining nodes in $K$-core. With Fig. 3.4 and 3.5, it demonstrates empirically that if the remaining ratio in $K$-core is above 50%, the
modularity obtained by our framework is almost invariant, and if the remaining ratio is above 20%, the modularity decreases slightly. In all those cases, the running time reduces significantly as the number of remaining nodes decreases.

3.4 Discussion

In this chapter, we propose a framework that can reduce the running time of community detection algorithms significantly. This is achieved by assigning community labels of peripheral nodes outside the $K$-core in relatively little time proportional to the number of nodes.

However, when the graph is extremely large, 20% of nodes in $K$-core can still be a lot and some algorithms can still suffer from high time complexity in $K$-core. In the next chapter, we propose a linear time algorithm using stochastic block models to find community structures. It can execute on parallel systems to obtain more speedup.
Chapter 4

Increase the Scalability

4.1 Stochastic Block Model

In this work, we develop a community detection algorithm based on stochastic block models. We define $N$ as the number of nodes and $M$ as the number of undirected and unweighted edges connecting those nodes. Each node belongs to one of $K$ blocks, and we use $Z \in \{0,1\}^{N \times K}$ to represent the block labels. $Z_{ir} = 1$ means node $i$ belongs to block $r$ and each row of $Z$ contains only one nonzero entry. We also define a matrix $B \in [0,1]^{K \times K}$ where $B_{r,k} (r \neq k)$ represents the probability of connections between nodes drawn from block $r$ and $k$, respectively. If $r = k$, $B_{r,k}$ represents the probability of connections inside the block.

Using the matrices $B$ and $Z$, we define a probability matrix $\Theta = ZBZ^T$. Then the adjacency matrix $W$ of a sample network can be drawn from the model by

$$
\Pr(W_{ij}) = \begin{cases} 
\Theta_{ij} & \text{if } W_{ij} = 1, \\
1 - \Theta_{ij} & \text{if } W_{ij} = 0,
\end{cases}
$$

(4.1)

for $i, j \in \{1, 2, \cdots, N\}$ and $i \neq j$. Typically, the adjacency matrix $W$ is available from the data set. Our primary purpose is to estimate $Z$. 
4.2 Methodology

As defined in Eq. (4.1), if only $W$ is given, the log-likelihood function is

$$L(B, Z|W) = \sum_{i\neq j} \log \Pr(W_{ij})$$

$$= \sum_{i\neq j} \log[(1 - W_{ij}) + (2W_{ij} - 1)\Theta_{ij}].$$

It is time consuming to maximize such a likelihood function directly through traditional optimization methods (for example, branch-and-bound) for large graphs in which there are at least $NK$ unknown variables.

For the sake of speed and scalability, we propose a fast algorithm that updates $B$ and $Z$ in turn to maximize the objective function $L(B, Z|W)$, and use a multi-stage framework to help the solution be close to the global optimum. We also develop a parallel implementation to make the model more scalable.

4.2.1 Estimation Algorithm

Given $W$, the maximum likelihood estimates of $(B, Z)$ are defined as

$$\arg\max_{B,Z} \left\{ L(B, Z|W) \right\}$$

$$= \sum_{i\neq j} \log[(1 - W_{ij}) + (2W_{ij} - 1)\Theta_{ij}]$$

$$= \sum_{i\neq j} \log[(1 - W_{ij}) + (2W_{ij} - 1)(ZBZ^T)_{ij}], \quad (4.2)$$

subject to $0 \leq B_{ij} \leq 1$, $Z_{ij} \in \{0, 1\}$, $\sum_j Z_{ij} = 1$. Roughly speaking, we solve the above optimization problem by alternatively updating $B$ and $Z$ and using a cluster shrinking and expanding strategy to improve accuracy.

We first describe the alternating updating procedure. When $Z$ is fixed and $B$
is considered as unknown, without loss of generality, let $\beta = B_{rk}$. If other entries are fixed, $L(B, Z|W) = s \log(\beta) + \hat{s} \log(1 - \beta) + C$, where $C$ is a constant, $s = \sum_{ij} W_{ij}(Z_r Z_{ik})_{ij}$ and $\hat{s} = \sum_{ij} (1 - W_{ij})(Z_r Z_{ik})_{ij}$. Taking the derivative of $L$ with respect to $\beta$

$$
\frac{\partial L}{\partial \beta} = \frac{s}{\beta} - \frac{\hat{s}}{1 - \beta}
$$

and setting the derivative to be zero, we have

$$
\beta = \frac{s}{s + \hat{s}}.
$$

As the inter-block connection probabilities are small, we use a representative scalar value to replace the off-diagonal entries of $B$, which can be computed by counting all the inter-cluster edges. Thus, the total time complexity of updating $B$ is $O(N) + O(K) + O(M) = O(M)$.

**Theorem 3.** For fixed $Z$, the objective function $L(B, Z|W)$ achieves its global maximum if entries of $B$ are updated according to Eq. (4.4).

**Proof.** When $Z$ is fixed, because each entry of $B$ can optimize the objective function independently, after updating all the entries by Eq. (4.4), the resulting $B$ is a stationary point of the objective function and each entry satisfies the constraints.

By taking the second derivative of the objective function, we have

$$
\frac{\partial^2 L}{\partial \beta^2} \bigg|_{\beta=\frac{s}{s+\hat{s}}} = -\frac{s}{\beta^2} - \frac{\hat{s}}{(1-\beta)^2} \bigg|_{\beta=\frac{s}{s+\hat{s}}} < 0.
$$

Therefore, when $Z$ is given, the objective function at $\beta$ (determined by Eq. (4.4)) is a global maximum. As the entries of $B$ are independent of each other, we can update $B$ by Eq. (4.4).

When $B$ is fixed, we use the block coordinate descent method to update $Z$ row by row. When updating the first $Z$ in $ZBZ^T$ in Eq. (4.2), the algorithm keeps the
second \( Z \) as its previous estimate \( Z^{(t-1)} \). Then, the likelihood function can be locally maximized by setting all the elements in the row to 0 but \( Z_{ir_{\text{max}}} = 1 \), where \( r_{\text{max}} \) is chosen by

\[
r_{\text{max}} = \arg\max_r \sum_{j \neq i} \log [(1 - W_{ij}) + (2W_{ij} - 1)(B[Z^{(t-1)}]^T)_{rj}].
\] (4.6)

The time complexity for updating one node by Eq. (4.6) is \( O(NK) \) and for all the nodes it is \( O(N^2K) \).

To reduce the time complexity, we propose a faster method. Let \( N^{(c)} \) be a column vector containing the number of nodes in each cluster, which is updated once the row \( Z_i \) is changed. Let \( N^{(d)} \) be a column vector containing the number of nodes connected to node \( i \) in each cluster. Thus, we have a new update rule.

\[
r_{\text{max}} = \arg\max_r \sum_{k=1}^{K} [N_k^{(d)} \log(B_{rk}) + (N_k^{(c)} - N_k^{(d)}) \log(1 - B_{rk})].
\] (4.7)

The time complexity is \( O(M_i) + O(K^2) \), where \( M_i \) is the degree of node \( i \) used in computing \( N^{(d)} \). If we use a scalar to represent the inter-block connection probabilities as before, we can replace the summation in Eq. (4.7) by two multiplications. We also note that a node will take only a block label from its neighbors, and the time to enumerate over different choices of \( r \) becomes \( O(M_i) \) for node \( i \) in Eq. (4.7). Thus, the time complexity of computing labels for all the node is reduced to \( \sum_{i=1}^{N} O(M_i) = O(M) \).

For fixed \( B \), we define \( G(B, Z^{(2)}, Z^{(1)}) = \sum_{i \neq j} \log[(1-W_{ij}) + (2W_{ij} - 1)(Z^{(2)}BZ^{(1)}^T)_{ij}] \). Therefore, the objective function can be rewritten as \( L(B, Z|W) := G(B, Z, Z) \).

If entries in \( Z \) are continuous variables, then the above optimization can find a
global maximum.

**Theorem 4.** $\min_{Z(t)} -G(B, Z(t), Z^{(t-1)})$ under the constraints $\sum_r Z_{ir} = 1$ and $0 \leq Z_{ir} \leq 1$ is a convex optimization problem when $B$ and $Z^{(t-1)}$ are given.

**Proof.** First we reshape the matrix $Z$ into a column vector $z$ where $Z_{ij}$ corresponds to $z_{K(i-1)+j}$.

Let $H \in \mathbb{R}^{NK \times NK}$ be the Hessian matrix of the objective function $G(B, Z(t), Z^{(t-1)})$ with respect to $z^{(t)}$. Let $M = \log[(1 - W) + (2W - 1) \odot Z^{(t)} BZ^{(t-1)^T}]$, which is the matrix in $G(B, Z(t), Z^{(t-1)})$ before the final summation, and we only need to consider the off-diagonal part of the matrix in this prove. Here $\odot$ denotes the element-wise product of two matrices.

If $[Z_i^{(t-1)}]^T$ are constants, the second derivative of the $(i, j)^{th}$ term $(i \neq j)$ in $M$ with respect to variables other than $[Z_i^{(t)}]^T$ is 0, while with respect to the variables in $[Z_i^{(t)}]^T$, it is as follows

\[
\begin{align*}
\frac{\partial^2 \{-M_{ij}\}}{(\partial [Z_i^{(t)}]^T)^2} &= \frac{\partial^2 \{-\log[(1 - W) + (2W - 1)Z_i^{(t)} B[Z_j^{(t-1)}]^T]\}}{(\partial [Z_j^{(t)}]^T)^2} \\
&= \frac{\partial \left\{-\frac{(2W_{ij} - 1)B[Z_j^{(t-1)}]^T}{(1-W_{ij}) + (2W_{ij} - 1)Z_i^{(t)} B[Z_j^{(t-1)}]^T}\right\}}{\partial [Z_i^{(t)}]^T} \\
&= \frac{\partial [Z_i^{(t)}]^T}{\left((2W_{ij} - 1)^2 B[Z_j^{(t-1)}]^T[Z_j^{(t-1)}]^T B^T\right)} \\
&= \frac{\left((1 - W_{ij}) + (2W_{ij} - 1)Z_i^{(t)} B[Z_j^{(t-1)}]^T\right)^2}{\left((1 - W_{ij}) + (2W_{ij} - 1)Z_i^{(t)} B[Z_j^{(t-1)}]^T\right)^2},
\end{align*}
\]

where the square operation is element-wise. It is positive semi-definite because it is a covariance matrix multiplied with a positive scalar.

Therefore, for any column vector $x$, we have
\[
-x^T H x \\
= -x^T \frac{\partial^2 \{ \sum_{i \neq j} M_{ij} \}}{\partial z^2} x \\
= \sum_{i \neq j} x^T \frac{\partial^2 \{- M_{ij}\}}{\partial z^2} x \\
\geq 0. \quad (4.8)
\]

Hence, the objective function is convex. The constraints are convex as well, this problem is a convex optimization. □

However, in our model, \( Z \) is a Boolean matrix. Thus, it will probably converge to a local optimum, in which nodes from several true clusters may become one single temporal cluster. Here we refer to the cluster that a subset of nodes indeed belongs to as the \textit{true cluster} for those nodes, and the cluster determined by an algorithm at each iteration is called the \textit{temporal cluster}.

This problem can be avoided by limiting each temporal cluster to contain only one true cluster in the initialization. The next subsection describes the detail.

**Community Shrinking and Expanding**

A community shrinking and expanding approach works as follows. First, randomly initialize the nodes into \( \alpha K \) temporary communities where \( \alpha \) should be large enough so that the community size is tiny. Second, run the inference algorithm while gradually merge communities and reduce the number of communities to \( K \). When \( K \) is unknown, the algorithm proceeds until no more merging is possible.

This approach reduces the “collision” probability in the initialization significantly, where a “collision” is the situation that two true communities both have most of the nodes in one temporary community. The ratio of the “non-collision” probability after
shrinking to the probability without shrinking is:

\[
\prod_{k=0}^{K-1} \frac{1 - \frac{k}{\alpha K}}{1 - \frac{k}{K}} = \prod_{k=0}^{K-1} \left[ 1 + \frac{(1 - \frac{1}{\alpha}) k}{K - k} \right] \geq \left( 1 - \frac{1}{\alpha} \right) \frac{K^K}{K!}.
\]

The lower bound is not sensitive to the choice of \(\alpha\) when \(\alpha\) is large enough. For example, the lower bounds at \(\alpha = 10\) and \(\alpha = 100\) are

\[
\frac{0.9K^K}{K!}
\]

and

\[
\frac{0.99K^K}{K!},
\]

respectively, both of which are very significant. In practice, we can choose \(\alpha K = N\).

Initializing an extra number of communities may introduce some tiny communities in the final result. The community expanding is devised by considering the likelihood of the inter-community edges. When they are more likely to be in certain communities, the corresponding nodes are merged. In detail, for any two communities \(r\) and \(k\), the number of edges between them is

\[
c_{rk} = \sum_{i \neq j} Z_{ir} W_{ij} Z_{jk},
\]

which is out of \(n_{rk} = \sum_i Z_{ir} \times \sum_i Z_{ik}\), the maximum possible connections. Therefore, the log likelihood of the inter-community edges \(c_{rk}\) belonging to the true community \(r\) can be represented by \(L_r\), while the likelihood of not belonging to the community
is \( L_q \).

\[
L_p = c_{rk} \log(p) + (n_{rk} - c_{rk}) \log(1 - p),
\]

\[
L_q = c_{rk} \log(q) + (n_{rk} - c_{rk}) \log(1 - q),
\]

(4.9)

(4.10)

where \( p = (c_{rr} + c_{rk} + c_{kr} + c_{kk})/(n_{rr} + n_{rk} + n_{kr} + n_{kk}) \) indicating the internal density after merging, and \( q = B_{rk} \). If \( L_p > L_q \), we merge \( r \) and \( k \) into one community, otherwise leave them separated.

The time complexity for each merging is \( O(K^2) \). However, if we only consider the pairs of communities that have at least one edge between them, the time complexity becomes \( O(M) \). Community merging runs at the end of each stage after updating \( Z \) and \( B \). Algorithm 4 is the pseudocode of the serial algorithm.

---

**Algorithm 4** Serial Algorithm (ML-SBM)

set community number to \( \alpha K \)

initialize \( B \), \( Z^{(t)} \) into many tiny communities

\( Z^{(t-1)} = Z^{(t)} \)

repeat

repeat

for \( i = 1 : N \) do

compute \( N^{(d)} \) for node \( i \)

update \( Z^{(t)}_{\hat{i}} \) according to Eq. (4.7)

\( Z^{(t-1)}_{\hat{i}} = Z^{(t)}_{\hat{i}} \)

update \( N^{(c)} \)

end for

update \( B \) according to Eq. (4.4)

until \( Z^{(t)} \) does not change anymore

merge communities according to Eq. (4.9) and (4.10)

until no more merging is possible

---

### 4.2.2 Parallelism

Our algorithm uses the message-passing programming model that can theoretically utilize all the processors on shared, distributed, or hybrid memory systems. The par-
allelism is designed as follows. Let $S_I$ be the set of integers from 1 to $N$, partitioned into non-overlapping and approximately equal-size subsets, each of which is represented by $S_{Ib}$ for the $b^{th}$ processor. For each processor, we also use local variables $s_b$, $\hat{s}_b$ and $\beta_b$, which have the similar definitions as $s$, $\hat{s}$ and $\beta$, but only taking the $i^{th}$ row ($i \in S_{Ib}$) of $W$ into account.

We use non-blocking Message Passing Interface (MPI) communications to improve communication-computation overlap. This has two-fold benefits: utilizing the network bandwidth during computation, and maintaining the convergence rate for $Z$ similar to the serial version by transmitting up-to-date community labels. In order to reduce the overall communication data, changes of $Z_i$ are sent to processor $b$ only if $i$ has neighboring nodes in $S_{Ib}$. Because the communication bandwidth will be higher as the message size is larger, we buffer the change information of $Z$ until computing $f$ rows of $Z$ finishes. Here $f$ is chosen to be inversely proportional to the number of MPI ranks, and proportional to $N$, maintaining a consistent convergence rate.

The algorithm uses similar partitioning and communicating strategies when computing $B$. Algorithm 5 presents the pseudocode of the parallel algorithm.

In parallel computing, the rows of $W$ are distributed on different processors. If there are totally $g$ processors (equal to the number of MPI ranks) and data are evenly distributed, the computation time of each processor for Eq. (4.4) and Eq. (4.7) is $O(\frac{M}{g})$. When the computations for $B$ are partitioned on each processor, reducing the number of clusters also requires $O(\frac{M}{g})$ for each processor. The total message length for $Z$ and $B$ during the iterations is $O(g(N + K))$. If the bandwidth is constant, the speedup of the parallel algorithm is

$$\frac{T(M)}{T(\frac{M}{g}) + O(N + K)},$$

where $T$ is a linear function, such that the numerator and the denominator indicate
Algorithm 5 Parallel Algorithm

set community number to $\alpha K$

initialize $B$, $Z(t)$ into many tiny communities

$Z(t-1) = Z(t)$

repeat

repeat

    do in parallel

    for $i \in S_{Ib}$ do

        compute $N^{(d)}$ for node $i$

        update $Z_i^{(t)}$ according to Eq. (4.7)

        $Z_{i; (t-1)} = Z_{i; (t)}$

        synchronize changes of $Z_{i;}$ at frequency $f$

    end for

    update $s_b$ and $\hat{s}_b$ for each $\beta_b$ entries

    synchronize $s = \sum_b s_b$ and $\hat{s} = \sum_b \hat{s}_b$

    update $B$ according to Eq. (4.4)

    update $N^{(c)}$

until $Z(t)$ does not change anymore

merge communities according to Eq. (4.9) and (4.10)

until no more merging is possible

the running time of the serial version and that of the parallel version respectively.

Therefore, the speedup will increase towards $g$ if the number of edges increases given

the same $N$ and $K$.

4.3 Model Analysis: An Ideal Example

In this section, through a simplified example, we illustrate how our algorithm solves

the SBM problem. In particular, we assume the diagonal part of the ground truth

$B$ is $p$, and the off-diagonal part is $q$. Thus, each pair of nodes from one cluster

is connected with probability $p$, and those from two different clusters are connected

with probability $q$. As mentioned earlier, $N$ is the number of nodes in the graph, and

$K$ is the number of the clusters. We define $n$ as the average number of nodes in each

cluster.
4.3.1 Self-Assembling Process

We initialize $Z$ by putting nodes almost evenly and randomly into different temporal clusters, and a bias will arise in large probability such that some temporal cluster should have a larger number of nodes truly from the same true cluster. Namely, we can define

$$p_R^{(0)} \approx \frac{1}{K^2}$$

as the expected ratio of nodes belonging to the same true cluster over the total number of nodes, in a temporal cluster at time 0.

At each iteration, each node will associate with a particular temporal cluster where it has a tight connection. In this case, the temporal cluster with more nodes from the same true cluster will take an advantage because the probability of the intra-cluster connection is higher than other temporal clusters. Due to this bias, the association will result in

$$p_R^{(t)} = p_R^{(t-1)} \frac{1}{K - p_R^{(t-1)}} + p_R^{(t-1)}$$

$$= 2p_R^{(t-1)} - (p_R^{(t-1)})^2 \times K$$

for the ratio of nodes in a temporal cluster and from the same true cluster. As the maximum $p_R^{(t)}$ is $\frac{1}{K}$, the condition $2 > p_R^{(t-1)}K$ is always true. Therefore, this equation indicates that at early stages, nodes from the same true cluster in other temporal clusters will tend to assemble to a unique temporal cluster.

In practice, we generate many more temporal clusters than the number of true clusters at the initialization, to avoid two or more true clusters assembling at the same temporal cluster.
### 4.3.2 Graphs with Heterogeneous Partitions

This subsection is complementary to Section 4.3.1 Self-Assembling Process, where we discuss the case when the cluster sizes are different.

First, we define $N_1, N_2, \ldots, N_{K-1}, N_K$ in ascending order as the size of the $K$ clusters. Without loss of generality, we can assume the $k^{th}$ partition corresponding to $N_k$ will emerge in the $k^{th}$ temporal cluster.

Then, when temporal clusters are initialized to have the same size, nodes from different clusters are assigned to temporal clusters in equal probability. To have the self-assembling process still work, we shall have the following expression, so that nodes from cluster 1 will not assemble to temporal cluster $K$.

$$\frac{qN_K}{K} < \frac{pN_1}{K}, \quad (4.14)$$

where $k \in \{1, 2, \ldots, K\}$. Under the above condition, nodes from other clusters are expected to assemble to correct temporal clusters because of the following inequality.

$$\frac{qN_k}{K} < \frac{qN_K}{K} < \frac{pN_1}{K} < \frac{pN_k}{K}. \quad (4.15)$$

During the self-assembling procedure, similar to Eq. (4.13), we have the following expression for cluster $k$

$$p_R^{(t)} = \frac{p_R^{(t-1)} \times \left( \frac{N_k}{N} - p_R^{(t-1)} \right) + p_R^{(t-1)}}{\frac{N}{N_k}} = \frac{2p_R^{(t-1)} - (p_R^{(t-1)})^2 \times \frac{N}{N_k^{(t-1)}}}{\frac{N}{N_k^{(t-1)}}}.$$

Hence, the convergence rate is still quadratic.

At the end of the iteration, to obtain the correct result, the following expression
is expected

\[ qN_K < pN_1, \quad (4.16) \]

which is in accordance with the assembling condition at the initialization in Eq. (4.14).

For undirected networks, \( B \) is symmetric. When the probability matrix is contains a variety of values, rather than only \( p \) and \( q \), the following inequality is required as a necessary condition for converging to the correct result.

\[ B_{ij}N_i < B_{jj}N_j, \text{ for } i > j. \quad (4.17) \]

4.3.3 Minimum Gap between \( p \) and \( q \)

The gap between \( p \) and \( q \) represents the difference in edge densities within and across clusters. A smaller value of \( p - q \) in SBMs causes more challenges in the identification of the true \( Z \). Thus, a parametric inference algorithm is more powerful if it allows a smaller gap between \( p \) and \( q \). We find that by our algorithm, the lower bound of \( p - q \) can be as small as \( \Omega\left(\sqrt{\frac{K}{N}}\right) \). The derivations are as follows.

Let \( \Pr(Z_{ik} = 1) \) be the probability that node \( i \) of true cluster \( k \) can also be correctly assigned via maximum likelihood. When correctly assigned, the destination temporal cluster should have more nodes connected to \( i \) than any other temporal cluster. We use \( \sum X \) and \( \sum Y \) to represent the number of nodes connected to \( i \) in the destination temporal cluster and some other temporal cluster respectively. \( X \) and \( Y \) are collections of binary random variables. Then we will demonstrate that node \( i \) can be correctly identified at least in certain probability \( p_I \), where \( p_I \) is the probability for correct identification between the true cluster and all the other clusters. Then, the probability of that between the true cluster and one of the other clusters is \( p_I^{\frac{1}{K-1}} \).
Using an auxiliary variable $d$ to represent the cutting point of the two probabilities, we have

$$p_i^{K_{1-1}} \geq \Pr(\sum X > d) \times \Pr(\sum Y < d)$$

$$= [1 - \Pr(\sum X < d)] [1 - \Pr(\sum Y > d)]$$

$$= \left[1 - \Pr(\sum X < pn - (pn - d))\right]$$

$$\times \left[1 - \Pr(\sum Y > qn + (-qn + d))\right]$$

$$\geq \left[1 - e^{-\frac{(pn-d)^2}{2pn}}\right] \left[1 - e^{-\frac{2(-qn+d)^2}{n}}\right]$$

$$= \gamma^{K_{1-1}},$$

(4.18)

where $n$ is the expected number of nodes within the cluster and $\gamma$ is set as a lower bound of probability $p_I$ satisfying the last equality. The second inequality is obtained from Chernoff’s inequality [115] and Hoeffding’s inequality.

By choosing an appropriate $d$, the lower bound of $p_i^{K_{1-1}}$ can be well approximated. If we choose $d$ to have the two factors equal, we obtain the following expressions

$$pn - d = 2\sqrt{p}(d - qn),$$

$$d = \frac{pn + 2\sqrt{p}(qn)}{1 + 2\sqrt{p}},$$

$$pn - d = pn - \frac{pn + 2\sqrt{p}(qn)}{1 + 2\sqrt{p}}$$

$$= \frac{pn + 2\sqrt{p}(pn) - pn - 2\sqrt{p}(qn)}{1 + 2\sqrt{p}}$$

$$= \frac{2\sqrt{p}(pn - qn)}{1 + 2\sqrt{p}}.$$ 

Hence,

$$\frac{(pn-d)^2}{2pn} = \frac{4p(pn-qn)^2}{(1+2\sqrt{p})^2} = \frac{2n(p-q)^2}{(1+2\sqrt{p})^2},$$

(4.19)
which is the exponent term in Eq. (4.18).

As the lower bound of the probability $p_I$, $\gamma$ is expected to be as large as possible. Then we can have $1 - \gamma \leq 1 - c_0 N^{-10}$ for some constant $c_0$ to meet the definition of high probability, which means there always exists a $c_0$ such that the right hand side can be large enough. According to Eq. (4.18), we have

\[
\frac{2n(p - q)^2}{(1 + 2\sqrt{p})^2} = - \log(1 - \gamma^\frac{1}{2^{(K-1)}})
\geq - \log[1 - (c_0 N^{-10})^{\frac{1}{2^{(K-1)}}}] \\
= (c_0 N^{-10})^{\frac{1}{2^{(K-1)}}} + \frac{1}{2} (c_0 N^{-10})^{\frac{2}{2^{(K-1)}}} + \frac{1}{3} (c_0 N^{-10})^{\frac{3}{2^{(K-1)}}} + \ldots \\
= \Omega(N^{-\frac{2}{2^{(K-1)}}}),
\]

where the expansion is based on the Taylor series.

Therefore, as $K$ is typically proportional to $N$ and $n = \frac{N}{K}$, we can have the following approximation

\[
p - q = \sqrt{\frac{(1 + 2\sqrt{p})^2}{2n} \Omega(N^{-\frac{2}{2^{(K-1)}}})} \\
= \sqrt{(1 + 2\sqrt{p})^2 \Omega(N^{-\frac{2}{2^{(K-1)-1}}} | K)} \\
\approx \Omega\left(\frac{pK}{N}\right). \tag{4.20}
\]

Compared to many other methods summarized in [102], whose lower bound is $\Omega\left(\sqrt{\frac{pN \log N}{K}}\right)$, the lower bound by our approach is relatively small.
4.3.4 Convergence Rate

When the sizes of clusters are equal, each cluster has $\frac{1}{K}$ nodes. Let $p_R^{(t-1)} = \frac{1}{K} + \theta$ where $\theta$ is a bias term. By Eq. (4.13), we have the following expression:

$$\lim_{t \to \infty} \frac{|p_R^{(t)} - \frac{1}{K}|}{|p_R^{(t-1)} - \frac{1}{K}|^2} = \lim_{t \to \infty} \frac{|2p_R^{(t-1)} - (p_R^{(t-1)})^2 \times K - \frac{1}{K}|}{|p_R^{(t-1)} - \frac{1}{K}|^2} = \frac{|2(\frac{1}{K} + \theta) - (\frac{1}{K} + \theta)^2 \times K - \frac{1}{K}|}{|\theta|^2} = K,$$

which means that the convergence is quadratic.

4.4 Experimental Results

4.4.1 The Serial Algorithm

In this section, we compare our algorithm (ML) with other serial algorithms in MATLAB. The competitors include the Louvain method (LV) [44], a Bayesian inference algorithm (BI) [42], spectral clustering (SC) [46], label propagation (LP) [53], and modularity optimization (MM) [47]. We use the LFR benchmark generator [116] to create graphs, and most of the parameters are the same as the default setting. Each generated graph is of size 1000, average degree 30, maximum degree 50, exponent of degree distribution $-2$, exponent of community size distribution $-2$, minimum community size 20, and maximum size 100. $\mu$ is the proportion of inter-community connections to the intra-community ones. The accuracy is evaluated by normalized mutual information (NMI) [117], which compares the similarity between the computed community structure and the ground-truth one. The larger the NMI is, the
more similar the two structures are. If two structures are identical, NMI is 1. The results are averaged on five runs.

From Figs. 4.1(a) and 4.1(b) we can see that our algorithm is relatively fast and has the best accuracy. When $\mu = 0$, most of the algorithms can find the correct result, and our algorithm is the second fastest one. As $\mu$ increases to 0.7, our algorithm needs a longer time to iterate, but its accuracy is still the best. Only SC and LV can achieve a similar accuracy over a similar amount of time.

However, for large graphs, SC is not scalable, and LV is not accurate. Our algorithm is the method of choice for both scalability and accuracy. Figs. 4.1(c) and 4.1(d) show the results on graphs similar as the previous experiment but the average degree is 10 and $\mu = 0.1$. On large graphs, compared with other algorithms, ours consistently produces more accurate results with respect to the ground-truth setting, and the running time growth rate is smaller than others as the graph size grows. The slow growth of the running time of our algorithm is in accordance with the theoretical time complexity which is linear in the number of edges.

### 4.4.2 The Parallel Algorithm

In this section, we exploit distributed memory parallelism for larger graphs. When the input graph has more than one million nodes belonging to more than one thousand communities, typical algorithms for stochastic block models are not able to infer the parameters within a reasonable amount of time (for example, 24 hours). Therefore, we only compare our algorithm (PS+number of processors) to a popular community detection algorithm, the Louvain method (LV) [44], and a fast graph partitioning algorithm Metis (MT) [118]. The Louvain method implemented in C can also generate level one structures as byproducts containing the finest communities (LVF). Those communities are merged into larger ones in LV. Parallel algorithms are often extended from serial versions [56], but the results are typically invariant. We use several metrics
Figure 4.1: Comparison to popular algorithms. (a) and (b) represent the variation of average accuracy and running time over different choices of $\mu$. (c) and (d) represent the variation over different choices of graph size $N$.

to compare the results, such as NMI, modularity [30], and conductance (the smaller, the better) [119]. The results on those quantities are averaged on five runs.

First, we run our algorithm on benchmark graphs (LFR-1e6) generated by LFR
Table 4.1: Properties of the test graphs

<table>
<thead>
<tr>
<th>Graph Name</th>
<th>Node Number</th>
<th>Edge Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>LFR-1e6</td>
<td>1,000,000</td>
<td>~5,000,000</td>
</tr>
<tr>
<td>cit-Patents</td>
<td>3,774,768</td>
<td>16,518,948</td>
</tr>
<tr>
<td>dblp-Coauthor</td>
<td>1,314,050</td>
<td>10,724,828</td>
</tr>
</tbody>
</table>

Figure 4.2: Comparison on benchmark data
Figure 4.3: Number of communities versus community size

benchmark [116]. The parameters are similar to the serial case except the graph size is increased to one million, and we change two parameters (the mixing parameter $\mu$ and maximum community size $mc$) to have more variations. Fig. 4.2 shows the results, in which GT is the ground truth given by the generator. Generally, the increase of $\mu$ and $mc$ will decrease the community detection accuracy, but the results generated by our algorithm are always most close to the ground truth according to NMI. The results with our algorithm are also very close to the ground truth in numbers of communities (except for MT which is predefined) and modularity, and are usually better than others in modularity. In Fig. 4.2(b), as the problem becomes more difficult from left to right, our algorithm spends more time to maintain the quality, while the other algorithms have less time variation but sacrifice the quality. In addition, the performance of our algorithm is stable as the running results using 8 processors (P8) and 64 processors (P64) are fairly close.

We also compare algorithms on real-world graphs: cit-Patents [120] and dblp-
Table 4.2: Comparison on real-world data

Coauthor\(^1\) [121]. Nodes in cit-Patents are patents in U.S. patent dataset and edges represent the citation relationships between the two patents. dblp-Coauthor is a coauthor graph extracted from publications in DBLP database. It is also worth noting that LVF can achieve higher NMI and have more communities than LV in benchmark test. It indicates that LV suffers from the resolution limit problem [59] that prefers unrealistically large communities. A similar problem is observed on LV for real-world data as it generates many communities containing more than \(10^4\) nodes (Fig. 4.3). Especially, for example, for the dblp-Coauthor network, the biggest community by LV contains about \(10^5\) nodes (10% of the whole graph). It is not realistic for collaboration networks and is orders of magnitude larger than a reasonable community size for human interaction [119], so although LV can achieve a high modularity, it is omitted from the remaining comparisons. Alternatively, choosing low level structures such as the one by LVF can be a remedy [122], although it still contains a few extremely large communities. On the other hand, MT tends to find even-size communities, which is not desirable for community detection either. Table 4.2 shows the quality of results by different algorithms.


<table>
<thead>
<tr>
<th>Graph</th>
<th>Algo</th>
<th>Cond</th>
<th>Mod</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>cit-Patents</td>
<td>PS1</td>
<td>0.418</td>
<td>0.631</td>
<td>1514</td>
</tr>
<tr>
<td></td>
<td>PS8</td>
<td>0.416</td>
<td>0.638</td>
<td>284</td>
</tr>
<tr>
<td></td>
<td>PS64</td>
<td>0.415</td>
<td>0.639</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>LVF</td>
<td>0.594</td>
<td>0.560</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>MT</td>
<td>0.488</td>
<td>0.487</td>
<td>243</td>
</tr>
<tr>
<td>dblp-Coauthor</td>
<td>PS1</td>
<td>0.114</td>
<td>0.631</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td>PS8</td>
<td>0.113</td>
<td>0.632</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>PS64</td>
<td>0.112</td>
<td>0.632</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>LVF</td>
<td>0.397</td>
<td>0.561</td>
<td>22</td>
</tr>
<tr>
<td></td>
<td>MT</td>
<td>0.621</td>
<td>0.253</td>
<td>73</td>
</tr>
</tbody>
</table>
4.4.3 Microscopic Example on dblp-Coauthor Network

In this section, we analyze the community detection results of one execution by our Par-SBM (PS) and the Louvain method [44] using random initialization. For the Louvain method, we pick the Level One structure with finest communities (LVF) and the Level Two structure denoted by LV2. Communities become larger in higher level structures by the Louvain method. Taking the community containing author “Michael Wooldridge” as an example, we use $S_{PS}$, $S_{LVF}$, and $S_{LV2}$ to represent the nodes in the community found by Par-SBM and the Louvain method, respectively. The community size by our algorithm is comparable to the one by LVF, but much smaller than the one by LV2, as $|S_{PS}| = 255$ and $|S_{LVF}| = 166$, while $|S_{LV2}| = 26294$.

All the algorithms are able to identify blue nodes in Fig. 4.4 as a separate community, while our Par-SBM provides the clearest detail. $S_{LV2}$ contains one hundred times more nodes than $S_{PS}$. Among all the communities found by Par-SBM, the red, green, and purple communities in Fig. 4.4(a) are the top-three contributors to $S_{LV2}$, owning 2.96% of total nodes in $S_{LV2}$. The three communities can be represented by three nodes respectively: purple by Dr. Michael Wooldridge (multi-agent systems), green by Dr. Bruce W. Porter (knowledge systems), and red by Dr. Christopher W. Geib (Reports of the AAAI Conference Workshops). Nodes in red forms a strong community because the authors (although from different areas of AI) are fully connected by the workshop reports annually. However, some nodes in the aforementioned three communities are inappropriately separated by LVF as shown in Fig. 4.4(b).

4.5 Discussion

In this chapter, we propose a parallel algorithm to find non-overlapping community structures using stochastic block models. The time complexity is linear in the number of nodes in the graph, and thus it is suitable for large graphs. To our knowledge, it
Figure 4.4: Subgraph of the dblp-Coauthor network, where colors represent the community labels determined by different algorithms.

is the first parallel algorithm using stochastic block models.

In the next chapter, we devise a framework to find overlapping communities using non-negative matrix factorization. Our framework allows a variety of constraints to improve the detection quality for different applications. We use a recommender system as an example.
Chapter 5

Improve the Quality

5.1 General Problem Definition

A typical dimension reduction problem through non-negative matrix factorization (NMF) can be written as the factorization $V \approx WH$, where $\approx$ means approximately equal, and entries in $V$, $W$, and $H$ are all non-negative.

Often, as defined in recommender systems, the matrix $V$ contains some unknown entries. Here we adopt a binary matrix $M$ whose entry equals 0 only if the corresponding entry in $V$ is unknown. Therefore, the original factorization becomes $M \odot V \approx M \odot (WH)$, where $\odot$ indicates the element-wise multiplication. It can be solved by Weighted Non-negative Matrix Factorization (W-NMF) [79]. We note that NMF is a special case of W-NMF in which all the entries of $M$ are 1.

In addition, sometimes we may have some information about the unknown entries of $V$, for example, in a matrix containing user ratings, each entry value may have an upper or lower bound.

Therefore, in this chapter, we propose a framework to apply constraints on NMF. If the factorization is weighted, our framework is called Weighted and Constrained Non-negative Matrix Factorization (WC-NMF). If unweighted, namely, the entries of the weight $M$ are all ones, we call our framework Constrained Non-negative Matrix Factorization (C-NMF).
Table 5.1: Notation of symbols for Non-negative Matrix Factorization

<table>
<thead>
<tr>
<th>Notation</th>
<th>Definition</th>
<th>Example &amp; Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>○</td>
<td>Element-wise multiplication. The result of the multiplication is called the Hadamard product.</td>
<td>For vectors and matrices.</td>
</tr>
<tr>
<td>−</td>
<td>Element-wise division.</td>
<td>For vectors and matrices.</td>
</tr>
<tr>
<td>[·]+</td>
<td>The positive part of the anterior matrix.</td>
<td>Given a real matrix $A$, $A^+$ is the matrix whose entries are $A^+<em>{i,j} = \begin{cases} A</em>{i,j}, &amp; \text{if } A_{i,j} &gt; 0 \ 0, &amp; \text{otherwise} \end{cases}$ If $h$ is a vector, $\nabla F(h)^+$ and $\nabla^2 F(h)^+$ are the positive parts of the gradient vector and the Hessian matrix respectively.</td>
</tr>
<tr>
<td>[·]−</td>
<td>The negative part of the anterior matrix.</td>
<td>Given a real matrix $A$, $A^-$ is the matrix whose entries are $A^-<em>{i,j} = \begin{cases} -A</em>{i,j}, &amp; \text{if } A_{i,j} &lt; 0 \ 0, &amp; \text{otherwise} \end{cases}$ so that $A = A^+ - A^-$. Similarly, $\nabla F(h)^-$ and $\nabla^2 F(h)^-$ are the corresponding negative parts.</td>
</tr>
<tr>
<td><a href="i">·</a></td>
<td>Different values of $i$ indicate different variables or functions.</td>
<td>$A^{(1)}$ and $A^{(2)}$ could be different matrices, but may share some common properties (e.g., coexisting in constraint functions).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>·</td>
</tr>
<tr>
<td>$\nabla_x F(·)$</td>
<td>The derivative of function $F$ with respect to matrix $x$.</td>
<td>If $F(x) =</td>
</tr>
</tbody>
</table>
5.1.1 Algorithm for Weighted Non-negative Matrix Factorization

Assume the data matrix is $V \in \mathbb{R}^{M \times N}$, and we want to represent it by the product of two lower rank matrices $W \in \mathbb{R}^{M \times K}$ and $H \in \mathbb{R}^{K \times N}$. Typically, $K$ is expected to be much smaller than $M$ and $N$, such that columns of $W$ or rows of $H$ can be considered as a collection of feature vectors. A feature vector is a numerical vector that can represent common properties of certain data items.

The NMF problem with weighting is

$$\min_{W \geq 0, H \geq 0} ||M \circ [V - (WH)]||,$$

in which $M \in \mathbb{R}^{M \times N}$, and $\geq$ is an element-wise operator.

Similar to the typical approach for ordinary NMF, we can update the two factors successively in each iteration for W-NMF until $W$ and $H$ do not change.

Algorithm 6 W-NMF: Weighted Non-negative Matrix Factorization

1: input Training set $V$, Integer $K$
2: output Factorization results $W$ and $H$
3: initialize $W$ and $H$;
4: repeat
5:     $W \leftarrow \text{argmin}_{W \geq 0} ||M \circ [V - (WH)]||$
6:     $H \leftarrow \text{argmin}_{H \geq 0} ||M \circ [V - (WH)]||$
7: until converged

Zhang et al. [79] in 2006 introduced a multiplicative algorithm [69] with element-by-element updates to solve the WNMF.

$$W_{i,a} \leftarrow W_{i,a} \frac{(M \circ V)H^T}{M \circ (WH)H^T}_{i,a},$$
\[ H_{a,j} \leftarrow H_{a,j} \frac{\{W^T(M \circ V)\}_{a,j}}{\{W^T[M \circ (WH)]\}_{a,j}}, \]  

(5.3)

where \( \{ \cdot \}_{i,j} \) indicates the entry \((i, j)\) of the matrix inside the brackets.

Zhang et al. [79] provide only the iterative formula, while we demonstrate herein that it is a method with guaranteed convergence.

5.2 Weighted and Constrained Non-negative Matrix Factorization

5.2.1 Algorithm

Let \( J(W, H) = ||M \circ [V - (WH)]||^2 \). Therefore, Eq. (5.6) and (5.7) can be rewritten as

\[ W \leftarrow W \circ \nabla_W J(W, H)^-, \]  

(5.4)

\[ H \leftarrow H \circ \nabla_H J(W, H)^-, \]  

(5.5)

Suppose we have an NMF algorithm with multiple constraints as illustrated in Algorithm 7.

By modifying the constraint function into squares, the Lagrange function can be written as

\[ F(W, H) = J(W, H) + \sum_i \lambda_i ||C^{(i)}||^2. \]  

(5.8)

Hence, after replacing \( J(W, H) \) with \( F(W, H) \) in Eq. (5.4) and (5.5), we have the update rules for each iteration:
Algorithm 7 Framework of Weighted and Constrained Non-negative Matrix Factorization

1: **input** Training set $V$, Integer $K$, Constraint function $C$
2: **output** Factorization results $W$ and $H$
3: Randomly initialize $W$ and $H$
4: **repeat**
5: $W \leftarrow \text{argmin}_{W \geq 0} J(W, H)$
   \hspace{1cm} s.t. $C(i)(W, H) = 0, i = 1, 2, 3, ...$
   \hspace{1cm} (5.6)
6: $H \leftarrow \text{argmin}_{H \geq 0} J(W, H)$
   \hspace{1cm} s.t. $C(i)(W, H) = 0, i = 1, 2, 3, ...$
   \hspace{1cm} (5.7)
7: **until** Converged

$W \leftarrow W \odot \frac{\nabla W J(W, H)^- + \sum_i \lambda_i |\nabla W||C(i)||^2|^\ominus}{\nabla W J(W, H)^+ + \sum_i \lambda_i |\nabla W||C(i)||^2|^\oplus}$, (5.9)

$H \leftarrow H \odot \frac{\nabla H J(W, H)^- + \sum_i \lambda_i |\nabla H||C(i)||^2|^\ominus}{\nabla H J(W, H)^+ + \sum_i \lambda_i |\nabla H||C(i)||^2|^\oplus}$, (5.10)

Sometimes, we need inequality constraints as well, for example, to set an upper or lower bound for the unknown variables. If there are inequality constraints, we can use a non-negative slackness variable $S$ to transform the inequality to be an equality. For example, given the constraint $C^{(j)} \leq 0$, the corresponding $S$ should satisfy $C^{(i)} + S^{(i)} = 0$, so $C^{(i)}$ in Eq. (5.9) and (5.10) should be replaced by $C^{(i)} + S^{(i)}$ with an additional update rule in each iteration:

$S^{(i)} \leftarrow S^{(i)} \odot \frac{[C^{(i)}]^-}{[C^{(i)}]^+ + S^{(i)}}$. (5.11)
5.2.2 Convergence Analysis of the Multiplicative Algorithms

In this section, we provide a convergence proof for the above multiplicative algorithm. In this algorithm, the objective function $F$ can be written as a quadratic function, with all the parameters non-negative in the quadratic terms.

In our framework, the constraint functions are linear in the variables $(W$ or $H)$. We first assume gradients of these linear constraint functions are non-negative matrices. Without loss of generality, we use vector $h$ to denote one column of $H$ with length $K$ and we consider $h$ as the only variable for function $F$ in Eq. (5.8). Because $F$ is a summation of Frobenius norms on linear functions of $h$, we can represent all the parameters in $F$ by a matrix $Y$, a row vector $Z$ and a constant $c$.

$$F(h) = h^T Y h + Z h + c.$$ (5.12)

We also partition $Y$ and $Z$ into two parts respectively: $Y = Y^+ - Y^-$ and $Z = Z^+ - Z^-$, where all the entries in $Y^+$, $Y^-$, $Z^+$, and $Z^-$ are non-negative. Because the gradients of the constraint functions are non-negative, $Y$ is the summation of non-negative covariance matrices. Therefore, $Y$ is also a non-negative covariance matrix and $Y^-$ is a zero matrix. Consequently, $Y^+$ is a covariance matrix so it is symmetric. By using these notations and omitting the constant terms in the final summation, we can rewrite the function $F$ as follows:

$$F(h) = h^T Y^+ h + Z^+ h - Z^- h.$$ (5.13)

Then we can partition the gradient of $F$ into two parts:

$$\nabla F(h) = \nabla F(h)^+ - \nabla F(h)^-,$$ (5.14)

where $\nabla F(h)^+ = 2Y^+ h + Z^+$ and $\nabla F(h)^- = Z^-$. 
Now we need to prove the following theorem.

**Theorem 5.** The objective function $F(h)$ in Eq. (5.13) is non-increasing if its argument $h$ is updated according to the following rule:

$$h_t \leftarrow h_{t-1} \circ \frac{\nabla F(h_{t-1})^-}{\nabla F(h_{t-1})^+},$$

(5.15)

where the division is an element-wise operator.

**Proof.** In the proof, we use $\text{diag}(\cdot)$ to represent a square matrix whose main diagonal is the vector inside the parentheses. Assume $v$ is an arbitrary vector of length $K$. Then we have

$$v^T \left[ \text{diag} \left( \frac{\nabla F(h)^+}{h} \right) - \nabla^2 F(h) \right] v$$

$$= v^T \left[ \text{diag} \left( \frac{Y^+ h + Z^+}{h} \right) - Y^+ \right] v$$

$$= \sum_{i=1}^{N} \left( v_i^2 \frac{\{Y^+ h + Z^+\}_i}{\{h\}_i} \right) - \sum_{i=1}^{N} \sum_{j=1}^{N} (v_i Y_{i,j}^+ v_j)$$

$$\geq \sum_{i=1}^{N} \left( v_i^2 \frac{\{Y^+ h + Z^+\}_i}{\{h\}_i} \right) - \sum_{i=1}^{N} \sum_{j=1}^{N} (v_i Y_{i,j}^+ v_j)$$

$$\geq \sum_{i=1}^{N} \sum_{j=1}^{N} (v_i^2 Y_{i,j}^+) - \sum_{i=1}^{N} \sum_{j=1}^{N} (v_i Y_{i,j}^+ v_j)$$

$$= \sum_{i=1}^{N} \sum_{j=1}^{N} \left[ \frac{1}{2} Y_{i,j}^+ (v_i - v_j)^2 \right]$$

$$\geq 0.$$
Consequently, according to the definition in Eq. (5.13), we have

\[ v^T \left[ \text{diag} \left( \frac{\nabla F(h^+)}{h} \right) - \nabla^2 F(h) \right] v \]

\[ = v^T \left[ \text{diag} \left( \frac{\nabla F(h^+)}{h} \right) - \nabla (\nabla F(h^+)) \right] v \]

\[ = v^T \left[ \text{diag} \left( \frac{\nabla F(h^+)}{h} \right) - \nabla^2 F(h^+) \right] v + v^T [\nabla^2 F(h^-)] v \]

\[ \geq 0, \]

where the last inequality is valid as \( \nabla^2 F(h^-) \) is always a zero matrix by assumption. This insures that the matrix

\[ \left[ \text{diag} \left( \frac{\nabla F(h^+)}{h} \right) - \nabla^2 F(h) \right] \]

is positive semi-definite.

Then we can use the idea of an auxiliary function [69] to verify the correctness of our update rule. First we define a function

\[ G(h^{(1)}, h^{(2)}) = F(h^{(2)}) + (h^{(1)} - h^{(2)})^T \nabla F(h^{(2)}) \]

\[ + \frac{1}{2} (h^{(1)} - h^{(2)})^T \text{diag} \left( \frac{\nabla F(h^{(2)^+})}{h^{(2)}} \right) (h^{(1)} - h^{(2)}). \]

By comparing it with Taylor series of the objective function

\[ F(h^{(1)}) = F(h^{(2)}) + (h^{(1)} - h^{(2)})^T \nabla F(h^{(2)}) \]

\[ + \frac{1}{2} (h^{(1)} - h^{(2)})^T \nabla^2 F(h^{(2)})(h^{(1)} - h^{(2)}), \]

and according to the result in Eq. (5.17), we find it always holds for any \( h^{(2)} \) that

\[ G(h^{(1)}, h^{(2)}) \geq F(h^{(1)}). \]
Further, if the arguments are the same, it is also easy to see that

$$G(h^{(2)}, h^{(2)}) = F(h^{(2)}).$$

Therefore, $G(h^{(1)}, h^{(2)})$ is an auxiliary function for $F(h^{(1)})$. Hence, $F(h)$ is non-increasing by taking the following update rule

$$h_t \leftarrow \arg\min_h G(h, h_{t-1}).$$

According to the results that $F(h_t) \leq G(h_t, h_{t-1})$ by Eq. (5.21), $G(h_{t-1}, h_{t-1}) = F(h_{t-1})$ by Eq. (5.22) and $G(h_t, h_{t-1}) \leq G(h_{t-1}, h_{t-1})$ by Eq. (5.23). It is similar as Lemma 1 in Lee and Seung’s work [69] except that we have additional terms for constraints.

According to the definition of $G$ in Eq. (5.19), we choose $h_t$ to be the solution of $\nabla_h G(h_t, h_{t-1}) = 0$, so that $h_t = \arg\min_h G(h, h_{t-1})$. Following Eq. (5.23) we have

$$h_t \leftarrow h_{t-1} - \left[ \text{diag} \left( \frac{\nabla F(h_{t-1})^+}{h_{t-1}} \right) \right]^{-1} \nabla F(h_{t-1})$$

$$= h_{t-1} - \left[ \text{diag} \left( h_{t-1} \frac{\nabla F(h_{t-1})^+}{\nabla F(h_{t-1})^+} \right) \right] \nabla F(h_{t-1})$$

$$= h_{t-1} - \frac{h_{t-1}}{\nabla F(h_{t-1})^+} \circ \nabla F(h_{t-1})$$

$$= h_{t-1} - \frac{h_{t-1}}{\nabla F(h_{t-1})^+} \circ (\nabla F(h_{t-1})^+ - \nabla F(h_{t-1})^-)$$

$$= h_{t-1} \circ \frac{\nabla F(h_{t-1})^-}{\nabla F(h_{t-1})^+}. \quad (5.24)$$

Because it is possible that some elements in the gradient are negative, a naive gradient descent method admits negative values. However, by the new approach above, the result is non-negative, and a local minimum of $F$ can be found after several iterations. Iterative methods described in Eq. (5.2), (5.3), (5.37), (5.43), and
(5.46) are in the framework of our updating rule if we write ours in element format

\[ \{h_t\}_a \leftarrow \{h_{t-1}\}_a \circ \left( \frac{\nabla F(h_{t-1})^-}{\nabla F(h_{t-1})^+} \right)_a. \]  

(5.25)

We can generalize Theorem 5 by analyzing Eq. (5.17) in the proof. We find that to make the inequality of Eq. (5.17) valid, \( \nabla^2 F(h)^- \) need not necessarily to be a zero matrix, but can be any positive semi-definite matrix, as follows.

**Theorem 6.** Given the objective function

\[
F(h) = h^T Y^+ h - h^T Y^- h + Z^+ h - Z^- h
= h^T Y^+ h - h^T Y^- h + Zh,
\]

(5.26)

where \( Y^+ \) and \( Y^- \) are non-negative matrices, \( Y^- \) is positive semi-definite, and \( Z \) can be any constant matrix. Then, \( F(h) \) is non-increasing if its argument \( h \) is updated according to the following rule:

\[
h_t \leftarrow h_{t-1} \circ \frac{\nabla F(h_{t-1})^-}{\nabla F(h_{t-1})^+}.
\]

(5.27)

\( Y^+ \) can, in fact, be unbounded. Therefore, we can add an appropriate non-negative matrix \( X \) to \( Y^- \) to make it positive semi-definite, and add the same matrix to \( Y^+ \), so that the algorithm can always converge, under the following update rule:

**Corollary 7.** Given the objective function

\[
F(h) = h^T Y h + Zh
= h^T Y^+ h - h^T Y^- h + Zh,
\]

(5.28)

where \( Y \) and \( Z \) can be any constant matrices, while \( Y^+ \) and \( Y^- \) are non-negative
matrices, let $X$ be such a non-negative matrix such that $X + Y^-$ is positive semi-definite. Then, $F(h)$ is non-increasing if its argument $h$ is updated according to the following rule:

$$h_t \leftarrow h_{t-1} \circ \frac{Xh + \nabla F(h_{t-1})^-}{Xh + \nabla F(h_{t-1})^+}.$$  \hspace{1cm} (5.29)

**Proof.** We choose $X$ to make sure that the derivative of the numerator with respect to $h$ in Eq. (5.29) is positive semi-definite. It is equivalent to choose $X$ so that $X + Y^-$ is positive semi-definite. If we set $Y^- \leftarrow X + Y^-$ and $Y^+ \leftarrow X + Y^+$, the requirements for Eq. (5.27) can be met. Then, according Theorem 6, this corollary can be proved.

This corollary allows us to remove the assumption at the beginning of this section, so that the gradients of the constraint functions can be real matrices. When without the assumption, to have our framework still works, we can use the update rule according to Eq. (5.29). If $Y^-$ is zero or positive definite, $X$ can be a zero matrix. Otherwise, $X$ should be a non-negative diagonal matrix such that $X + Y^-$ is positive definite.

Because the original objective function of NMF is always quadratic, we need only to make sure the constraint is quadratic or linear, which includes a large variety of functions.

When the argument $h$ is a row vector, the proof is almost the same by simply considering $h^T$ as an argument for function $F$. The vector version can also be extended easily to a matrix version.
5.3 Sample Implementations

In this section, we give two sample constraints for the weighted and constrained NMF problem. The first is a matrix of upper bounds $U$ for entries in $V$. The second is to require $H$ to be an orthogonal matrix: $HH^T = I$.

These two constraints can be applied independently, by choosing appropriate parameters. The first constraint should be derived from the property of the problem itself, while the second is established for the ease of analyzing the basis components.

This produces a new algorithm, in which $\leq$ is again an element-wise operator.

Algorithm 8 Weighted and Constrained Non-negative Matrix Factorization

1: **input** Training set $V$, Integer $K$, Upper bound $U$
2: **output** Factorization result $W$ and $H$
3: Randomly Initialize $W$ and $H$;
4: **repeat**
5: \[
W \leftarrow \arg\min_{W \geq 0} ||M \circ [V - (WH)]||^2 \\
s.t. WH \leq U
\]
6: \[
H \leftarrow \arg\min_{H \geq 0} ||M \circ [V - (WH)]||^2 \\
s.t. HH^T = I
\]
7: **until** converged

Theoretically, taking the orthogonalization into account, the time complexity is similar to Ding et al’s algorithm [71].

5.3.1 Procedure to Calculate $H$ in Eq. (5.31)

By rewriting the optimization problem in Lagrangian function and applying the Frobenius norm on the constraints, we get the following expression with a Lagrange multiplier [71] defined by $\lambda_H$:
\[ H = \arg \min_{H \geq 0} ||M \circ [V - (WH)]||^2 + \lambda_H ||HH^T - I||^2 \]
\[ = \arg \min_{H \geq 0} \frac{1}{2} ||M \circ [V - (WH)]||^2 + \frac{1}{2} \lambda_H ||HH^T - I||^2. \]  

(5.32)

We approximate the quadratic term in the constraint by a linear expression \( HH^T_{(t-1)} - I \) where the notation \( H_{(t-1)} \) represents the value of the unknown variable \( H \) in the previous iteration.

Let \( C \) be the second term of the Lagrangian function:

\[ C(H) = \frac{1}{2} \lambda_H ||HH^T_{(t-1)} - I||^2. \]  

(5.33)

We then have

\[
\nabla_H C(H) \\
= \nabla_H \left[ \left( \frac{1}{2} \lambda_H \right) \left( HH^T_{(t-1)} - I \right) \right] \\
= \lambda_H \left[ (HH^T_{(t-1)} - I) H_{(t-1)} \right] \\
= \lambda_H \left[ HH^T_{(t-1)} H_{(t-1)} - I H_{(t-1)} \right] \\
= \lambda_H H H^T_{(t-1)} H_{(t-1)} - \lambda_H H_{(t-1)} \\
= p^+ - p^-,
\]

where variables \( p^+_{a,i} \) and \( p^-_{a,i} \) are shorthand for two terms of the derivatives for the constraints for short. Each of the variables are of the same size as \( H \).

Eq. (5.35) is an iterative approach to solve the optimization problem by a gradient descent method [123]. For simplicity, we omit the subscript \( (t-1) \) for \( H_{(t-1)} \) on the right part, as \( H \) is always from the previous iteration.
\[ H \leftarrow H - \eta_H \{-W^T [M \odot V - M \odot (WH)] + p^+ - p^-\} \]

\[ = H + \eta_H \{W^T [M \odot V - M \odot (WH)] - (p^+ - p^-)\} \tag{5.35} \]

\[ = H + \eta_H \{W^T (M \odot V) + p^- - W^T [M \odot (WH)] - p^+\}. \]

In the gradient descent method, let the step length \( \eta_H \) have the expression

\[ \eta_{H,a,i} = \frac{\{H\}_{a,i}}{\{W^T [M \odot (WH)] + p^+\}_{a,i}}. \tag{5.36} \]

to

\[ \{H\}_{a,i} \leftarrow \{H\}_{a,i} \frac{\{W^T (M \odot V) + p^-\}_{a,i}}{\{W^T [M \odot (WH)] + p^+\}_{a,i}} \]

\[ = \{H\}_{a,i} \frac{\{W^T (M \odot V) + \lambda_H H\}_{a,i}}{\{W^T [M \odot (WH)] + \lambda_H H H^T H\}_{a,i}}. \tag{5.37} \]

When \( M \) is a binary matrix matching the training matrix, as in our application, Eq. (5.37) can be simplified by removing \( M \) from the numerator:

\[ \{H\}_{a,i} \leftarrow \{H\}_{a,i} \frac{\{W^T V + \lambda_H H\}_{a,i}}{\{W^T (WH) + \lambda_H H H^T H\}_{a,i}}. \tag{5.38} \]

Another advantage of our algorithm is that, it allows a free choice of parameter \( \lambda_H \), rather than being limited to a fixed setting [71], to cater to the non-negativity of the variables.

### 5.3.2 Procedure to Calculate \( W \) in Eq. (5.30)

First, we use a slackness variable \( S \) for the inequality in the constraints of Eq. (5.30), so we have a new but equivalent expression:
\[ W = \underset{W \geq 0, S \geq 0}{\text{argmin}} \| M \circ [V - (W H)] \|^2, \text{ s.t. } WH + S = U. \quad (5.39) \]

In terms of the Lagrange function, it is

\[ W = \underset{W \geq 0, S \geq 0}{\text{argmin}} \| M \circ V - M \circ (W H) \|^2 \\
+ \lambda_W \| WH + S - U \|^2. \quad (5.40) \]

Similarly, for \( W \) we also have an iterative formula based on a gradient descent method.

\[
W \leftarrow W - \eta_W \{-[M \circ V - M \circ (W H)]H^T \\
+ \lambda_W (WH + S - U)H^T\} \\
= W + \eta_W \{(M \circ V)H^T \\
- M \circ (WH)H^T - \lambda_W (WH + S - U)H^T\} \quad (5.41) \\
= W + \eta_W \{(M \circ V)H^T + \lambda_W UH^T \\
- M \circ (WH)H^T - \lambda_W (WH + S)H^T\}. 
\]

The corresponding expression for \( \eta_W \) is

\[
\eta_{W_{i,a}} = \frac{\{W\}_{i,a}}{\{M \circ (WH)H^T + \lambda_W (WH + S)H^T\}_{i,a}}. \quad (5.42) 
\]

By replacing \( \eta_W \) accordingly, we have a multiplicative expression for \( W \)

\[
\{W\}_{i,a} \leftarrow \{W\}_{i,a} \frac{\{(M \circ V)H^T + \lambda_W UH^T\}_{i,a}}{\{M \circ (WH)H^T + \lambda_W (WH + S)H^T\}_{i,a}}. \quad (5.43) 
\]
As we want to minimize the error term due to the slackness, we have

$$S = \arg\min_{S \geq 0} ||WH + S - U||. \quad (5.44)$$

Then $S$ should have the following expression

$$S \leftarrow S - \eta_S \lambda_W (WH + S - U)$$

$$= S + \eta_S \lambda_W (U - W_i H - S). \quad (5.45)$$

Hence,

$$\{S\}_{i,a} \leftarrow \{S\}_{i,a} \frac{\{U\}_{i,a}}{\{WH + S\}_{i,a}}, \quad (5.46)$$

where $\eta_S$ is set in accordance with $\lambda_W$, namely

$$\{\eta_S\}_{i,a} = \frac{\{S\}_{i,a}}{\lambda_W \{WH + S\}_{i,a}}. \quad (5.47)$$

Usually, $W$ and $H$ are randomly initialized, but when $S$ is involved and initialized by $S = U - WH$, $W$ need to be multiplied with a small enough scalar to make sure that every element in $U$ is larger than the corresponding element in $WH$.

### 5.4 Experiments

In the second section, our proposed algorithm based on stochastic block models performs better than other algorithms on synthetic graphs, and is scalable on large real graphs. There are some interesting findings on the Wikipedia graph that cannot be found by traditional algorithms due either to resolution limit or to large time complexity.

In the third section, we use a recommender system example on bipartite networks
to demonstrate the effectiveness of our constrained non-negative matrix factorization example.

5.4.1 Application to Movielens Rating Data

Movielens data has been adopted by a number of researchers as a benchmark for their work [78, 79, 83, 106, 107, 108]. In this work, we used the data set called MovieLens 100k, which consists of 100,000 ratings from 1000 users on 1700 movies \(^1\). The smallest rating value in the data set is 1, and the largest is 5. During the test, these data were divided into two sets: about 98.6% for training, and about 1.4% for testing. The test set contains all the 5-score ratings.

We list the algorithms for comparison in Table 5.2. There are also listed some implementation details. The number of iterations are all 30 in this experiment. In C-NMF, we simply use Algorithm 5.3 by setting all the entries in \(W\) to be 1, with \(U = 5\), \(\lambda_W = 1e-4\) and \(\lambda_H = 1\). In EM-NMF, the numbers of outer iterations and inner ones are both 10, and pure-NMF [69] is used for inner iteration. In WC-NMF, we use the full version of Algorithm 5.3 where \(W\) serves as a mask, with \(U = 5\), \(\lambda_W = 1e-4\) and \(\lambda_H = 0.1\). W-NMF uses Algorithm 6.

We run algorithms on the training set and evaluate the quality based on the test set. The results are averaged from ten runs.

There are several metrics for measuring recommender system algorithms. Among them, the top-N method measures the probability that a 5-score test movie can be in the list when obtaining the \(N\) top-ranked movies according to the algorithm [83]. This measurement has its practical significance, because a recommender system is usually asked to provide a few items that appeal to the user. The detailed procedure of the evaluation is [83]: for each 5-score rating in the test set, first, we randomly select 1000 unrated movies for the user; second, we predict the user ratings for all the

\(^1\)MovieLens Data Sets, http://grouplens.org/datasets/movielens/
\(^2\)http://www.cs.toronto.edu/~dross/code/nmfm
Table 5.2: List of algorithms for comparison

<table>
<thead>
<tr>
<th>Without Weights</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>C-NMF</td>
<td>Our algorithm, NMF only with constraints but no weights.</td>
</tr>
<tr>
<td>Pure NMF</td>
<td>Pure NMF algorithm [69] with internal initialization.</td>
</tr>
<tr>
<td>EM NMF</td>
<td>Expectation Maximization, with NMF [79].</td>
</tr>
<tr>
<td>Orth NMF</td>
<td>NMF with orthogonalization, initialized by K-means [71].</td>
</tr>
<tr>
<td>Pure SVD</td>
<td>Recommender simply by SVD decomposition [83].</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>With Weights</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Algorithm Name</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>WC-NMF</td>
<td>Our algorithm, NMF with weights and constraints.</td>
</tr>
<tr>
<td>W-NMF</td>
<td>Weighted NMF [79].</td>
</tr>
<tr>
<td>Hybrid NMF</td>
<td>run EM NMF for initialization, and then run W-NMF [79].</td>
</tr>
</tbody>
</table>

1001 movies, and recommend the $N$ top-ranked movies in the prediction; third, if the 5-score movie is in the recommendation, the recall is 1 and otherwise, it is zero. The recalls in the experiment are averaged from all the 5-score ratings.

The traditional element-wise comparisons like RMSE and NMAE are made as well. Root-Mean-Square Error (RMSE) compares the difference between the predicted values with the values in the test set. It first averages squares of differences, and take the root-mean-square over the average. Normalized-Mean-Absolute Error (NMAE) is similar except that it averages the magnitude of differences at first. RMSE emphasizes more on large differences than NMAE.

From the test results in Fig. 5.1, we can find that the algorithms without weights perform better in top-N metric, while those with weights perform better in RMSE and NMAE.

In top-N when $K \geq 50$, our algorithm C-NMF performs better than all the other
methods, including pure SVD, the best algorithm at $K = 50$ in Cremonesi et al.’s work [83]. Among the algorithms without weights, when $K \geq 40$, C-NMF is the best in RMSE and NMAE, and in other cases, it has a comparable achievement. EM-NMF is the worst in top-N in the category, and although it does better in RMSE and NMAE for small $K$, it is extremely unstable when $K$ is large. Among those with weights, in top-N when $K \geq 20$, our algorithm WC-NMF performs much better than the other methods, and has a comparable result in RMSE and NMAE.

To fully utilize the advantage of our upper-bound constraint, we run a second test by small modification. First, we squeeze the lower bound space by deducting 0.5 for the nonzero entries in the training set $V$, and apply an upper-bound of 4.5. Therefore, the actual data range becomes $[0.5, 4.5]$. Second, after running each algorithm, we add 0.5 to their predicting result to compare with the original test data. For this test, we set $U = 4$, $\lambda_W = 0.001$ and $\lambda_H = 0.1$ for C-NMF and $U = 4.5$, $\lambda_W = 0.01$ and $\lambda_H = 0.1$ for WC-NMF. In Fig. 5.2, We find that both C-NMF and WC-NMF have a similar performance in their own group, while WC-NMF turns out to be consistently better than W-NMF.

Cremonesi et al. [83] claim that some popular items may mislead the top-N result. In their Movielens data set, about 6% of the most popular movies containing 33% of the total ratings, and when these movies are removed, SVD performs the best when $K = 50$ and $K = 150$ respectively. Therefore, we launch a third test, in which we remove the aforementioned 6% most popular movies, and used the same parameters as in the first test. From Fig. 5.3(a) and 5.4, we can see that in top-N, C-NMF still performs better than all the others including SVD when $K \geq 50$. Especially compared to SVD in Fig. 5.4, it outperforms 3% at $K = 50$ and 45% at $K = 150$ when $N = 20$. Also in top-N, WC-NMF is the best within the weighted category.

In the fourth test, we use the squeezed lower-bound strategy on the data set with the most popular items removed. Under the same parameter setting as in the second
Figure 5.1: Tests on data with upper bound applied in C-NMF and WC-NMF
Figure 5.2: Tests on data with squeezed lower bound in C-NMF and WC-NMF
Figure 5.3: Tests on data with most popular items removed
Figure 5.4: Top-N test on data with most popular items removed

(a) Recall versus $N$ at $K = 50$

(b) Recall versus $N$ at $K = 150$
test, the relative performance of our algorithms in Fig. 5.3(b) and 5.3(c) is quite similar to that in Fig. 5.2.

In summary, if top-N metrics are emphasized, C-NMF is an ideal choice, and if all the three metrics are important, WC-NMF is good especially when the lower bound of the data is squeezed.

5.5 Discussion

In this chapter, we propose a constrained non-negative matrix factorization algorithm using multiplicative methods. Specific constraints can improve the quality for specific problem, and we use an application on recommender system via user-item bipartite graphs as an example.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we propose several algorithms and frameworks to solve the community detection problem on networks.

To reduce the running time for many existing community detection algorithms, we provide a $K$-core based framework. It can accelerate existing algorithms significantly and preserve the quality in terms of modularity maximization. It includes three steps: first, find the $K$-core of the original graph; second, run a community detection algorithm on the $K$-core; third, find labels for all the nodes outside the $K$-core and then optimize for the whole graph. Experiments demonstrate efficiency and accuracy on different real graphs, under several quality measurements. Theoretical analysis supports our approach through $K$-core.

As the second aspect, we propose a fast algorithm for clustering nodes in large graphs using stochastic block models. We adopt alternating updates and coordinate descent methods to infer the latent parameters, and use cluster shrinking and expanding to improve the accuracy. We demonstrate experimentally that the time complexity of our algorithm is only linear in the number of edges, and can scale to multi-processor systems for parallel computing. Compared with other community detection algorithms on benchmark graphs, our algorithm has outstanding performance.
in terms of speed and accuracy. For large real-world data, our algorithm is scalable on multi-processor systems. This work can boost the application of SBMs on big data and bring new insights by overcoming the accuracy or running time shortages of traditional algorithms.

To improve the node clustering performance on bipartite networks, we devise a framework for both Weighted and Constrained Non-negative Matrix Factorization and Constrained Non-negative Matrix Factorization. We provide a convergence guaranteed multiplicative method for NMF with general formulation of constraints, as long as the constraint function is linear with respect to each of the factors. Our approach also allows the freedom to choose a suitable tightness for each constraint when embedded into the objective function. These two properties can facilitate the application of NMF method in different domains. In addition, we give examples of specialized constraints. The first constraint is to limit the upper bounds for unknown entries during the factorization, and the second is to orthogonalize the factor matrices. Taking the recommender system as an example, our algorithm can achieve excellent results in top-N metric, and can gain small improvements under other metrics.

6.2 Directions for Future Work

Several directions can be explored to extend our work in this thesis.

First, the $K$-core framework can be extended for overlapping community detection algorithms. A similar three step framework may be as follows: first, extract the $K$-core; second, run an traditional overlapping community detection algorithm on the $K$-core; third, use a fast local optimization algorithm to find the community labels of the whole graph. The fast local algorithm may be a local modularity maximization for overlapping communities [124].

Second, the parallel algorithms for inferring parameters for stochastic block models
may be extended to mixed membership stochastic block models [84]. Approximations for the model may be introduced to reduce the communication.

Third, the multiplicative algorithm for non-negative matrix factorization may be accelerated by choosing a non-positive $X$ in Eq. (5.29). Here a non-positive matrix means no entry in the matrix is larger than zero. This is possible because we need only to make sure that $Y^+ \leftarrow X + Y^+$ is positive semi-definite and $Z^- \leftarrow Xh + Z^-$ is non-negative, and then use the update rule of Eq. (5.27). Intuitively, the convergence can be accelerated because the numerator and the denominator are subtracted by the same value while maintain the positiveness.

In short, our research advances the community detection algorithms in speed, scalability, and quality. In addition, this work suggests many open research directions.
REFERENCES


[64] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer,


7 Papers Submitted and Under Preparation

- Chengbin Peng, Ka-Chun Wong, “Exploring Mixed Membership Stochastic Block-models via Non-negative Matrix Factorization”, *The Third International Workshop on Intelligent Data Processing in Conjunction with the 14th IEEE International Conference on Data Mining, Shenzhen, 2014*.