HEURISTICS FOR LARGE-SCALE SEMIDEFINITE PROGRAMMING FOR THE K DISJOINT CLIQUE PROBLEM

by

ALEX BARNES

BRENDAN AMES, COMMITTEE CHAIR
DAVID HALPERN
VOLODYMYR MELNYKOV
SHAN ZHAO
WEI ZHU

A DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Mathematics
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2018

ABSTRACT

Large-scale semidefinite programming has many applications, including optimal control, computer vision, and machine learning. However, current algorithms for solving semidefinite programs (SDPs) can be time consuming and memory intensive. We look at new heuristics for solutions of the K Disjoint Clique problem. We model the K Disjoint Clique optimization problem as a SDP based on non-convex low rank factorization, and solve using Alternating Direction Method of Multipliers, augmented Lagrangian, and alternating direction. We will present numerical results illustrating the efficacy of our approach for clustering of real and simulated data and pose future questions of interest.

DEDICATION

I would like to dedicate this work to my wife, Rachel, and our son, Liam. Thank you for your unwavering support, and love during writing this. I am also very thankful to my family and friends that have encouraged me along the way from grade school to graduate school.

LIST OF ABBREVIATIONS AND SYMBOLS

λ	Normal lowercase letters depict constants
Σ	Normal uppercase letters depict sets
$\boldsymbol{X}$	Bold uppercase letters depict matrices
$\boldsymbol{x}$	Bold lowercase letters depict vectors
$\boldsymbol{X_{ij}}$	Bold uppercase letters with bold subscripts depict matrices
$\boldsymbol{X}_{ij}$	Normal uppercase letters with subscripts depict matrix elements
$\boldsymbol{x_i}$	Bold lowercase letters with bold subscripts depict vectors of a tensor

ACKNOWLEDGMENTS

I would like to acknowledge and thank my dissertation advisor, Dr. Brendan Ames, as well as my dissertation defense committee members: Dr. David Halpern, Dr. Volodymyr Melnykov, Dr. Shan Zhao, and Dr. Wei Zhu, and thank The University of Alabama for their support through the entire process.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The world continues to collect data at an increasing rate. Data drives daily decisions and quantifies performance of decisions. As data plays an ever increasing role in the world, we desire to analyze data more efficiently and accurately.

One of the first questions when analyzing data is which data item is similar to other data. Clustering data has become a very important problem as the world continues to collect and analyze more data. This problem is the main goal of the clustering problem. Clustering works to group data points together that are similar, and exclude points from clusters which are not similar. We specifically look at the K Disjoint Clique optimization problem which, as a model for the clustering problem, aims to find K disjoint cliques, or complete subgraphs, while maximizing the number of data points included in the K disjoint cliques. A clique refers to a fully connected cluster or subgraph and relates data to a graph via a similarity graph. For example, if we consider the case where $K = 1$, which is known as the maximum node K disjoint clique problem [Ames and Vavasis, 2014], we want to find one disjoint clique that includes the most points of our data. Although, the K Disjoint Clique problem is know to be NP-hard; recent research has been suggested this problem can be solved exactly if the underlying data consists of planted disjoint cliques plus noise in the form of edges between cliques and outliers [Ames and Vavasis, 2014].

We focus on optimization schemes for solving the K Disjoint Clique problem. These approaches will be based on first order methods for solving SDP's. Semidefinite programming, SDP, has received much attention recently in the math community as it has many applications and many still open questions. Applications include, but are not

limited to, relaxations of combinatoral optimization problems, machine learning, statistics, and computer vision. In this paper we use techniques, including Alternating Direction Method of Multipliers [Boyd et al., 2011], low rank factorization [Burer and Monteiro, 2003], augmented Lagrangian or penalization [Burer and Monteiro, 2003], and Biconvex relaxation with alternating direction [Shah et al., 2016], and specialize them for the K Disjoint Clique problem. For each technique we develop an algorithm. For each algorithm, we provide a per-iteration complexity analysis to estimate computational cost of each. With each of these methods offering an improvement on the classical interior point method without loss of accuracy in the solution. We will verify this with numerical simulations and results.

The motivation behind our works stems from the high cost per iteration of interior point methods, and their inability to scale to large scale problems. We expand briefly on the techniques we will use to solve the K Disjoint Clique problem. First, Alternating Direction Method of Multipliers uses a linear splitting to simplify the approximation of the dual gradient. We minimize the augmented Lagrangian with respect to each variable in succession followed by updates of dual variables. Next, we use a low rank factorization to relax the K Disjoint Clique problem, and solve using the augmented Lagrangian. Lastly, we rewrite the K Disjoint Clique problem as a Biconvex problem and solve using alternating direction.

In Chapter 2 we give a brief overview of semidefinite programming, the motivation of our work, and a short review of interior point methods. In Chapter 3, we provide a detailed description of the K Disjoint Clique problem with an SDP relaxation, and recovery guarantees for the K Disjoint Clique problem. In Chapter 4, we detail the use of Alternating Direction Method of Multipliers, and provide and justify an algorithm. In Chapter 5, we describe a low rank factorization and the Method of Multipliers, and provide an algorithm. In Chapter 6, we outline a Biconvex relaxation using alternating direction, and provide an algorithm. In Chapter 7, we

expound on numerical simulations of our algorithms, and formulate future work. In Chapter 8, we summarize results, and contributions. Finally, in Chapter 8, we provide justification of complexities of algorithms, and provide figures for results.

CHAPTER 2

SEMIDEFINITE PROGRAMMING

We begin by introducing semidefinite programming, SDP, to serve as background knowledge for the rest of the dissertation. SDP is a relatively new exciting branch of convex optimization with growing popularity due to its many applications including convex constrained optimization, control theory, and combinatoral optimization [Freund, 2004]. The general form includes an objective function, linear constraints, and a positive semidefinite constraint on our matrix variable. We want to minimize or maximize the objective function relative to the linear constraints, while ensuring our matrix variable belongs to the cone of positive semidefiniteness. The general form would be:

$$\begin{aligned} \min \ & tr(\boldsymbol{CX}) \\ \text{s.t.} \ & tr(\boldsymbol{A_i X}) \leq b_i, \ \text{for } i = 1, ..., m \\ & \boldsymbol{X} \succeq 0, \end{aligned} \tag{2.1}$$

where $\boldsymbol{C}$, $\boldsymbol{A_i}$ for $i = 1, ..., m$ are data matrices, $\boldsymbol{b}$ is a data vector, m is the number of linear constraints, $tr(\boldsymbol{CX}) = \sum_{i=1}^{N} [CX]_{ii}$ is the sum of the diagonal elements for the resulting matrix or trace inner product, and $\boldsymbol{X} \succeq 0$ means that $\boldsymbol{X}$, our matrix variable, belongs to the positive semidefinite cone [Freund, 2004]. A matrix $\boldsymbol{X} \in \mathbf{R}^{N \times N}$ is positive semidefinite, written $\boldsymbol{X} \succeq 0$, if for any $\boldsymbol{v} \in \mathbf{R}^N$ then $\boldsymbol{v}^T \boldsymbol{X} \boldsymbol{v} \geq 0$, and $\boldsymbol{X}$ is symmetric meaning $\boldsymbol{X} = \boldsymbol{X}^T$. If we can replace the $\geq$ with $>$ then $\boldsymbol{X} \in \mathbf{R}^{N \times N}$ is positive definite, written $X \succ 0$. That is $\boldsymbol{X}$ is positive definite if $\boldsymbol{v}^T \boldsymbol{X} \boldsymbol{v} > 0$ for any $\boldsymbol{v} \in \mathbf{R}^N$ [Boyd and Vandenberghe, 2004, Chapter 2]. We let Σ^N be the set of N by N matrices that are symmetric, and let Σ_+^N be the set of N by N matrices that are

positive semidefinite and symmetric. Finally, Σ_{++}^N will denote the set of matrices that are positive definite and symmetric. As we will be maximizing or minimizing over these sets it is nice to note that they are convex closed cones under addition and scalar multiplication.

Lemma 1. *The positive semidefinite cone, Σ_{+}^N, is a closed convex cone under addition and scalar multiplication.*

Proof. Let $\boldsymbol{X}$, and $\boldsymbol{Y}$ be positive semidefinite, $\boldsymbol{X} \in \mathcal{S}_{+}^N$ and $\boldsymbol{Y} \in \Sigma_{+}^N$, and $\alpha \geq 0$, and $\beta \geq 0$ be scalars, then for an $\boldsymbol{v} \in \mathbf{R}^N$ we have:

$$\begin{aligned} \boldsymbol{v}^T(\alpha \boldsymbol{X} + \beta \boldsymbol{Y})\boldsymbol{v} &= \boldsymbol{v}^T(\alpha \boldsymbol{X})\boldsymbol{v} + \boldsymbol{v}^T(\beta \boldsymbol{Y})v \\ &= \alpha \boldsymbol{v}^T \boldsymbol{X} \boldsymbol{v} + \beta \boldsymbol{v}^T \boldsymbol{Y} \boldsymbol{v} \\ &\succeq 0, \end{aligned} \tag{2.2}$$

by knowledge that $\boldsymbol{X},\ \boldsymbol{Y} \in \Sigma_{+}^N$. □

By a similar argument, you can also see that Σ_{++}^N is convex cone. Now we state a very important fact regarding symmetric matrices and its eigenvalue decomposition [Cherney et al., 2016, Chapter 15].

Lemma 2. *Every symmetric matrix $\boldsymbol{X}$ is similar to a diagonal matrix of its eigenvalues. That is matrix $\boldsymbol{X} = \boldsymbol{X}^T$ if and only if $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$ where $\boldsymbol{E}$ is an orthogonal matrix, whose columns are eigenvectors of $\boldsymbol{X}$, and $\boldsymbol{D}$ is a diagonal matrix whose entries are the eigenvalues of $\boldsymbol{X}$.*

Another important fact to state is that $\boldsymbol{X} \succeq 0$ if and only if it has eigenvalues greater than or equal to zero [Freund, 2004, Section 3].

Lemma 3. *The matrix $\boldsymbol{X} \succeq 0$ if and only if $D_{ii} \geq 0$ for $i = 1, ..., N$ for spectral decomposition $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^{\boldsymbol{T}}$. Moreover the matrix $\boldsymbol{X} \succ 0$ if and only if $D_{ii} > 0$ for $i = 1, ..., N$ for spectral decomposition $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^{\boldsymbol{T}}$.*

Lemma 4. *$\boldsymbol{X} \succ 0$ if and only if $D_{ii} > 0$ for $i = 1, ..., N$ for spectral decomposition $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$.*

Now that we have defined a SDP in its primal form we consider the dual form, SDD, of the standard SDP (2.1):

$$\begin{aligned} \max \quad & \sum_{i=1}^{m} y_i b_i \\ \text{s.t.} \quad & \sum_{i=1}^{m} y_i \boldsymbol{A_i} + \boldsymbol{S} = \boldsymbol{C} \\ & \boldsymbol{S} \succeq 0, \end{aligned} \tag{2.3}$$

where $\boldsymbol{y}$ is an m vector which is the dual variable and $\boldsymbol{S}$ is an N by N matrix. One key fact is that if the objective of SDP (2.1) and SDD (2.3) are equal, $tr(\boldsymbol{C}\boldsymbol{X}) = \sum_{i=1}^{m} y_i b_i$ then $\boldsymbol{X}$ and $(\boldsymbol{y}, \boldsymbol{S})$ are optimal for their problem and we have a duality gap of 0 [Freund, 2004].

Lemma 5. *Given a feasible solution $\boldsymbol{X}$ of (2.1) and feasible solution pair $(\boldsymbol{y}, \boldsymbol{S})$ of (2.3) then the duality gap, $tr(\boldsymbol{S}^T\boldsymbol{X}) \geq 0$, if $tr(\boldsymbol{S}^T\boldsymbol{X}) = 0$ then $\boldsymbol{X}$ and $(\boldsymbol{y}, \boldsymbol{S})$ are optimal for their respective problem and $\boldsymbol{S}\boldsymbol{X} = 0$.*

Proof. For the proof of Lemma 1 we refer the reader to Proof of Proposition 5.1 in [Freund, 2004]. □

SDP's have many applications including convex optimization, combinatorial optimization, machine learning, computer vision, and control theory. Linear programs can be viewed as SDP, with vectors stored in the diagonal of diagonal matrices, which would serve as our data matrices. Though this is impractical for solving the thought shows how SDP's can be applied to a wide range of problems. A nonexhaustive list of examples includes Convex Quadratically Constrained Quadratic Programs (Convex QCQP), the Max Cut problem, Smallest Circumscribed Ellipsoid problem, Largest Inscribed Ellipsoid problem, Eigenvalue Optimization, matrix norm minimization,

image segmentation, matrix completing, structural optimization, statistics, kernel matrix learning, and the K Disjoint Clique problem, which this paper focuses on [Freund, 2004, Boyd et al., 2011, Vandenberghe and Boyd, 1996].

2.1 Interior Point Methods for SDP

One of the most widely used and well known methods for solving SDP's is interior point methods using barrier functions such as,

$$\phi(\boldsymbol{X}) = -\log(\det(\boldsymbol{X})), \tag{2.4}$$

where $\phi : \Sigma^N_{++} \to \mathbf{R}$ has domain of the cone of positive definite matrices. One thing of note is that ϕ is a strictly convex function as the Hessian is positive definite. Also as we will be discussing the central path for interior point methods, we mention that for any sequence of $\boldsymbol{X}^{(k)} \in \Sigma^N_{++}$ converging to a boundary point, $\boldsymbol{X}^{(n)} \in \Sigma^N_{+}$ the value of $\phi(\boldsymbol{X}^{(n)}) \to \infty$ [Tuncel, 2016].

We now discuss interior point methods, and their challenges for large scale SDP. Interior point methods to solve (2.1) by solving a sequence of minimizations of convex functions consisting of the linear objective function, $tr(\boldsymbol{CX})$, and a barrier function ϕ, while optimizing over the set of linear constraints, $tr(\boldsymbol{A_i X}) \leq b_i$ for $i = 1, .., m$. Though we have defined a sample barrier function above (2.4), we now let $\phi(\boldsymbol{X}) : \Sigma^N_{++} \to \mathbf{R}$ be general smooth barrier function, i.e. a smooth penalty function, with domain of ϕ is the cone of positive definite matrices, Σ^N_{++}. This yields the central path or centering problem:

$$\begin{aligned} \min \;\; & tr(\boldsymbol{CX}) + t\phi(\boldsymbol{X}) \\ \text{s.t} \;\; & tr(\boldsymbol{A_i X}) = b_i \; \forall i = 1, ..., m, \end{aligned} \tag{2.5}$$

where $\boldsymbol{C}$, and $\boldsymbol{A_i}$ are data matrices, $\boldsymbol{b} \in \mathbf{R}^m$ is a data vector, ϕ is our barrier function, t is the barrier parameter, which increases, and $\boldsymbol{X}$ is our matrix variable. The domain

of (2.5) is the intersection of that of (2.1) and Σ_{++}^N, and interior point methods solve (2.1) by creating a sequence of iterates $\boldsymbol{X}^{(k)}$ until the duality gap is below a desired tolerance, say ϵ. Each successive iterate is computed using Newton-type search directions trying to find another feasible solution while maintaining positive definiteness. Interior point methods exhibit fast convergence to desired tolerance of ϵ in a few number of steps specifically within $\mathcal{O}(\sqrt{N}\log(\frac{1}{\epsilon}))$ for any $\epsilon > 0$ [Boyd and Vandenberghe, 2004, Vandenberghe and Boyd, 1996].

2.2 Per-Iteration Complexity of Interior Point Methods

However, even with a minimal number of steps required for convergence, interior point methods are expensive, depending on m and N, and do not scale well for large scale problems since their per cost iteration can be large. To expand on this consider the dual problem (2.3) and consider the central path or centering problem on the dual:

$$\min\ t\boldsymbol{b}^T\boldsymbol{y} - \log(\det(\boldsymbol{C} - \sum_{i=1}^{m} y_i\boldsymbol{A_i})) \tag{2.6}$$

where our data matrices $\boldsymbol{C}, \boldsymbol{A} \in \mathbf{R}^{N\times N}$, $\boldsymbol{b} \in \mathbf{R}^m$ an m dimensional data vector, $\boldsymbol{y} \in \mathbf{R}^m$ is the dual variable, and our barrier function $\phi(\boldsymbol{X}) = -\log(\det(\boldsymbol{X}))$. To solve (2.6) we need to take a Newton direction step from iterate $\boldsymbol{y}$ for which we will need to calculate the gradient, Hessian, dual pair of $\boldsymbol{y}$, $\boldsymbol{S} = \boldsymbol{C} - \sum_{i=1}^m y_i\boldsymbol{A_i}$, $\boldsymbol{S}^{-1}$ for calculating the Hessian, and products of $\boldsymbol{S^{-1}A_i}$ for $i = 1, ..., m$ for calculating the Hessian. Now looking at required flop counts for each of these. First we will need $\boldsymbol{S}$ at a cost of $\mathcal{O}(N^2m)$, the products of $\boldsymbol{S}^{-1}$ and data matrices $\boldsymbol{A_i}$ at a cost of $\mathcal{O}(N^3m)$, all Hessian entries at a cost of $\mathcal{O}(N^2)$ each for m choose 2 entries, and the actual Newton step solve costing $\mathcal{O}(m^3)$. From here we can see the cost per iteration of solving the (2.6) would be $\mathcal{O}(\max\{(N^3m, N^2m^2, m^3\})$ flops which does not scale well for large N or m.

Due to the challenges associated with applying interior point methods and large-scale problems, much recent attention has been paid to the development of first

order methods for SDP's. These include interior point methods using barrier functions, first order methods, penalization or augmented Lagrangian, biconvex relaxation, and alternating direction method of multipliers [Boyd et al., 2011, Burer and Monteiro, 2003, Shah et al., 2016, Tu and Wang, 2014, Burer and Monteiro, 2005, Renegar, 2014, Zheng and Lafferty, 2015]. The focus of the remained of the dissertation will be the development of efficient and scaleable methods for SDP's in particular the K Disjoint Clique problem.

CHAPTER 3

K DISJOINT CLIQUE PROBLEM

We begin by detailing clustering and its wide variety of applications including machine learning and statistics. Let us first start with the general goal of clustering and some definitions. The goal of the clustering optimization problem is to take data, and partition it so that the objects in a group, called a cluster, are similar, and objects that are not in a group are not similar to objects in the cluster. Clustering is generally modeled as a combinatoral optimization problem in which certain characteristics of clusters are required. Some characteristics of clustering problems include, complete clusters, or cliques, disjoint clusters, or number of clusters as in the maximum node K disjoint clique problem. Clustering can be applied to information retrieval, pattern recognition, computational biology, and image processing. However as our data set to partition becomes large the clustering problem is time consuming and memory exhaustive while using interior point methods to solve. For an overview in the clustering problem and its applications we refer the reader to [Berkhin, 2006, Mirkin, 1998, Peng, 2005, Peng and Wei, 2007].

The K Disjoint Clique combinatorial optimization problem seeks K larger cliques, or complete subgraphs, that include as much of a given data set as possible. A clique is a complete subgraph of G where all nodes in the clique are similar. The data is modeled as a graph G where the edges indicate similarity or adjacency. We will assign an edge in the graph between any pair of similar nodes. If end points are not similar then they are not adjacent. The K Disjoint Clique problem would be equivalent to partitioning G into K clusters of similar objects, which we will refer to as cliques plus a set of outliers. The K Disjoint Clique problem can be modeled as a

combinatorial optimization problem:

$$\max_{S=\{\boldsymbol{x}_1,...,\boldsymbol{x}_K\}} \quad \sum_{i=1}^{K} \boldsymbol{x}_i^T \boldsymbol{e} \tag{3.1}$$

$$\text{s.t.} \quad \boldsymbol{x}_i^T \boldsymbol{x}_j = 0, \quad \forall \ i \ , \ j = 1, ..., K \ \ , \ i \neq j \tag{3.2}$$

$$[\boldsymbol{x}_i \boldsymbol{x}_i^T]_{uv} = 0, \quad \text{if } uv \notin E, \ u \neq v, \ \ \forall \ i = 1, ..., K \tag{3.3}$$

$$\boldsymbol{x}_i \in \{0,1\}^V, \ \ \forall i = 1, ..., K \tag{3.4}$$

where $\boldsymbol{e}$ is the set of all ones in $\mathbf{R}^N$, E is the set of edges for our graph G, and $S = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_K\}$ is the collection of characteristic vectors of a set of disjoint cliques of G [Ames and Vavasis, 2014].

We next review (3.1) in more detail, which will allow us to see how we are be able to relax this combinatorial optimization problem to a semidefinite program. Let us begin with the objective function, (3.1). This is is simply the addition of all nodes included in our K cliques represented by the set of characteristic vectors S. So we want to maximize this or maximize the number of nodes contained in the optimal K disjoint clique subgraph, meaning the same node can not be two in different cliques. However, it is clear that not all nodes may be included in our K cliques as some nodes may be an outlier. If we move on to the first constraint (3.2), this requires the dot product of characteristic vectors representing each clique to be zero. This constraint ensures that no node is in more than one clique. Continuing, we now look at the second constraint (3.3), which forces us to work with our set E and not add edges to complete a subgraph and form a clique. Our goal is to find cliques within a certain set of data and constraint (3.3) enforces this by restricting our data to the original data. Lastly, we can look at the final constraint (3.4), which can be view as an integer constraint. It forces each element in $\boldsymbol{x}$ to be either 0 or 1, clustering the nodes based on similarity. If a node has value 1 in $\boldsymbol{x}_i$ then it is in clique i and similar to other nodes in this clique.

A rank constrained SDP relxation of (3.1) is derived in [Ames and Vavasis,

2014]:

$$
\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} \\
\text{s.t.} \quad & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{e}, \\
& X_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j \\
& \text{rank}(\boldsymbol{X}) = K \\
& \boldsymbol{X} \succeq 0,
\end{aligned}
\tag{3.5}
$$

where N is equal to the cardinally of $\boldsymbol{x}$ or number of nodes in the data set, $\boldsymbol{X}$ is an N by N matrix, $\boldsymbol{e} \in \mathbf{R}^N$ is the vector of all ones, E is the set of edges from graph G, and K is the desired number of cliques. We now reference work from [Ames and Vavasis, 2014] to show it is a relaxation of (3.1). To show that (3.5) is a relaxation of (3.1) and understand its derivation consider a solution set of characteristic vectors from (3.1) $S = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_K\}$ and form rank-one representation of C_i for each $i = 1, .., K$ by $C_i = \boldsymbol{x}_i \boldsymbol{x}_i^T$. Then normalize each C_i by cluster size and add together then directly from [Ames and Vavasis, 2014] it follows that every feasible solution S of (3.1) is a feasible solution $\boldsymbol{X}$ of (3.5) and in both cases the objective function is equal to number of nodes in the K disjoint cliques. From here we can relax the rank constraint $\text{rank}(\boldsymbol{X}) = K$ to the trace constraint $tr(\boldsymbol{X}) = K$ to get:

$$
\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} \\
\text{s.t.} \quad & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{e}, \\
& X_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j \\
& tr(\boldsymbol{X}) = K \\
& \boldsymbol{X} \succeq 0,
\end{aligned}
\tag{3.6}
$$

from [Ames and Vavasis, 2014]. Suppose G contains cliques $C_1, ..., C_K$ and we have clique sizes $c_1, ..., c_K$ for $l = 1, ..., K$ cliques in a K disjoint clique subgraph of G. Then we can define, $\boldsymbol{X}^*$ a feasible solution to (3.6):

$$X^*_{ij} = \begin{cases} 1/c_l & \text{if both } i, j \text{ belong to clique } l,\ C_l \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Theorem 2.1 from [Ames and Vavasis, 2014], give conditions based on the Karush-Kuhn-Tucker conditions, KKT, [Boyd and Vandenberghe, 2004] for which (3.7) is the optimal solution for (3.6).

That is, Ames and Vavasis in [Ames and Vavasis, 2014] determine two cases in which we are guaranteed that (3.7), the feasible solution to (3.6) our relaxation of (3.1), is optimal for (3.6) and the graph, G^* from the same cliques is the optimal minimizer to (3.1). Before we get to those two theorems, let us first discuss how our G is created in each theorem. Recall that $G = [V, E]$ in which we add K disjoint cliques, $C_1, C_2, ..., C_K$ with sizes $c_1, c_2, ..., c_K$. These cliques are the cliques we aim to recover in our relaxation (3.6) of the K Disjoint Clique problem. C_{K+1} of set vertices and edges not in $C_1, ..., C_K$ will also be added to our K disjoint clique subgraph. We will refer to this graph G as the planted K clique disjoint graph.

Stating Theorem 3.1 from [Ames and Vavasis, 2014], we have the following recovery guarantee where the number of added edges can be up to $\mathcal{O}(\hat{c}^2)$ with $\hat{c} := \min\{c_1, c_2, ..., c_K\}$:

Theorem 1. *Consider an instance of the K Disjoint Clique problem constructed according to the preceding description, namely, G contains a K-disjoint-clique graph G^* whose nodes are partitioned as $C_1 \cup ... \cup C_K$ where $|C_q| = c_q,\ q = 1, ..., K$, plus additional nodes denoted C_{K+1} and additional edges (which may have endpoints chosen from any of $C_1, ..., C_{K+1}$). Assume the following conditions are satisfied:*

1. *For all* $q = 1, ..., K,\ i \in C_q,\ \forall s \in \{1, ..., K+1\} - q,$

$$n_i^s \leq \delta\ \min(c_q, c_s).$$

Here, δ *is a scalar satisfying* $0 < \delta < (1-\delta)^2$.

2. $|E(G) \setminus E(G^*)| \leq \rho \hat{c}^2$, *where* ρ *is a positive scalar depending on* δ.

Then $\boldsymbol{X}^*$ *given by* (3.7) *is the unique optimal solution to* (3.6)*, and* G^* *is the unique optimal solution of the* K *Disjoint Clique problem.*

This is known as the adversarial case in [Ames and Vavasis, 2014], where someone is choosing to add up to $\mathcal{O}(\hat{c}^2)$ edges to our planted K disjoint clique graph to potentially prevent the SDP relaxation (3.6) from recovering the planted K clique subgraph.

Now will we detail the randomized case, which has stronger theoretical guarantees, from [Ames and Vavasis, 2014] in which G is generated slightly differently. We will have $C_1, C_2, ..., C_{K+1}$ disjoint sets of vertices with sizes $c_1, c_2, ..., c_K$ with V equal to the union of all vertices and graph $G = [V, E]$ and we construct the edge set based on the following.

1. We fill in our first K cliques.

2. Each of the remaining possible edges is added to E independently at random with probability $p \in (0, 1)$.

Stating Theorem 4.5 from [Ames and Vavasis, 2014] we have the following recovery guarantee which relies heavily on Theorem 2.2 in the same paper.

Theorem 2. *Suppose that* $G = [V, E]$ *has a* K*-disjoint-clique subgraph* G^* *composed of the cliques* $C_1, C_2..., C_K$ *and let* $C_{K+1} := V \setminus (\cup_{i=1}^K C_i)$. *Let* $c_i = |C_i|$ *for all* $i = 1, 2, ..., K+1$ *and suppose that* $c_q \leq \hat{c}^{3/2}$ *for all* $q = 1, 2, ..., K$ *where*

$\hat{c} = \min\{c_1, c_2, ..., c_K\}$. *Then there exists some* $\beta_1, \beta_2 > 0$ *depending only on* p *such that if*

$$\beta_1 \left(\sum_{s=1}^{K} c_s^2 \right)^{1/2} \left(\sum_{q=1}^{K} \frac{1}{c_q} \right)^{1/2} + \beta_2 \sqrt{N} \leq \hat{c},$$

then $\boldsymbol{X}^*$ *given by* (3.7) *is the unique optimal solution to* (3.6)*, and* G^* *is the unique optimal solution of the* K *Disjoint Clique problem with high probability.*

For detailed explanation on this theorem and examples of cases that fulfill the above theorem refer to [Ames and Vavasis, 2014] Section 4 The Randomized Case. We notice that recovery is better in the random case, which we will work with in examples, rather than the adversarial case, because the pathological cases forbidden by the adversarial Theorem happen with very low probability in the random case.

Recent papers including, [Abbe et al., 2016], [Ailon et al., 2013], [Ames, 2014], [Ames and Vavasis, 2014], [Amini and Levina, 2014], [Cai et al., 2015], [Chen and Xu, 2014], [Chen et al., 2014b], [Chen et al., 2014a], [Guédon and Vershynin, 2015], [Hajek et al., 2015], [Lei et al., 2015], [Mathieu and Schudy, 2010], [Nellore and Ward, 2013], [Oymak and Hassibi, 2011], [Rohe et al., 2011], [Qin and Rohe, 2013], [Vinayak et al., 2014], have shown that if data is sampled from some distribution of clusterable data we can efficiently recover the clusters. In most cases the data is assumed to be sampled from some generalization of stochastic block models [Holland et al., 1983]. Our formulation of the K Disjoint Clique problem, (3.1), falls under this generalization as in [Ames and Vavasis, 2014]. Though the work on recovery is important, the real world applications of these data sets are limited since most algorithmic approaches for these convex relaxations are not tractable for large data sets..

As technology and data continue to grow the world becomes more data driven, there is an increased need for us to be able to solve the K Disjoint Clique problem accurately and efficiently for large data sets. However, most current algorithmic

approaches, such as interior point methods described in Chapter 2, do not scale well for large scale problems. Specifically work from [Ames and Vavasis, 2014] shows that the K Disjoint Clique problem can be solved in polynomial time with recovery under the conditions provided by Theorems 1 and 2 using interior point methods. However using interior point methods, where per iteration flop counts are $\mathcal{O}(\max\{N^3m, N^2m^2, m^3\})$ for m linear constraints and matrices which are N by N, do not scale well for our K Disjoint Clique problem. We would expect flop counts $\mathcal{O}(N^6)$ after adding slack variables to deal with inequality constraints, since we would have $m = \mathcal{O}(N^2)$ linear constraints. This greatly limits the size of data sets we can cluster, and it prompts the question of if there is a more efficient way. In the following sections we will explore three different methods for solving (3.6), specifically the ADMM, low-rank factorization and the method of multipliers, and biconvex relaxation and block coordinate descent, motivated by the recent approaches in [Burer and Monteiro, 2003, Boyd et al., 2011, Shah et al., 2016], with a goal of decreasing flop counts per iteration while not drastically increasing the number of iterations.

CHAPTER 4

ALTERNATING DIRECTION METHOD OF MULTIPLIERS

In this section we will discuss how the method of Alternating Direction Method of Multipliers, which was first credited to Gabay, Mercier, Glowinski, and Marrocco in the 1970s [Gabay and Mercier, 1976, Gabay, 1983, Glowinski and Marroco, 1975], can be used to solve the K disjoint clique problem.

4.1 Method of Multipliers

Let us begin by reviewing some of the background information on the method of multipliers or augmented Lagrangian as it is background to ADMM. The Method of Multipliers was introduced in the 1960s by Hestenes and Powell [Hestenes, 1969b, Powell, 1969, Hestenes, 1969a]. Consider the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbf{R}^N} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{4.1}$$

then we can form the following augmented Lagrangian:

$$\mathcal{L}_\sigma(\boldsymbol{x}, \boldsymbol{y}) = f(\boldsymbol{x}) + \boldsymbol{y}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \frac{\sigma}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2 \tag{4.2}$$

with $\boldsymbol{A} \in \mathbf{R}^{M\times N}$, $\boldsymbol{x} \in \mathbf{R}^N$, $\boldsymbol{b} \in \mathbf{R}^M$, $\boldsymbol{y} \in \mathbf{R}^M$ the dual variable, and $\sigma > 0$ the penalty parameter. This is the Lagrangian of (4.1) with an extra penalty term including σ. It is clear that minimizing the augmented Lagrangian over $\boldsymbol{x}$ and $\boldsymbol{y}$ is equivalent to our previous optimization problem since any feasible $\boldsymbol{x}$ is equivalent to evaluating the objective function at $\boldsymbol{x}$. Since the last two terms would be equal to zero for $\boldsymbol{x}$ that is feasible in problem (4.1). The reason for using the augmented Lagrangian instead of

the Lagrangian is to ensure the dual problem is differentiable under rather general conditions compared to the original optimization problem because of the penalty term which creates a smoother function [Boyd et al., 2011]. If we summarize [Boyd et al., 2011, Section 2.3], and apply dual ascent to (4.2) we get the update algorithm:

$$\begin{aligned} \boldsymbol{x}^{k+1} &= \arg\min_{\boldsymbol{x}\in\mathbf{R}^N} \mathcal{L}_\sigma(\boldsymbol{x},\boldsymbol{y}) \\ \boldsymbol{y}^{k+1} &= \boldsymbol{y}^k + \sigma(\boldsymbol{A}\boldsymbol{x}^{k+1} - \boldsymbol{b}). \end{aligned} \tag{4.3}$$

This is the general form of method of multipliers or augmented Lagrangian. Which is similar to dual ascent with the $\boldsymbol{x}$ update based on the augmented Lagrangian, and step size of σ in the update of our dual variable $\boldsymbol{y}$. Augmented Lagrangian converges to the solution of optimization problem (4.1) under less strict conditions than dual ascent which is part of our motivation for using it.

4.2 Alternating Direction Method of Multipliers

Let us now discuss some general background information about Alternating Direction Method of Multipliers, ADMM. ADMM is an algorithm which was derived from the decomposition of dual ascent and the superior convergence of method of multipliers. It can solve problems where the objective function, and constraints can be split over multiple variables [Boyd et al., 2011]. In our case this is the splitting of variable $\boldsymbol{X}$ into variables $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$. This splitting allows us to use alternating direction when minimizing the augmented Lagrangian with the method of multipliers before updating our dual variables with dual ascent. Each iterate of Algorithm 1, which will be given below, will consist of $\boldsymbol{Z}$ and dual variables $\boldsymbol{\Lambda_{XY}}$, $\boldsymbol{\Lambda_{XZ}}$, and $\boldsymbol{\Lambda_{YZ}}$ with $\boldsymbol{X}$, and $\boldsymbol{Y}$ being intermediate steps to achieve this state.

Now we recall our semidefinite program (3.6) from Chapter 3 that is a relaxation of the K disjoint clique problem:

$$
\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} \\
\text{s.t.} \quad & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{e}, \\
& X_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j \\
& tr(\boldsymbol{X}) = K \\
& \boldsymbol{X} \succeq 0,
\end{aligned}
$$

where $\boldsymbol{e} \in \mathbf{R}^N$ is the vector of ones, our graph G is N by N, E is the set of edges in our graph G, and K is the desired number of clusters. We used a three way splitting of our main variable $\boldsymbol{X}$ into $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ by adding the constraints $\boldsymbol{X} = \boldsymbol{Y}$, $\boldsymbol{X} = \boldsymbol{Z}$, and $\boldsymbol{Y} = \boldsymbol{Z}$. Along with modifying the current constraints to these new constraints $\boldsymbol{Y}\boldsymbol{e} \leq \boldsymbol{e}$, $\boldsymbol{Y} = \boldsymbol{Y}^T$, $Z_{ij} = 0 \ \ \forall\, (i,j) \notin E$ s.t. $i \neq j$, $tr(\boldsymbol{X}) = K$, and $\boldsymbol{X} \succeq 0$. We also rewrite our objective function in terms of $\boldsymbol{Y}$ to give us the new equivalent semidefninite program:

$$
\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} Y_{ij} \\
\text{s.t.} \quad & \boldsymbol{Y}\boldsymbol{e} \leq \boldsymbol{e}, \ \boldsymbol{Y} = \boldsymbol{Y}^T \\
& Z_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j \\
& tr(\boldsymbol{X}) = K \\
& \boldsymbol{X} \succeq 0 \\
& \boldsymbol{X} = \boldsymbol{Y} \ , \ \boldsymbol{X} = \boldsymbol{Z} \ , \ \boldsymbol{Y} = \boldsymbol{Z},
\end{aligned}
\tag{4.4}
$$

with $\boldsymbol{Y}$ and $\boldsymbol{Z}$ N by N matrices as well.

Algorithm 1: Alternating Direction Method of Multipliers Algorithm

Data: Input $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, $\boldsymbol{\Lambda_{XY}}$, $\boldsymbol{\Lambda_{XZ}}$, $\boldsymbol{\Lambda_{YZ}}$, β

for *Until Converged*

$$\boldsymbol{X}^{k+1} = \underset{tr(\boldsymbol{X})=K\ ,\ \boldsymbol{X}\succeq 0}{\arg\min} \mathcal{L}_\beta(\boldsymbol{X}, \boldsymbol{Y}^k, \boldsymbol{Z}^k, \boldsymbol{\Lambda_{XY}}^k, \boldsymbol{\Lambda_{XZ}}^k, \boldsymbol{\Lambda_{YZ}}^k)$$

$$\boldsymbol{Y}^{k+1} = \underset{\boldsymbol{Ye}\leq \boldsymbol{e}\ ,\ \boldsymbol{Y}^T=\boldsymbol{Y}}{\arg\min} \mathcal{L}_\beta(\boldsymbol{X}^{k+1}, \boldsymbol{Y}, \boldsymbol{Z}^k, \boldsymbol{\Lambda_{XY}}^k, \boldsymbol{\Lambda_{XZ}}^k, \boldsymbol{\Lambda_{YZ}}^k)$$

$$\boldsymbol{Z}^{k+1} = \underset{\boldsymbol{Z}_{ij}=0,\ \boldsymbol{Z}_{ji}=0\ \forall\ (i,j)\notin E \text{ s.t } i\neq j}{\arg\min} \mathcal{L}_\beta(\boldsymbol{X}^{k+1}, \boldsymbol{Y}^{k+1}, \boldsymbol{Z}, \boldsymbol{\Lambda_{XY}}^k, \boldsymbol{\Lambda_{XZ}}^k, \boldsymbol{\Lambda_{YZ}}^k)$$

$$\boldsymbol{\Lambda_{XY}}^{k+1} = \boldsymbol{\Lambda_{XY}}^k - \beta(\boldsymbol{X}^{k+1} - \boldsymbol{Y}^{k+1})$$

$$\boldsymbol{\Lambda_{XZ}}^{k+1} = \boldsymbol{\Lambda_{XZ}}^k - \beta(\boldsymbol{X}^{k+1} - \boldsymbol{Z}^{k+1})$$

$$\boldsymbol{\Lambda_{YZ}}^{k+1} = \boldsymbol{\Lambda_{YZ}}^k - \beta(\boldsymbol{Y}^{k+1} - \boldsymbol{Z}^{k+1})$$

end

Data: Output $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$

4.3 ADMM Algorithm

Let us now create the augmented Lagrangian using the equality constraints on variables $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ and use the other constraints as constraints during the update process. We get the following augmented Lagrangian for (4.4):

$$\begin{aligned}\mathcal{L}_\beta(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z}, \boldsymbol{\Lambda_{XY}}, \boldsymbol{\Lambda_{XZ}}, \boldsymbol{\Lambda_{YZ}}) = &-\boldsymbol{ee}^T\boldsymbol{Y} \\ &+ tr(\boldsymbol{\Lambda_{XY}}(\boldsymbol{X}-\boldsymbol{Y})) + tr(\boldsymbol{\Lambda_{XZ}}(\boldsymbol{X}-\boldsymbol{Z})) + tr(\boldsymbol{\Lambda_{YZ}}(\boldsymbol{Y}-\boldsymbol{Z})) \\ &+ \frac{\beta}{2}(||\boldsymbol{X}-\boldsymbol{Y}||_F^2 + ||\boldsymbol{X}-\boldsymbol{Z}||_F^2 + ||\boldsymbol{Y}-\boldsymbol{Z}||_F^2),\end{aligned} \tag{4.5}$$

where β is the penalty parameter. We will use the Alternating Direction Method of Multipliers which uses both dual ascent, and augmented Lagrangain to solve (4.4). This will give us the general framework of our Algorithm 1.

We will now take some time to discuss each one of the update steps above in our algorithm. Let us first start with the update of our variable $\boldsymbol{X}$. Looking back at our augmented Lagrangian, (4.5), we can reduce our terms by only considering the terms containing $\boldsymbol{X}$ giving us the following problem to solve:

$$\boldsymbol{X} := \underset{tr(\boldsymbol{X})=K\ ,\ \boldsymbol{X}\succeq 0}{\arg\min} tr(\boldsymbol{\Lambda_{XY}}\boldsymbol{X}) + tr(\boldsymbol{\Lambda_{XZ}}\boldsymbol{X}) + \frac{\beta}{2}tr(\boldsymbol{XX}^T - 2\boldsymbol{XY} + \boldsymbol{XX}^T - 2\boldsymbol{XZ}). \tag{4.6}$$

By factoring out β then separating our terms quadratic in $\boldsymbol{X}$ and linear in $\boldsymbol{X}$ we are able to further simplify our problem to:

$$\boldsymbol{X} := \underset{tr(\boldsymbol{X})=K\ ,\ \boldsymbol{X}\succeq 0}{\arg\min} \beta tr(\boldsymbol{X}\boldsymbol{X}^T - \boldsymbol{X}^T\boldsymbol{U}_{\boldsymbol{X}}),$$

with $\boldsymbol{U}_{\boldsymbol{X}} = \frac{1}{2}(\frac{1}{\beta}\boldsymbol{\Lambda}_{\boldsymbol{XY}} + \frac{1}{\beta}\boldsymbol{\Lambda}_{\boldsymbol{XZ}} - \boldsymbol{Y} - \boldsymbol{Z})$. Then we can use the fact that $||\boldsymbol{A}||_F^2 = tr(\boldsymbol{A}\boldsymbol{A}^T)$ [Petersen et al., 2008], while still only considering the terms containing $\boldsymbol{X}$ to get

$$\boldsymbol{X} := \underset{tr(\boldsymbol{X})=K\ ,\ \boldsymbol{X}\succeq 0}{\arg\min} ||\boldsymbol{X} - \boldsymbol{U}_{\boldsymbol{X}}||_F^2. \tag{4.7}$$

We now see that the optimal $\boldsymbol{X}$ for this update (4.6) is the $\boldsymbol{X}$ that is the projection of $\boldsymbol{U}_{\boldsymbol{X}}$ onto the set of $\boldsymbol{X}$ where $\boldsymbol{X} \succeq 0$ and $tr(\boldsymbol{X}) = K$. To detail this more consider $\boldsymbol{E}$ an N by N matrix that has the eigenvectors as columns and $\boldsymbol{D}$ is the diagonal matrix of corresponding eigenvalues found from the spectral decomposition of $\boldsymbol{U}_{\boldsymbol{X}}$ such that $\boldsymbol{U}_{\boldsymbol{X}} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$. From here we will use this spectral decomposition to find $\boldsymbol{X} = \boldsymbol{E}\hat{\boldsymbol{D}}\boldsymbol{E}^T$ and make sure all eigenvalues are greater than or equal to zero to find $\boldsymbol{X}$. As well as shifting them by λ so the $tr(\boldsymbol{X}) = K$. This update has a solution of $\boldsymbol{X} = \boldsymbol{E}\hat{\boldsymbol{D}}\boldsymbol{E}^T$ where $\hat{\boldsymbol{D}}$ is a diagonal matrix defined by $\hat{D}_{ii} = d_i$ with $d_i = \max\{0, u_i + \lambda\}$ [Wang and Carreira-Perpinán, 2013]. We use a max function here so that the eigenvalues of $\boldsymbol{D}$ are greater than or equal to zero ensuring we get a $\boldsymbol{X}$ such that $\boldsymbol{X} \succeq 0$. Now let us define how to find $d_i \in \mathbf{R}$, $u_i \in \mathbf{R}$, and $\lambda \in \mathbf{R}$. Using our $\boldsymbol{E}$ from the spectral decomposition we can form a vector $\boldsymbol{u}$ from the diagonal matrix $\boldsymbol{E}^T\boldsymbol{U}_{\boldsymbol{X}}\boldsymbol{E}$ which we can assume without loss of generality is sorted in descending order. Then we have

$$\lambda = \frac{1}{\rho}\left(K - \sum_{i=1}^{\rho} u_i\right) \text{ with } \rho = \max\left\{1 \leq j \leq N \ :\ u_j + \frac{1}{j}\left(K - \sum_{i=1}^{j} u_i\right) > 0\right\}. \tag{4.8}$$

With this we have updated $\boldsymbol{D}$ so that $\boldsymbol{X} \succeq 0$ and $tr(\boldsymbol{X}) = K$ and hence have projected $\boldsymbol{U}_{\boldsymbol{X}}$ back onto the set and found an optimal $\boldsymbol{X}^{k+1}$ for this update step. A more in

depth proof using Karush-Kuhn-Tucker, KKT, conditions of this will be provided in the appendix, Chapter 8, with reference to [Wang and Carreira-Perpinán, 2013]. This spectral decomposition back onto the set is the most expensive update step needing $\mathcal{O}(N^3)$ flops per iterations, and where the bottleneck of Alternating Direction Method of Multipliers occurs in the K disjoint clique problem.

The next update step to discuss is the update of our variable $\boldsymbol{Y}$. This step goes through a similar process of reducing the augmented Lagrangian, (4.5), and then projecting back onto the set of $\boldsymbol{Y}$ where $\boldsymbol{Y}\boldsymbol{e} \leq \boldsymbol{e}$, and $\boldsymbol{Y} = \boldsymbol{Y}^T$. To take a more in depth look at it, consider the augmented Lagrangian reduced to include only terms with $\boldsymbol{Y}$:

$$\begin{aligned}\boldsymbol{Y} := \underset{\boldsymbol{Y}\boldsymbol{e}\leq\boldsymbol{e}\,,\boldsymbol{Y}=\boldsymbol{Y}^T}{\arg\min} & -tr(\boldsymbol{e}\boldsymbol{e}^T\boldsymbol{Y}) - tr(\boldsymbol{\Lambda}_{\boldsymbol{XY}}\boldsymbol{Y}) + tr(\boldsymbol{\Lambda}_{\boldsymbol{YZ}}\boldsymbol{Y}) \\ & + \frac{\beta}{2}tr(\boldsymbol{Y}\boldsymbol{Y}^T - 2\boldsymbol{X}\boldsymbol{Y} + \boldsymbol{Y}\boldsymbol{Y}^T - 2\boldsymbol{Y}\boldsymbol{Z}). \end{aligned} \tag{4.9}$$

We will once again begin by factoring out β and separating our terms quadratic in $\boldsymbol{Y}$ and linear in $\boldsymbol{Y}$ to simplify our problem to:

$$\boldsymbol{Y} := \underset{\boldsymbol{Y}\boldsymbol{e}\leq\boldsymbol{e}\,,\boldsymbol{Y}=\boldsymbol{Y}^T}{\arg\min} \beta tr(\boldsymbol{Y}\boldsymbol{Y}^T - \boldsymbol{Y}^T\boldsymbol{U}_{\boldsymbol{Y}}),$$

with $\boldsymbol{U}_{\boldsymbol{Y}} = \frac{1}{2}(\frac{1}{\beta}\boldsymbol{e}\boldsymbol{e}^T + \frac{1}{\beta}\boldsymbol{\lambda}_{\boldsymbol{XY}} - \frac{1}{\beta}\boldsymbol{\lambda}_{\boldsymbol{YZ}} + \boldsymbol{X} + \boldsymbol{Z})$. Giving us a similar structured subproblem to the $\boldsymbol{X}$ update step to solve (4.9):

$$\boldsymbol{Y} := \underset{\boldsymbol{Y}\boldsymbol{e}\leq\boldsymbol{e}\,,\boldsymbol{Y}=\boldsymbol{Y}^T}{\arg\min} ||\boldsymbol{Y} - \boldsymbol{U}_{\boldsymbol{Y}}||_F^2, \tag{4.10}$$

which allows us to see that the optimal $\boldsymbol{Y}$ for this update is the $\boldsymbol{Y}$ that is the projection of $\boldsymbol{U}_{\boldsymbol{Y}}$ back onto the set of $\boldsymbol{Y}$ where $\boldsymbol{Y}\boldsymbol{e} \leq \boldsymbol{e}$, and $\boldsymbol{Y} = \boldsymbol{Y}^T$. This update seems to be achieved by just scaling the row sums of $\boldsymbol{U}_{\boldsymbol{Y}}$ so that they are less than or

equal to one. However this leaves us with a $\boldsymbol{Y}$ that is potentially not symmetric, and we need a symmetric $\boldsymbol{Y}$ to ensure that we get real eigenvalues from a symmetric $\boldsymbol{U_X}$ when we do the spectral decomposition in the $k+1$ iteration. To see how to solve (4.10) we have to look at its KKT conditions:

$$\begin{aligned} &\boldsymbol{Y} - \boldsymbol{U_Y} + \boldsymbol{\mu}\boldsymbol{e}^T + \boldsymbol{e}\boldsymbol{\mu}^T = 0 \\ &\boldsymbol{\mu}^T(\boldsymbol{Y}\boldsymbol{e} - \boldsymbol{e}) = 0 \text{ and } \boldsymbol{\mu}^T(\boldsymbol{Y}^T\boldsymbol{e} - \boldsymbol{e}) = 0 \\ &\boldsymbol{Y}\boldsymbol{e} \leq \boldsymbol{e} \text{ and } \boldsymbol{Y} \in \Sigma^N, \end{aligned} \tag{4.11}$$

where $\boldsymbol{\mu} \in \mathbf{R}^N$ is the dual variable as a vector and $\boldsymbol{e} \in \mathbf{R}^N$ is the vector of all ones. Which gives us a formula to update $\boldsymbol{Y} = \boldsymbol{U_Y} - \boldsymbol{\mu}\boldsymbol{e}^T - \boldsymbol{e}\boldsymbol{\mu}^T$, leading us to find our dual variable vector $\boldsymbol{\mu}$ so that we can finish our $\boldsymbol{Y}$ update. It is clear that when the i-th row sum of $\boldsymbol{U_Y} \leq 1$ then $\mu_i = 0$. We will now create a set S that contains the indices of the row such that the row sum of $\boldsymbol{U_Y}$ is greater than one. So $S = \{i \ : \ \sum_{j=1}^N Y_{ij} > 1\}$ and $\mu_i \neq 0$ in these cases. So we use our complementary slackness conditions from (4.11) to get the system based on the indices in set S:

$$\mu_S^T \left(\left([\boldsymbol{U_Y}]_S - \boldsymbol{\mu}_S \boldsymbol{e}_S^T - \boldsymbol{e}_S \boldsymbol{\mu}_S^T \right) \boldsymbol{e}_S - \boldsymbol{e}_S \right) = 0,$$

where $\boldsymbol{e}_S$ is the ones vector with size equal to cardinality of S, $\boldsymbol{\mu}_S$ is the vector of size equal to cardinality of S including the indices in S, and $[\boldsymbol{U_Y}]_S$ are the rows of $\boldsymbol{U_Y}$ from indices in S. From here we can solve for $\boldsymbol{\mu}_S = (N\boldsymbol{I}_S + \boldsymbol{E}_S)^{-1}[\boldsymbol{U_Y}]_S\boldsymbol{e}_S - \boldsymbol{e}_S)$ where $\boldsymbol{I}_S$ is the identity, and $\boldsymbol{E}_S$ is the matrix of all ones both with dimensions cardinality of S by cardinality of S. This will then allow us to finish our $\boldsymbol{Y}$ update by matching $\boldsymbol{\mu}_S$ with μ_i such that i in not in the set S and recalling that $\boldsymbol{Y} = \boldsymbol{U_Y} - \boldsymbol{\mu}\boldsymbol{e}^T - \boldsymbol{e}\boldsymbol{\mu}^T$. Though the update step solving (4.10) was more complicated than just scaling the rows of $\boldsymbol{U_Y}$ it requires $\mathcal{O}(N^3/3)$ which is less than the update step of our $\boldsymbol{X}$ variable.

We now turn our attention to the update step of our last primal variable $\boldsymbol{Z}$.

This is the easiest update step in both logic and code. It uses an almost identical process of reducing the augmented Lagrangian, and then we project back onto the set of $\boldsymbol{Z}$ where $Z_{ij} = 0,\ Z_{ji} = 0\ \ \forall\ (i,j) \notin E$ s.t. $i \neq j$. As before, consider the augmented Lagrangian reduced to include only terms with $\boldsymbol{Z}$:

$$\boldsymbol{Z} := \underset{[\boldsymbol{Z}]_{ij}=0,\ \forall\ (i,j)\notin E \text{ s.t. } i\neq j}{\arg\min} -tr(\boldsymbol{\Lambda_{XZ}Z}) - tr(\boldsymbol{\Lambda_{YZ}Z}) + \frac{\beta}{2}tr(\boldsymbol{ZZ}^T - 2\boldsymbol{XZ} + \boldsymbol{ZZ}^T - 2\boldsymbol{YZ}).$$

We will once again begin by factoring out β and separating our terms quadratic in $\boldsymbol{Z}$ and linear in $\boldsymbol{Z}$ to simplify our problem to:

$$\boldsymbol{Z} := \underset{[\boldsymbol{Z}]_{ij}=0,\ \forall\ (i,j)\notin E \text{ s.t. } i\neq j}{\arg\min} \beta tr(\boldsymbol{ZZ}^T - \boldsymbol{Z}^T\boldsymbol{U_Z}),$$

where $\boldsymbol{U_Z} = \frac{1}{2}(\frac{1}{\beta}\boldsymbol{\Lambda_{XZ}} + \frac{1}{\beta}\boldsymbol{\Lambda_{YZ}} + \boldsymbol{X} + \boldsymbol{Y})$; which is comparable in structure to the subproblems from the $\boldsymbol{X}$ and $\boldsymbol{Y}$ update steps (4.7) and (4.10):

$$\boldsymbol{Z} := \underset{[\boldsymbol{Z}]_{ij}=0,\ \forall\ (i,j)\notin E \text{ s.t. } i\neq j}{\arg\min} ||\boldsymbol{Z} - \boldsymbol{U_Z}||_F^2.$$

From here one can easily see that the optimal $\boldsymbol{Z}$ for this update is the $\boldsymbol{Z}$ that is the projection of $\boldsymbol{U_Z}$ back onto the set of $\boldsymbol{Z}$ where $Z_{ij} = 0,\ \ \forall\ (i,j) \notin E$ s.t. $i \neq j$. This update is easily achieved by setting

$$Z_{ij} = \begin{cases} 0 & \text{if } \ i,j \notin E \ \text{ and } \ i \neq j \\ [U_Z]_{ij} & \text{if } \ i,j \in E. \end{cases}$$

One thing of note is the fact that $\boldsymbol{Z}^{k+1}$ will be symmetric since $\boldsymbol{U_Z}$ is symmetric. We know this since our dual variables are guaranteed to be symmetric by update, and $\boldsymbol{X}^k$ and $\boldsymbol{Y}^k$ are symmetric by projection or decomposition.

The last part of our algorithm that we need to detail is the update step of our

dual variables $\boldsymbol{\Lambda_{XY}}$, $\boldsymbol{\Lambda_{XZ}}$ and $\boldsymbol{\Lambda_{YZ}}$ based on the theory of Alternating Direction Method of Multipliers. This update step is the standard dual variable update step in dual ascent with step size β so that our step is dual feasible [Boyd et al., 2011, Section 2]. One part of convergence to consider is when is our solution primal feasible. This can be looked at in many ways, but one of the most simple ways is notice that as dual variables stop changing then our problem is becoming primal feasible. We used this as part of our convergence requirements.

After using ADMM to solve the K disjoint clique problem, one can see many advantages or disadvantages to this method which we will discuss in this section. First let use begin with the disadvantages including the positive semidefinite constraint on $\boldsymbol{X}$ in our split SDP. This constraint and the projection required to solve this update step are the bottleneck of using ADMM to solve the K disjoint clique problem. The flop count for this update step is $\mathcal{O}(N^3)$. However there are many advantages, including the use of alternating direction for minimization to get closed form solutions for each update step. This is possible because of the splitting of $\boldsymbol{X}$ into $\boldsymbol{X}$, $\boldsymbol{Y}$, and $\boldsymbol{Z}$ which also helps make the updates more manageable and less complicated than if we did not split. Finally one other advantage is the guarantee of convergence of our solution based on using dual ascent and the method of multipliers combined with the knowledge of a closed, proper, and convex objective function and a saddle point for our unagumented Lagrangain. This is achieved with flop counts of $\mathcal{O}(N^3)$ for each iteration [Boyd et al., 2011].

Convergence for the two-block ADMM is shown in [Boyd et al., 2011] and is well know. However the convergence of the three-block ADMM is unclear. There have even been counter examples to convergence if no other assumptions are present in [Lin et al., 2015], and [Chen et al., 2016]. Our numerical experiments in Chapter 7 show the convergence of our three-block ADMM method in practice. We refer the readers to [Lin et al., 2015] for a proof of when the three-block ADMM method converges based on

splittings of the objective function and a bound on the penalty parameter.

CHAPTER 5

LOW RANK FACTORIZATION WITH METHOD OF MULTIPLIERS

In this section, we will apply a low rank factorization to the K Disjoint Clique problem, and the use method of multipliers to solve as discussed in [Burer and Monteiro, 2003]. We work to specialize this factorization to our formulation of K disjoint clique problem.

We will apply the method of multipliers, discussed in Chapter 4, specifically (4.3), to the KDC problem following a low-rank factorization of the decision variable $\boldsymbol{X}$. Recall our relaxed SDP (3.6) the K Disjoint Clique problem which we have worked with in Chapter 3 and Chapter 4:

$$\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} \\
\text{s.t.} \quad & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{e}, \\
& X_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0,
\end{aligned}$$

where $\boldsymbol{e} \in \mathbf{R}^N$ is the vector of ones, our graph G contains N vertices, E is the set of edges in our graph G, K is the desired number of cliques, which is given, and $\boldsymbol{X} \in \mathbf{R}^{N\times N}$ is our variable. To see specifically how we can apply a low rank factorization from [Burer and Monteiro, 2003] and follow by using the method of multipliers, we will rewrite our SDP with variable $\boldsymbol{X}$ in terms of the trace inner

product function to get:

$$
\begin{aligned}
\min \quad & -tr(\boldsymbol{C}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\boldsymbol{A_i}\boldsymbol{X}) \leq 1 \;\; \forall\, i = 1, ...N, \\
& tr(\boldsymbol{A_{ij}}\boldsymbol{X}) = 0, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0,
\end{aligned}
\tag{5.1}
$$

where $\boldsymbol{C}$ and the $\boldsymbol{A}$'s are N by N data matrices, and the right hand side of our constraints can be grouped into vector $\boldsymbol{b}$ for ease of notation. We will discuss in more detail how to find these $\boldsymbol{A}'s$ in the appendix.

Our motivation for using a low rank factorization, and the method of multipliers stems from SDP's (5.1), (3.6), and the difficult positive semidefinite constraint $\boldsymbol{X} \succeq 0$. This constraint is the most time consuming, in terms of flops, in Algorithm 1. As an attempt to avoid or eliminate this constraint we recall the fact that if $X \succeq 0$ then there exist a $\boldsymbol{V}$ which is N by N such that $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$. If we recall that our $tr(\boldsymbol{X}) = K$ constraint is a relaxation of the constraint $rank(\boldsymbol{X}) = K$, we can actually require $\boldsymbol{V}$ to be N by K since K is given. For more detail into why this is true we refer the reader to [Kulis et al., 2007] which details the clustering problem as an SDP. With this low rank factorization of $\boldsymbol{X}$ into $\boldsymbol{V}\boldsymbol{V}^T$ we get the following SDP:

$$
\begin{aligned}
\min \quad & -tr(\boldsymbol{C}\boldsymbol{V}\boldsymbol{V}^T) \\
\text{s.t.} \quad & tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) \leq 1 \;\; \forall\, i = 1, ...N, \\
& tr(\boldsymbol{A_{ij}}\boldsymbol{V}\boldsymbol{V}^T) = 0, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T) = K,
\end{aligned}
\tag{5.2}
$$

which reduces our variable size from N by N to N by K. This can be a very significant reduction if K is much smaller than N, which it will be the most interesting cases

based on recovery theorems and examples from [Ames and Vavasis, 2014] discussed in Chapter 3. Eliminating the difficult constraint $\boldsymbol{X} \succeq 0$ is beneficial, however it does not come without it challenges. First, we have taken constraints that were linear in $\boldsymbol{X}$ to constraints that are quadratic in $\boldsymbol{V}$. We have also introduced local solutions that we must avoid since our problem is no longer convex.

5.1 Recovery of our Factored SDP

In similar thought to recovery of our K Disjoint Clique problem in Chapter 3, we add a theorem for recovery of our factored SDP (5.2) resembling the same logic as theorems from [Ames and Vavasis, 2014].

Theorem 3. *Suppose that $G = [V, E]$ is sampled from the planted K-disjoint-clique subgraph model with adversarial noise satisfying the hypothesis of Theorem 1 or random noise satisfying the hypothesis of Theorem 2. Let $C_1, C_2, \ldots, C_K$ denote the cliques comprising the planted K-disjoint-clique subgraph and $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_K$ be their characteristic vectors. Let*

$$\boldsymbol{V}^* = \left[\frac{\boldsymbol{x}_1}{\|\boldsymbol{x}_1\|} | \cdots | \frac{\boldsymbol{x}_K}{\|\boldsymbol{x}_K\|} \right] \tag{5.3}$$

and let $\boldsymbol{X}^* = \boldsymbol{V}^*(\boldsymbol{V}^*)^T$. Then $\boldsymbol{V}^*$ is an optimal solution of (5.2)*, $\boldsymbol{X}^*$ is the unique optimal solution of* (3.6)*, and G^* is the unique maximum node K-disjoint-clique subgraph of G (with high probability in the case of random noise).*

Proof. That X^* and G^* are the unique optimal solutions of (3.6) and (3.1), respectively, follows immediately from Theorems 1 and 2. Moreover, any K-disjoint-clique subgraph (and its collection of characteristic vectors) defines a feasible solution $\boldsymbol{V}$ for (5.2) by (5.3), which in turn defines a feasible solution for (3.6) by $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$. This implies that (5.2) is a (nonconvex) relaxation of (3.1) and (3.6) is a relaxation of (5.2). Since (3.6) is a tight relaxation of (3.1), we see that all relaxations are tight here, and $\boldsymbol{V}^*$ corresponding to X^* and G^* must be optimal for (5.2). □

It should be noted that $\boldsymbol{V}^*$ is not unique as we could permute the columns of

Algorithm 2: Low Rank Factorization with Method of Multipliers Algorithm

Data: Input $\boldsymbol{V}, \boldsymbol{y}, \sigma, \boldsymbol{C}, \boldsymbol{A}'s, \boldsymbol{b}$
for *Until Converged*
  $\boldsymbol{V}^{k+1} = \arg\min \mathcal{L}_\sigma(\boldsymbol{V}^k, \boldsymbol{y})$
  Compute $\nu = \sum_{i=1}^m (tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i)^2$
  If $\nu < \zeta\nu_k$, then set,
  $y_i^{k+1} = \max(y_i^k - \sigma_k(tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i), 0) \ \ \forall \ \ i = 1, ..., m,$
  $\sigma_{k+1} = \sigma_k,$
  $\nu_{k+1} = \nu;$
  otherwise, set,
  $y_i^{k+1} = y_i^k \ \ \forall \ \ i,$
  $\sigma_{k+1} = \gamma\sigma_k,$
  $\nu_{k+1} = \nu_k;$
end
Data: Output $\boldsymbol{V}$

$\boldsymbol{V}^*$ and still get the corresponding $\boldsymbol{X}^*$.

We will now take time to look at the details of the method of multipliers to our low rank factorization (5.2). First let us consider the augmented Lagrangian of SDP (5.2):

$$\mathcal{L}_\sigma(\boldsymbol{V}, \boldsymbol{y}) = -tr(\boldsymbol{C}\boldsymbol{V}\boldsymbol{V}^T) + \sum_{i=1}^m y_i(tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i) + \frac{\sigma}{2}(\sum_{i=1}^m (tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i)^2) \quad (5.4)$$

with dual variable vector $\boldsymbol{y} \in \mathbf{R}^m$ and penalty parameter σ. We will use the method of multipliers to minimize augmented Lagrangian (5.4), by updating our penalty parameters and dual variables to achieve convergence of a desired tolerance. The method of multipliers uses this penalty parameter to achieve an answer that is both feasible and optimal. This gives us the general framework of our low rank factorization algorithm which was inspired by [Burer and Monteiro, 2003] with exception of our update for dual variable $\boldsymbol{y}$ which we detail later.

Algorithm 2 has parameters $\gamma > 1$ and $\zeta < 1$ given. The reason we must change our update of the dual variable $\boldsymbol{y}$ is in [Burer and Monteiro, 2003] their work is for SDP's with equality constraints only. However, that is not true for our problem as we

have inequality and equality constraints, and thus need to use slack variables with the inequality constraints changing our dual variable update. We must enforce the requirement that $y_i \geq 0$ for $i = 1, ..., m$ for Lagrange multipliers with inequality constraints [Kulis et al., 2007] hence we have used the max function during the update step of $\boldsymbol{y}$, our dual variable. We use the dual variable update explicitly from [Kulis et al., 2007]. If one was to describe the algorithm above in words, one might say that we should calculate an iterate $\boldsymbol{V}$, and then check to see how infeasible $\boldsymbol{V}$ is at the current iteration by finding ν. If that infeasibility is smaller than $\zeta\nu_k$, the best infeasibility so far multiplied by ζ, then we update the dual variable $\boldsymbol{y}$. If it is not smaller then we continually increase the penalty parameter σ by a factor of γ to reduce infeasibility of our new $\boldsymbol{V}$ obtained by minimizing the augmented Lagrangian. This gives us a sequence of $\boldsymbol{V}^k$ that converges to the optimal $\boldsymbol{V}$ under suitable assumptions [Burer and Monteiro, 2003].

Since we have an understanding of the algorithm, and how it generally works lets take a look at each step in detail. Starting with the first step of minimizing the augmented Lagrangian $\mathcal{L}_\sigma(\boldsymbol{V}, \boldsymbol{y})$, which we will refer to as the subproblem. This is the step with the most freedom, and impact on the convergence rate or efficiency of the algorithm. There are many methods that could be used to solve this unconstrained minimization such as first order methods, second order methods, or stochastic methods. However, we have chosen to follow in Burer Monteiro's lead in [Burer and Monteiro, 2003], and use a gradient based method only requiring calculations of the function and gradient which we calculated with [Petersen et al., 2008]. We have used the `minFunc` package from [Schmidt, 2005] to optimize this step. One can choose from many methods to solve the problem when using `minFunc` such as first order methods, or second order methods. However, to be efficient in memory usage, and quickness of convergence we have use the LBFGS method to solve the subproblem [Nocedal and Wright, 2006, Chapter 6].

The rest of the algorithm can be viewed as an evaluation, and following if else statement so we will discuss it altogether. First, the evaluation of ν is calculating the penalty term of our augmented Lagrangian without our penalty parameter σ. This is done to see how close our current $\boldsymbol{V}^{k+1}$ is to feasibility and compare to $\zeta\nu_k$, a decreased factor of our previous best infeasibility. If our evaluation of ν is less than $\zeta\nu_k$ then we have acceptable decrease in our infeasibility, and we will update our dual variable $\boldsymbol{y}_{k+1}$ as described in the above algorithm. While keeping our penalty parameter, σ, and saving our new best infeasibility, ν_{k+1}. However if ν is greater than or equal to $\zeta\nu_k$ we do not have acceptable decrease in our infeasibility. So we keep the same dual variable $\boldsymbol{y}_k$, and best infeasibility ν_k while increasing the penalty parameter by a factor of γ. We do this to increase the penalization of an infeasible solution, and drive down infeasibility until we are back in stage one of the if else statement. As we continue this we drive our infeasibility down, and create a feasible solution that is optimal for the original SDP (5.2) which we created the augmented Lagrangian (5.4) from earlier.

With the detail of our algorithm above, we now note a few advantages and disadvantages. First for the disadvantages, by using the augmented Lagrangian over our variable $\boldsymbol{V}$, and not splitting the variable like in the previous Chapter 4, we have to solve the subproblem, and do not have explicit updates which can be time consuming. The method used to solve the subproblem is very important to the efficiency of the algorithm, so improvements in this would be of great interest such as possible stochastic gradient methods. Also as mentioned above introducing the variable $\boldsymbol{V}$ such that $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ does not come without its challenges including constraints being quadratic in $\boldsymbol{V}$ rather than linear in $\boldsymbol{X}$, and our problem is no longer convex, so we have introduced local solutions. However, there are many advantages to solving our K Disjoint Clique problem when written as a low rank factorization (5.2). First we have greatly reduced the size of our variable from $\boldsymbol{X} \in \mathbf{R}^{N\times N}$ to $\boldsymbol{V} \in \mathbf{R}^{N\times K}$ which is of great value when $K << N$. Also, we have eliminated the positive semidefinite

constraint by introducing $\boldsymbol{V}$ [Burer and Monteiro, 2003]. This constraint was the bottleneck of our Alternating Direction Method of Multipliers algorithm above, and limited us to $\mathcal{O}(N^3)$ per iteration. However, with Algorithm 2 we can reduce our flop counts to $\mathcal{O}(N^2K)$.

Finally, we should detail the two versions of our low rank factorization code we used in solving the K Disjoint Clique problem. They both follow the same outline of the above Algorithm 2, and use `minFunc` [Schmidt, 2005] with LBFGS to solve the subproblem. Solving this requires evaluations of the augmented Lagrangian and the gradient. We have the previous mentioned augmented Lagrangian (5.4):

$$\mathcal{L}_\sigma(\boldsymbol{V}, \boldsymbol{y}) = -tr(\boldsymbol{C}\boldsymbol{V}\boldsymbol{V}^T) + \sum_{i=1}^{m} y_i(tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i) + \frac{\sigma}{2}(\sum_{i=1}^{m}(tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i)^2).$$

Which leads us to the following gradient of the augmented Lagrangian (5.4):

$$\nabla_{\boldsymbol{V}}\mathcal{L}_\sigma(\boldsymbol{V}, \boldsymbol{y}) = -2\boldsymbol{C}\boldsymbol{V} + 2\sum_{i=1}^{m} y_i\boldsymbol{A_i}\boldsymbol{V} + 2\sigma\sum_{i=1}^{m}(tr(\boldsymbol{A_i}\boldsymbol{V}\boldsymbol{V}^T) - b_i)(\boldsymbol{A_i}\boldsymbol{V}). \tag{5.5}$$

As N and the size of variable $\boldsymbol{V}$, increase the matrix multiplications of $\boldsymbol{C}\boldsymbol{V}$ and $\boldsymbol{A_i}\boldsymbol{V}$, which are required in (5.5), become more time consuming during the subproblem. So, as an attempt to optimize the algorithm we have created one algorithm, called `BM2`, to complete the matrix multiplication and another, called `BMHC`, used analytic formulas for the matrix product of $\boldsymbol{A}\boldsymbol{V}$'s rather than performing the matrix multiplications to save flops while solving the subproblem. It is clear theoretically that `BMHC` will be faster than `BM2` as N grows. We will look at this in the numerical results and then discuss these analytic formulas in more detail in Chapter 8 including finding the analytic formulas for all $\boldsymbol{A}'s$ and matrix multiplications $\boldsymbol{A}\boldsymbol{V}$'s.

CHAPTER 6

BICONVEX RELAXATION

In this section, we will look at how the biconvex relaxation approach from Shah *et al.* in [Shah et al., 2016] can be applied to the K Disjoint Clique problem. Let us start by reviewing what it means for an optimization problem to be biconvex, and how we can relax our semidefinite program (3.6) for the K Disjoint Clique problem to be biconvex. First, an optimization problem of the form:

$$\min f(x, y) \text{ such that } (x, y) \in B$$

is said to be biconvex if the feasible set B is biconvex on $X \times Y$ and the objective function f is biconvex on B [Gorski et al., 2007]. We also consult [Gorski et al., 2007] to find definitions of biconvex sets and functions. The set $B \subseteq X \times Y$ is called a biconvex set on $X \times Y$ if $B_x := \{y \in Y : (x, y) \in B\}$ is convex for every x in X and $B_y := \{x \in X : (x, y) \in B\}$ is convex for every y in Y. A function $f : B \to \mathbf{R}$ on a biconvex set $B \subseteq X \times Y$ is called a biconvex function on B if $f_x(\bullet) := f(x, \bullet) : B_x \to \mathbf{R}$ is a convex function on B_x for every $x \in X$ and $f_y(\bullet) := f(\bullet, y) : B_y \to \mathbf{R}$ is a convex function on B_y for every $y \in Y$. Which can be thought of as the optimization problem is biconvex, when it is convex with respect to a group of variables while the remaining variables are held constant. Now we need to show how the K Disjoint Clique problem can be relaxed to a biconvex form. To refresh your memory or aid those who may start reading in this section, we recall our combinatoral optimization problem for the

K Disjoint Clique (3.1) which is discussed in detail in Chapter 3:

$$\begin{aligned}
\max_{S=\{\boldsymbol{x}_1,\ldots,\boldsymbol{x}_K\}} \quad & \sum_{i=1}^{K} \boldsymbol{x}_i^T \boldsymbol{e} \\
\text{s.t.} \quad & \boldsymbol{x}_i^T \boldsymbol{x}_j = 0, \quad \forall\ i\ ,\ j = 1, ..., K\ \ ,\ i \neq j \\
& [x_i x_i^T]_{uv} = 0, \quad \text{if } uv \notin E,\ u \neq v,\ \ \forall\, i = 1, ..., K \\
& \boldsymbol{x}_i \in \{0,1\}^V,\ \ \forall i = 1, ..., K
\end{aligned}$$

where $\boldsymbol{e}$ is the set of all ones in $\mathbf{R}^N$, E is the set of edges for our graph G, and $S = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_K\}$ is the collection of characteristic vectors of a set of disjoint cliques of G. In Chapter 5, we relaxed this to a semidefinite program with variable $\boldsymbol{X} \in \mathbf{R}^{N\times K}$ in terms for the trace or inner product function to find the relaxed SDP (3.6):

$$\begin{aligned}
\min \quad & -tr(\boldsymbol{C}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\boldsymbol{A_i}\boldsymbol{X}) \leq 1 \ \ \forall\, i = 1, ...N, \\
& tr(\boldsymbol{A_{ij}}\boldsymbol{X}) = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0.
\end{aligned}$$

Recall that our $\boldsymbol{A}$'s are data matrices that we find explicit forms for in the appendix, Chapter 8. For us to be able to relax problem above (3.6) into a biconvex form we need to require the data matrices to be positive semidefinite [Shah et al., 2016]. We will refer to the positive semidefinite data matrices as $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{A}}$ with $\hat{\boldsymbol{C}} \succeq 0$ and $\hat{\boldsymbol{A}} \succeq 0$. We do this so that each $\hat{\boldsymbol{A}}$ has the factorization $\hat{\boldsymbol{A}} = \boldsymbol{L}^T\boldsymbol{L}$. This is guaranteed if $\hat{\boldsymbol{A}}$ is positive semidefinite, because any matrix $\hat{\boldsymbol{A}}$ is positive semidefinite can be factored in $\hat{\boldsymbol{A}} = \boldsymbol{L}^T\boldsymbol{L}$ [Petersen et al., 2008, Chapter 5].

Lemma 6. *If matrix $\boldsymbol{X}$ is positive semidefinite, $\boldsymbol{X} \succeq 0$, then $\boldsymbol{X}$ can be factored into $\boldsymbol{X} = \boldsymbol{L}\boldsymbol{L}^T$ with $\boldsymbol{L}$ a lower triangular matrix.*

One thing to note is that this factorization is not unique unless $\boldsymbol{X} \succ 0$. We will discuss the factorization of $\hat{\boldsymbol{A}}$ into $\boldsymbol{L}^T\boldsymbol{L}$ in more detail in the appendix as well as detailing how to make $\boldsymbol{A}$'s into $\hat{\boldsymbol{A}}$'s, so we can have the desired factorization. However to have the general thought, we simply add a multiple of the identity to ensure that all eigenvalues are greater than or equal to zero. For example, add $\zeta\boldsymbol{I}$ where ζ is the shift needed to make all eigenvalues greater than or equal to zero and $\boldsymbol{I}$ is the identity matrix size N by N. We shift $\boldsymbol{A}$ to $\hat{\boldsymbol{A}}$ by adding $\zeta\boldsymbol{I}$ to $\boldsymbol{A}$, so that the smallest eigenvalue is greater than or equal to zero, and we now have constructed each $\hat{\boldsymbol{A}}$ which is positive semidefinite. This shift will give us the following equivalent SDP:

$$\begin{aligned}
\min \quad & -tr(\hat{\boldsymbol{C}}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\hat{\boldsymbol{A}}_{\boldsymbol{i}}\boldsymbol{X}) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\hat{\boldsymbol{A}}_{\boldsymbol{ij}}\boldsymbol{X}) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0.
\end{aligned} \tag{6.1}$$

One thing of note is that our final $\boldsymbol{A}$ in the constraint based on the rank of $\boldsymbol{X}$, $tr(\boldsymbol{A}\boldsymbol{X}) = K$, does not have to be shifted as this $\boldsymbol{A} = \boldsymbol{I}$ is already positive semidefinite, and clearly has a factored form, $\boldsymbol{A} = \boldsymbol{I}^T\boldsymbol{I}$ that we desire for our problem. Also this does change our data vector $\boldsymbol{b}$ when we shift from $\boldsymbol{A}$ to $\hat{\boldsymbol{A}}$, and this is explained in detail in Chapter 8. Now to relax the SDP (6.1) into a biconvex form, we need to use the fact that $\hat{\boldsymbol{A}} = \boldsymbol{L}^T\boldsymbol{L}$ for some $\boldsymbol{L} \in \mathbf{R}^{N\times N}$, whether it be the square root of $\hat{\boldsymbol{A}}$ or the Cholesky decomposition since this decomposition is not necessarily unique. Which allows use to rewrite each of our constraints as Frobenius matrix norms, $tr(\hat{\boldsymbol{A}}\boldsymbol{X}) = tr(\boldsymbol{V}^T\hat{\boldsymbol{A}}\boldsymbol{V}) = tr(\boldsymbol{V}^T\boldsymbol{L}^T\boldsymbol{L}\boldsymbol{V}) = ||\boldsymbol{L}\boldsymbol{V}||_F^2$. With $||\cdot||_F$ the Frobenius matrix norm, by considering the fact $tr(\boldsymbol{A}^T\boldsymbol{A}) = ||\boldsymbol{A}||_F^2$. We will show the above rewrite in

four steps as SDPs. First,

$$
\begin{aligned}
\min \quad & -tr(\hat{\boldsymbol{C}}\boldsymbol{V}\boldsymbol{V}^T) \\
\text{s.t.} \quad & tr(\hat{\boldsymbol{A}}_{\boldsymbol{i}}\boldsymbol{V}\boldsymbol{V}^T) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\hat{\boldsymbol{A}}_{\boldsymbol{ij}}\boldsymbol{V}\boldsymbol{V}^T) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T) = K. \\
& ,
\end{aligned}
\tag{6.2}
$$

which we get from the fact that $\boldsymbol{X} \succeq 0$ implying that $\boldsymbol{X}$ has low rank factorization $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ used in Chapter 5 from [Burer and Monteiro, 2003]. Then we use another factorization based on the positive semidefinite requirement on $\hat{\boldsymbol{A}}$'s $= \boldsymbol{L}^T\boldsymbol{L}$ to get:

$$
\begin{aligned}
\min \quad & -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) \\
\text{s.t.} \quad & tr(\boldsymbol{V}^T\boldsymbol{L}_{\boldsymbol{i}}^T\boldsymbol{L}_{\boldsymbol{i}}\boldsymbol{V}) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\boldsymbol{V}^T\boldsymbol{L}_{\boldsymbol{ij}}^T\boldsymbol{L}_{\boldsymbol{ij}}\boldsymbol{V}) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{V}^T\boldsymbol{L}^T\boldsymbol{L}\boldsymbol{V}) = K.
\end{aligned}
\tag{6.3}
$$

Next we rewrite it in the form of Frobenius matrix norms:

$$
\begin{aligned}
\min \quad & -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) \\
\text{s.t.} \quad & \boldsymbol{Q}_{\boldsymbol{i}} = \boldsymbol{L}_{\boldsymbol{i}}\boldsymbol{V} \;\; , \;\; ||\boldsymbol{Q}_{\boldsymbol{i}}||_F^2 \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& \boldsymbol{Q}_{\boldsymbol{ij}} = \boldsymbol{L}_{\boldsymbol{ij}}\boldsymbol{V} \;\; , \;\; ||\boldsymbol{Q}_{\boldsymbol{ij}}||_F^2 = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& \boldsymbol{Q} = \boldsymbol{L}\boldsymbol{V} \;\; , \;\; ||\boldsymbol{Q}||_F^2 = K,
\end{aligned}
\tag{6.4}
$$

where we have introduced a new variable $\boldsymbol{Q} \in \mathbf{R}^{N \times K}$ such that $\boldsymbol{Q} = \boldsymbol{L}\boldsymbol{V}$ in each constraint. Now the only thing that is not convex in our SDP (6.4) is the equality constraints which represent the missing edge constraint, and the rank constraint from the original optimization problem (3.5) which we relaxed to a trace constraint in (3.6).

To deal with this, we introduce one final relaxation to make our problem biconvex in $\boldsymbol{V}$ and $\boldsymbol{Q}$:

$$\begin{aligned}
\min_{\boldsymbol{V},\boldsymbol{Q}} \quad & -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) + \frac{\alpha}{2}\sum_{\boldsymbol{ij}\notin E}||\boldsymbol{Q_{ij}} - \boldsymbol{L_{ij}}\boldsymbol{V}||_F^2 + \frac{\alpha}{2}||\boldsymbol{Q} - \boldsymbol{L}\boldsymbol{V}||_F^2 + \\
& \frac{\alpha}{2}\sum_{\boldsymbol{i}=1}^{N}||\boldsymbol{Q_i} - \boldsymbol{L_i}\boldsymbol{V}||_F^2 - \frac{\beta}{2}\sum_{\boldsymbol{ij}\in E}||\boldsymbol{Q_{ij}}||_F^2 - \frac{\beta}{2}||\boldsymbol{Q}||_F^2 \\
\text{s.t.} \quad & ||\boldsymbol{Q_i}||_F^2 \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& ||\boldsymbol{Q_{ij}}||_F^2 \leq K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& ||\boldsymbol{Q}||_F^2 \leq K,
\end{aligned} \tag{6.5}$$

where $\alpha > \beta > 0$ are relaxation parameters discussed in detail in [Shah et al., 2016]. This last relaxation requires some verification for motivation, and why it works. The major details can be found in [Shah et al., 2016], however the thought comes from the need to eliminate equality constraints to achieve our desired biconvex quality. We have relaxed them all to inequality constraints, but added negative quadratic penalties to the objective to force the equality of these constraints when the objective is minimized. The final biconvex problem has a similar spirit as the augmented Lagrangian with penalty terms.

6.1 Recovery of our Factored Biconvex SDP

In similar thought to recovery of our K Disjoint Clique problem in Chapter 3, we add a theorem for recovery of our factored biconvex SDP (6.5) resembling the same logic as theorems from [Ames and Vavasis, 2014].

Theorem 4. *Suppose that $G = [V, E]$ is sampled from the planted K-disjoint-clique subgraph model with adversarial noise satisfying the hypothesis of Theorem 1 or random noise satisfying the hypothesis of Theorem 2. Let $C_1, C_2, \ldots, C_K$ denote the cliques comprising the planted K-disjoint-clique subgraph and $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_K$ be their*

characteristic vectors. Let

$$\boldsymbol{V}^* = \left[\frac{\boldsymbol{x}_1}{\|\boldsymbol{x}_1\|}|\cdots|\frac{\boldsymbol{x}_K}{\|\boldsymbol{x}_K\|}\right]$$

and let $\boldsymbol{X}^* = \boldsymbol{V}^*(\boldsymbol{V}^*)^T$*. Then* $\boldsymbol{V}^*$ *is an optimal solution of* (6.5)*,* $\boldsymbol{X}^*$ *is the unique optimal solution of* (3.6)*, and* G^* *is the unique maximum node* K*-disjoint-clique subgraph of* G *(with high probability in the case of random noise).*

Proof. That $\boldsymbol{X}^*$ and G^* are the unique optimal solutions of (3.6) and (3.1), respectively, follows immediately from Theorems 1 and 2. Moreover, any K-disjoint-clique subgraph (and its collection of characteristic vectors) defines a feasible solution $\boldsymbol{V}$ for (6.5) by (5.3), which in turn defines a feasible solution for (3.6) by $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$. This implies that (6.5) is a (nonconvex) relaxation of (3.1) and (3.6) is a relaxation of (6.5). Since (3.6) is a tight relaxation of (3.1), we see that all relaxations are tight here, and $\boldsymbol{V}^*$ corresponding to $\boldsymbol{X}^*$ and G^* must be optimal for (6.5). □

Now that we have verified the K Disjoint Clique problem can be rebuilt in a biconvex form and recovery of optimal solution is guaranteed we now need to discuss how we will solve it. Which brings us to one of the most beneficial aspects of writing our problem in biconvex form, the SDP can be approximately solved using alternating minimization since problem (6.5) can be globally minimized with respect to $\boldsymbol{Q}$ or $\boldsymbol{V}$ while the other is held constant. For a detailed discussing on alternating minimization and convergence we refer the reader to [Douglas and Gunn, 1964, Duchi and Singer, 2009]. However, alternating minimization is exactly what it sounds like, minimize with respect to variable one while the variable two is held constant, and then minimize with respect to variable two while variable one is held constant in the case of two sets of variables such as our above problem. This approach yields the general framework for Algorithm 3 using our biconvex relaxation to solve the K Disjoint Clique problem in biconvex form. With Algorithm 3 in mind we need to discuss each of the steps, let us start with the input. From our K Disjoint Clique problem one can find all of the

Algorithm 3: Biconvex Relaxation Alternating Direction

Data: Input $\hat{\boldsymbol{C}}, \boldsymbol{L}'s, \alpha, \beta, \hat{\boldsymbol{A}}, \boldsymbol{b}, \zeta$
for *Until Converged*
  Find $\boldsymbol{Q}'s$ by minimizing with respect to each $\boldsymbol{Q}$,
  Find $\boldsymbol{V}$, by minimizing with respect to $\boldsymbol{V}$
end
Data: Output $\boldsymbol{V}$

inputs, $\hat{\boldsymbol{C}}$, $\boldsymbol{L}$'s, α, β, $\hat{\boldsymbol{A}}$, $\boldsymbol{b}$, and ζ, but finding them efficiently will help with the speed of our algorithm. We provide analytic formulas for each $\boldsymbol{L}$ and $\hat{\boldsymbol{A}}$ in the appendix, as well as efficient formulas for their action as linear operators. We have also set relaxation parameters α and β to the suggestions of paper [Shah et al., 2016]. It should also be noted that $\boldsymbol{b}$ is a vector of size equal to the number of constraints, and is the right hand side of the constraints in vector form as mentioned in previous formulations of the general problem.

Now we move onto the step where we find each $\boldsymbol{Q}$. First, let us mention that the total number of $\boldsymbol{Q}$'s will be N plus the number of zero entries above the diagonal in graph G plus one which could be a significant number, $\mathcal{O}(\tau N^2)$ with τ a constant, as N or the probability of zero entries increases with initial parameters. Since each $\boldsymbol{Q_i}$ is independent of the other we can minimize our objective, which is quadratic in $\boldsymbol{Q_i}$, for each $\boldsymbol{Q}_i$ individually. This can be done by minimizing, and then reprojecting back onto the unit Frobenius-norm ball of radius $\sqrt{b_i}$ for each i. Which leads us the the following update from [Shah et al., 2016]:

$$\boldsymbol{Q_i} = \frac{\boldsymbol{L_i V}}{||\boldsymbol{L_i V}||_F} \min\{\sqrt{b_i}, \frac{\alpha}{\alpha - \beta_i}||\boldsymbol{L_i V}||_F\},$$

where $\beta_i = 0$ if the i-th constraint is an inequality or $\beta_i = \beta$ if the i-th constraint is an equality. This causes $\boldsymbol{Q_i}$ to expand if i is equality thus trying to satisfy equality, which we desire, even though we only require inequality in our final biconvex formulation (6.5). This could possibly lead us to converge to the all-zeros matrix, which is a local

solution of the biconvex relaxation K Disjoint Clique problem, i.e, $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T \to \boldsymbol{0}$ to avoid this we slightly changed the update for the $\boldsymbol{Q}$ corresponding to constraint $tr(\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T) = K$. We used the update

$$\boldsymbol{Q} = \frac{\boldsymbol{L}\boldsymbol{V}}{||\boldsymbol{L}\boldsymbol{V}||_F}(\sqrt{K}),$$

so that in our $\boldsymbol{V}$ update, discussed next, $\boldsymbol{Q}$ would be bounded away from $\boldsymbol{0}$ therefore avoiding the local solution $\boldsymbol{0}$.

After we update each $\boldsymbol{Q_i}$, we would then need to update $\boldsymbol{V}$. At this point we have some freedom on how to calculate this $\boldsymbol{V}$ as discussed in [Shah et al., 2016], based on the size of your problem. When holding each $\boldsymbol{Q}$ constant you get the following least-squares problem to minimize to find $\boldsymbol{V}$:

$$\boldsymbol{V} := \arg\min_{\boldsymbol{V}} -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) + \frac{\alpha}{2}\sum_{\text{all constraints}} ||\boldsymbol{Q_i} - \boldsymbol{L_i}\boldsymbol{V}||_F^2.$$

This is a stationary point of a convex quadratic, and has formulation from [Shah et al., 2016] using the matrix inverse to get:

$$\boldsymbol{V} := \left(\hat{\boldsymbol{C}} + \alpha \sum_{\text{all constraints}} \boldsymbol{L_i}^T\boldsymbol{L_i}\right)^{-1}\left(\sum_{\text{all constraints}} \boldsymbol{L_i}^T\boldsymbol{Q_i}\right),$$

where the matrix inverse part can be computed once at the beginning, since it does not change for each iteration. This is potentially time consuming if our matrices are large so to avoid this you can use the pseudo-inverse, use gradient descent, or even call `minFunc` we used in Algorithm 2. To be the most time efficient, and least memory extensive we decided to use gradient descent, and call `minFunc` [Schmidt, 2005] for two versions of Algorithm 3. We were able to solve for the optimized step size in the gradient descent case. We did all of this with the added constraint, $tr(\boldsymbol{V}\boldsymbol{V}^T) = K$ to

our least-squares problem to ensure that $\boldsymbol{V} \nrightarrow 0$. This gives us the following problem:

$$\begin{aligned} \boldsymbol{V} := \arg\min \quad & -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) + \frac{\alpha}{2}\sum_{\text{all constraints}} ||\boldsymbol{Q_i} - \boldsymbol{L_i}\boldsymbol{V}||_F^2 \\ \text{s.t.} \quad & tr(\boldsymbol{V}\boldsymbol{V}^T) = K, \end{aligned} \tag{6.6}$$

which we solved by taking a limited number of steps of gradient descent. If we consider f equal to the Lagrangian, and corresponding gradient ∇f we can find step length, ϕ, that optimizing the following gradient descent process:

$$\boldsymbol{V}^{k+1} = \boldsymbol{V}^k - \phi(\nabla f).$$

If we work through the KKT conditions, project back so $tr(\boldsymbol{V}\boldsymbol{V}^T) = K$, and let $\boldsymbol{D} = -\nabla f$, for ease of notation, then we have the following:

$$\phi = \frac{-2tr(\hat{\boldsymbol{C}}\boldsymbol{V}\boldsymbol{D}^T) + \sum_{\text{all}} \alpha(tr[(\boldsymbol{Q_i} - \boldsymbol{L_i}\boldsymbol{V})(\boldsymbol{D}^T\boldsymbol{L_i}^T)]}{2tr(\hat{\boldsymbol{C}}\boldsymbol{D}\boldsymbol{D}^T) + \sum_{\text{all}} \alpha tr(\boldsymbol{L_i}\boldsymbol{D}\boldsymbol{D}^T\boldsymbol{L_i}^T)}.$$

This allows us to find our $\boldsymbol{V}$ by the above gradient descent formula. The calculations of the gradient can be time consuming, however we have chosen to hard code parts of the Lagrangian and gradient when possible to save flops. Our second version of Algorithm 3 using `minFunc` follows the same outline, but we have used `minFunc` to solve the $\boldsymbol{V}$ update step with the augmented Lagrangian of (6.6) as our function. We used the LBFGS option in `minFunc` to decrease memory use, and improve run time.

With great insight of Algorithm 3, we can now discuss a few advantages and disadvantages. First, the most obvious and concerning disadvantage is the possibility of local solutions. We have altered the algorithm and updates from [Shah et al., 2016] to attempt to avoid these local solutions, and achieve convergence to the global expected solution of our K Disjoint Clique problem (3.6). Another disadvantage could be the size, and number of $\boldsymbol{Q}$'s and $\boldsymbol{L}$ that potentially have to be stored in Algorithm (3).

The process in which we create the $\hat{\boldsymbol{A}}$'s matrices needed for $\boldsymbol{L}$'s is not difficult however due to the size of the shift ζ, generally N in our case, needed it made finding initial parameters α, and β that promote optimal convergence and speed difficult. Similar to Chapter 5 when we introduced the variable $\boldsymbol{V}$ such that $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ we change our simple linear constraints to quadratic constraints. However this is at the same time an advantage since we eliminate the positive semidefinite constraint $\boldsymbol{X} \succeq \boldsymbol{0}$. Another advantage that (6.5) has over other formulations of our problem previously discussed, specifically (5.2), is our updates our explicitly solved for, and there is no need to solve a subproblem if not using `minFunc`, which could take many iterations potentially.

CHAPTER 7

NUMERICAL RESULTS

In this section, we will detail our numerical results from running experiments. We will begin with some background information on our experiments such as how we create our data or adjacency graph, G, how we create initial values of $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ and $\boldsymbol{V}$, and given parameters including how we vary them.

7.1 Initialization

First consider our adjacency matrix of graph G, $\boldsymbol{A}(\boldsymbol{G})$, which is the original N by N matrix of data we are trying to cluster into K disjoint cliques. That is $\boldsymbol{A}(\boldsymbol{G})$ is a symmetric matrix such that $[A(G)]_{ij}$ has value equal to 1 if node i and node j are similar, and is zero if the nodes are not similar. So to ensure that our algorithms are rigorously tested we want to include noise, or extra edges, as well as our K disjoint cliques. We will work with the random case from [Ames and Vavasis, 2014] and the planted K clique graph. To do this we have inputs of N, m, and *prob* where, N is the number of data points, m is the smallest possible size of a clique, and *prob* is the chance of an edge outside a clique, and the knowledge that $K = \lfloor \frac{N}{m} \rfloor$. The general idea of creating such a graph, G, is to create cliques along the diagonal of size at least m, and noise outside these cliques. However it is easy to see that m may not divide evenly into N, and hence we potentially will have outliers causing the size of each clique to possibly not all be the same. How you distribute these outliers will greatly affect your expected solution, so that is where most of the detail in creating G is found. In the following work we have chosen to eliminate outliers by evenly distributing them to cliques, but we could also choose a specific number of outliers to keep in G. For example, say $N = 100$ and $m = 40$ then we would have $K = 2$ with 20 potential

outliers, but we evenly distribute them to each clique giving us $K = 2$ cliques of size 50 and no outliers. Another example would be $N = 105$ and $m = 40$ then we once again have $K = 2$, but now have 25 potential outliers. We can not put half an node in each clique so to evenly distribute them we have one clique of size 53 and another size 52. How we distribute these outliers will greatly affect our expected solutions. So with the above in mind let us discuss our expected solution from [Ames and Vavasis, 2014]. Assume we have K cliques in graph G with size of each clique c_i for $i = 1, ..., K$, then as discussed in Chapter 3 our expected solution would be:

$$\boldsymbol{X}^*_{lj} = \begin{cases} \frac{1}{c_i} & \text{if both } l, j \text{ belong to clique i, } C_i \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

Hence, as we change how the outliers are distributed, therefore changing clique size we have altered our expected solution (7.1). The outputs of our function to create graph G as an adjacency matrix are $\boldsymbol{A(G)}$, K and a vector $\boldsymbol{c}$ containing the size of each planted clique so it may be referenced later. One future question this bring up is, if we have a set number of outliers how does this affect our clustering results.

Now that we made G, the data we want to cluster, we need a starting point for all our algorithms. Though you could start with a random matrix as your starting point, if we start with a good approximation, a matrix resembling clustering, it will help with recovery, and iterations needed to achieve convergence. To do this we reference [Von Luxburg, 2007], a paper detailing spectral clustering, including initialization methods.

Let us discuss some definitions before we detail the algorithm from [Von Luxburg, 2007] to find a starting point $\boldsymbol{V}$ such that $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$. First, the degree matrix $\boldsymbol{D}$ which is an N by N diagonal matrix where D_{ii} is equal to the number of nodes adjacent with node i in our graph G. Second, the weighted matrix, $\boldsymbol{W}$, which describes how strongly similar nodes are in graph G. This is an N by N matrix with

Algorithm 4: Initialization of $\boldsymbol{V}$ and $\boldsymbol{X}$

Data: Input G , K
for *Do the following:*
  $\boldsymbol{W} = G$
  Find $\boldsymbol{D}$ with $D_{ii} = \sum_{j=1}^{N} [A(G)]_{ij}$
  $\boldsymbol{L} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2}$
  Find the K smallest eigenvectors of $\boldsymbol{L}$ call them $\boldsymbol{v}_1, ..., \boldsymbol{v}_K$
  $\boldsymbol{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_K]$
  $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$
end
Data: Output $\boldsymbol{X}$, and $\boldsymbol{V}$

weights, $W_{ij} \in [0, 1]$. In our case $\boldsymbol{W}$ will simply be the adjacency matrix of graph G as our node are similar or not similar with no varying degrees of similarity, but this does not have to be true for this initialization scheme. Finally we need to discuss the normalized graph Laplacian which is an N by N symmetric matrix:

$$\boldsymbol{L_{sym}} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2} = \boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{W}\boldsymbol{D}^{-1/2},$$

where $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ is the unnormalized graph Laplacian defined in [Von Luxburg, 2007]. For a more in depth look at the graph Laplacian, its variants, and properties we refer the reader to [Von Luxburg, 2007]. As we are given $\boldsymbol{W} = \boldsymbol{A}(\boldsymbol{G})$, and $\boldsymbol{D}$ can easily be found we use the second definition to find $\boldsymbol{L_{sym}}$.

Now with a good background on the information matrices needed to find our initial $\boldsymbol{V}$ and $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ let us discuss the algorithm used. In general Algorithm 4 works by finding the K smallest eigenvectors of $\boldsymbol{L}$, and making them the columns of $\boldsymbol{V}$. Then for an initial $\boldsymbol{X}$ in Algorithm 1 we let $\boldsymbol{X} = \boldsymbol{V}\boldsymbol{V}^T$ after finding above $\boldsymbol{V}$. One thing of note is this initialization has a maximum flop counts $\mathcal{O}(N^3)$ for the eigenvalue decomposition, but in practice is smaller as we only need to perform the eigenvalue decomposition to get the K smallest eigenvectors. We have summarized the above process in Algorithm 4.

7.2 Rounding Schemes

Initialization plays a key role in how quickly our algorithm will converge with error less than a specified tolerance, say ϵ. There will be some error in our solution, and we need to determine if our approximate solution, which we will call $\hat{\boldsymbol{X}}$, is a good approximation of the optimal solution. We now discuss how we compare the approximated solution with our expected solution, (7.1), from above. We want speed in our algorithm, but also accuracy so we test for recovery of our expected solution as well. We can look at figures in Chapter 8 of our approximated solution to see the clustering that occurs, but this does us little to compare the solutions. To do this we will cleverly round our approximated solution based on the knowledge of our expected solution's values and cluster sizes. Our rounding scheme will attempt to identify which elements are in cliques, and which are not. Then give the elements in cliques the expected value of one divided by the size of the clique, and elements outside cliques the value of zero. We have worked with different rounding schemes that we will detail below.

As the value in our (7.1) can differ greatly based on the size of cliques, we will need to partition our approximated solution in groups based on cliques size. For example if we return to our previous example of $N = 100$ and $m = 40$ where we get $K = 2$ cliques of size 50. We expect elements in clique i to be roughly $\frac{1}{c_i}$, and elements outside clique i to be close to zero. So our first rounding scheme takes elements of approximate solution, $\hat{\boldsymbol{X}}$, based on their distance from our expected value. So for element $\hat{\boldsymbol{X}}_{lj}$ in clique i columns one possible rounding scheme would be:

$$\hat{\boldsymbol{X}}_{lj} = \begin{cases} \frac{1}{c_i} & \text{if } \hat{\boldsymbol{X}}_{lj} \geq \frac{1}{2}(\frac{1}{c_i}) \\ 0 & \text{if } \hat{\boldsymbol{X}}_{lj} < \frac{1}{2}(\frac{1}{c_i}), \end{cases} \tag{7.2}$$

which identifies larger elements closer to our expected value, and assumes they should be part of clique i. This rounding scheme is intuitive, and can be changed to be more

strict on inclusion in cliques or less strict based on the factor of $\frac{1}{2}$ which we multiply by our expected value. If we decrease the value of $\frac{1}{2}$ as it approaches 0 we will including more and more elements until when we include all values greater than zero. Yet if we increase the value of $\frac{1}{2}$ towards one we will include less and less elements until we only include elements if their value is greater than our equal to the expected value of $\frac{1}{c_i}$. After running experiments and analyzing resulting rounded graph $\hat{\boldsymbol{X}}$, we notice the value of this factor is dependent on K, the desired number of disjoint cliques, and *prob*, the chance of an outside edge. We found with values of K close to two our factor was too large and omitted some elements in cliques. However as K increased towards 10 we needed more strict requirements for membership of a clique. This poses an interesting question of finding a function to describe this factor for optimal recovery during a rounding scheme.

With an attempt to recover cliques at a better rate for smaller N, larger number of desired cliques, K, and larger *prob*, we also consider another rounding scheme. This rounding scheme is more reliant on the mean of each individual column rather than expected value of our solution, (7.1), as in (7.2). In this scheme we find the column mean of each column. Then use some factor, we call Γ, of this for membership in a clique. Once again this factor greatly influences inclusion in cliques and is dependent on K and *prob* as well. Consider the following scheme which we will explain in more detail below:

$$\hat{\boldsymbol{X}}_{lj} = \begin{cases} \frac{1}{c_i} & \text{if } \hat{\boldsymbol{X}}_{lj} \geq \Gamma(\text{column mean}(j)) \text{ for } j = 1, ..., N \\ 0 & \text{if } \hat{\boldsymbol{X}}_{lj} < \Gamma(\text{column mean}(j)) \text{ for } j = 1, ..., N, \end{cases} \tag{7.3}$$

where c_i is the size of cluster i in which column j belongs corresponding to our K planted disjoint cliques. As similar to our factor of $\frac{1}{2}$ in (7.2), Γ is dependent on both K and *prob*. As we increase Γ the requirement for membership in our cliques become more strict and likewise if we decrease Γ the requirement for membership dwindles as

well. Rounding scheme (7.3) works slightly better than (7.2), and could be optimized by finding Γ that would be optimal for each value of K and *prob*.

The last rounding scheme we implement is the use of the function `kmeans` in Matlab to identify cliques from our approximate solution. Due to the structure of our solution, the number of iterations of `kmeans` will be minimal. Hence, this will not be an overly time consuming part of our algorithms.

7.3 Numerical Experiments

Next, we will discuss the parameters of experiments, including N, K, *prob*, and ϵ, we run and how we varied them to test each of our algorithms 1, 2, 3. The first and most obvious parameter is N, the size of our data set from which we create graph G. We started testing at $N = 50$ with goals of working up to $N = 5000$ including $N = 100, 250, 500, 1000$ to have a variety data sets. Additionally, we have the value K, which is desired number of disjoint cliques to find in our graph G, and is directly related to the smallest size clique. We have fluctuated this value from $K = 2$ to $K = 10$ by one throughout experiments by choice of smallest size clique. The tolerance for convergence, ϵ, was set at 10^{-5} with number of trials for each parameter set at ten. For recovery of our expected solution, (7.1) from the approximated solution, $\hat{\boldsymbol{X}}$, we compared (7.1) with our rounded approximate solution $\hat{\boldsymbol{X}}$. If the norm of (7.1) minus $\hat{\boldsymbol{X}}$ divided by the norm of (7.1) was less than or equal to 10^{-1} we accepted the expected solution as recovered. Otherwise the expected solution, (7.1), was not recovered in that given experiment. Lastly, we then recorded the error, recovery, and time for each trial in each experiment.

We move onto the actual experiments and results from them. First, let us list all codes which we used to solve our SDP relaxation of the K Disjoint Clique problem (3.1) in various forms detailed in previous chapters 1, 2, and 3. As a control, we have used `CVX` from [Grant et al., 2008] to solve SDP's when $N \leq 250$. This is a general convex optimization solver. That has an SDP option, and by default uses `SDPT3` [Grant

et al., 2008] as the solver. `SDPT3` uses interior point methods, detailed previously in Chapter 2, to solve the SDP relaxation [Toh et al., 1999]. We have used code based on Algorithm 1 which we will refer to as `ADMM`. As well as two separates codes based on Algorithm 2. We will refer to as `BM2`, for smaller N as $\boldsymbol{AV}$ is not hard coded here, and `BMHC`, where we hard code $\boldsymbol{AV}$ as detailed in Chapter 8. Then our last set of codes based on Algorithm 3 which we will refer to as `BCR`, for when gradient descent is used to solve for $\boldsymbol{V}$, and `BCRMF` when `minFunc` [Schmidt, 2005] is used to solve for $\boldsymbol{V}$. Now for a comprehensive list of all experiments run:

1. $N = 50$ for `CVX, ADMM, BM2, BMHC, BCR, BCRMF` with number of cliques $K = 2, 3, 4, 5, 6, 7, 8, 10$ and $prob = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ and ten trials for each set of conditions

2. $N = 80$ for `CVX, ADMM, BM2, BMHC, BCR, BCRMF` with number of cliques $K = 2, 3, 4, 5, 6, 7, 8, 10$ and $prob = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ and ten trials for each set of conditions

3. $N = 100$ for `CVX, ADMM, BM2, BMHC, BCR, BCRMF` with number of cliques $K = 2, 3, 4, 5, 6, 7, 8, 9, 10$ and $prob = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ and ten trials for each set of conditions

4. $N = 250$ for `CVX, ADMM, BMHC, BCRMF` with number of cliques $K = 2, 3, 4, 5, 6, 7, 8, 9, 10$ and $prob = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$ and ten trials for each set of conditions

Note that experiments run with `CVX` were run on my laptop while all others where run on the HPC at The University of Alabama. We have dropped both `BM` and `BCR` as N increases because their counter part was more efficient in solving the relaxed SDP.

First let us consider the unrounded results of our approximated solution, $\hat{\boldsymbol{X}}$. We have some examples as graphs in Chapter 8, and compare them to our expected

disjoint K clique graph described in (7.1). We also comment on the clustering patterns in our approximated solutions. First, note that are approximated solutions are not perfect but rather have some noise as we have solved up to a certain tolerance. For the figures displayed the color scale goes from blue, representing no similarity, to red, indicating similarity between nodes i and j in our approximated solution. Though our answers are not exact, we can see general patterns of clustering or cliques from all codes for given parameters. As the size of the clique increases and probability of an outside edge decreases our cliques become very evident for example Figure 9.11 or Figure 9.13. However cliques are more difficult to pick out as clique size decreases and probability of an outside edge increases such as in Figure 9.14 and 9.16. This matches the recovery guarantee from [Ames and Vavasis, 2014] and the pattern continues in the rounded results as well.

We now present rounded results of our $\hat{\boldsymbol{X}}$ with figures in Chapter 8. By analyzing the unrounded results and rounded results, one can see that the rounding schemes, (7.2), (7.3), and `kmeans` generally capture the clusters that our experiment finds. It is of note is how often the graph was very close to the established threshold. That is, rounded results represent desired cliques with little noise, even if the approximated solution was unacceptable for recovery. Also our recovery is comparable to the recovery presented by Ames and Vavasis in [Ames and Vavasis, 2014] as show in Figures 7.1, 7.2, and 7.3.

As expected our recovery is better with less noise and larger clique. We mention that as N grows and more experiments are run these recovery results will become even more comparable with [Ames, 2014]. As sizes of cliques grow the algorithms and rounding scheme approximate our expected solution with better accuracy. Lastly, we discuss the time required to solve the relaxed SDP, since it served as motivation to move away from interior point methods used in [Ames and Vavasis, 2014]. To do this we created 100 random graphs for each N and recored run time for each N. Clearly

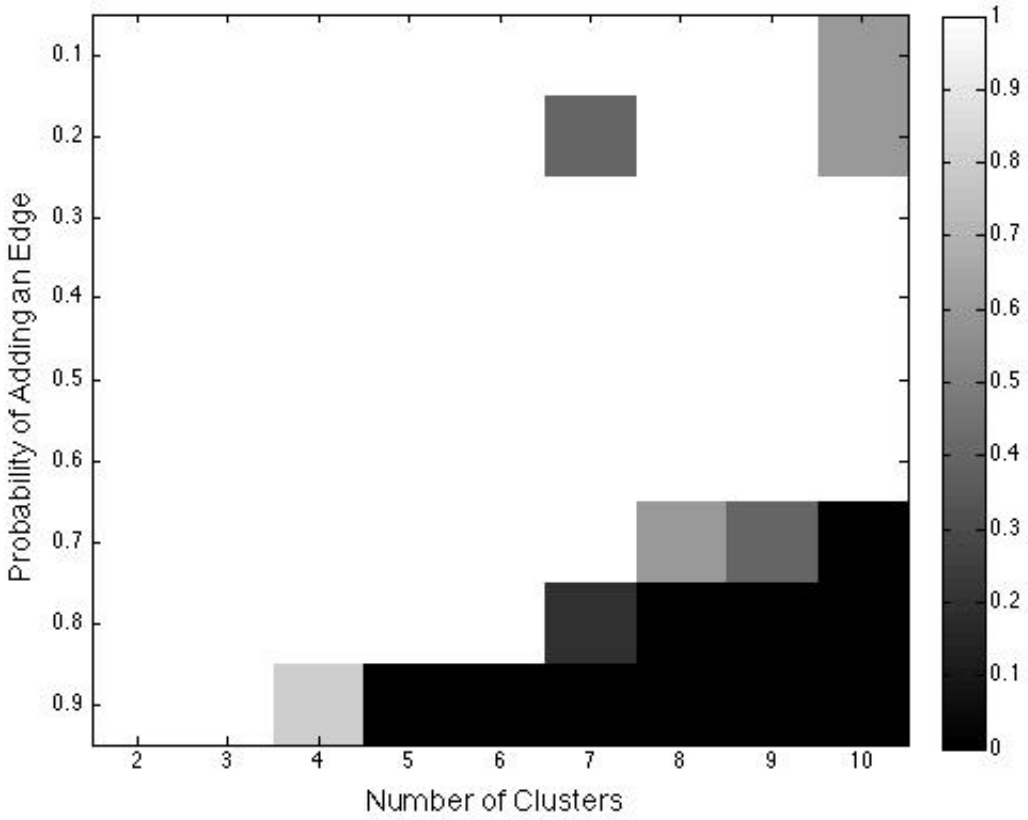

Figure 7.1: Recovery for `ADMM` with $N = 250$

represented in Figures 7.4, 7.5, 7.6, and 7.7 below, our optimized Algorithms 1, 2, 3 solve the relaxed SDP more efficiently than `CVX`, which uses interior point methods. While this is clearly evident in the smallest of N that we test, it would only be more prominent as N increases, but due to time and memory constraints we restrict our tests of `CVX` to smaller N. We can also conclude that `BM2` takes more time than `BMHC`. This is due to the calculation of $\boldsymbol{AV}$ by matrix multiplication in `BM2` versus analytic formulas in `BMHC`. Also it is clear that `BCR` converges slowest of our algorithms due to the large number of iterations it requires for convergence.

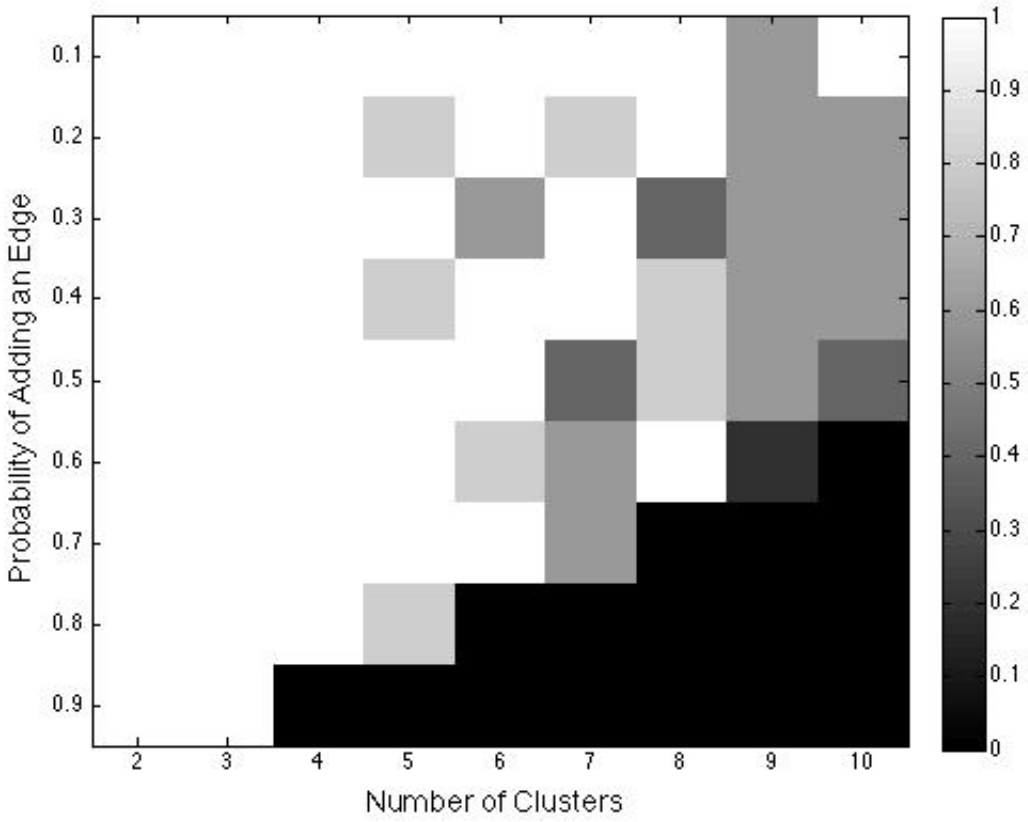

Figure 7.2: Recovery for `BMHC` with $N = 250$

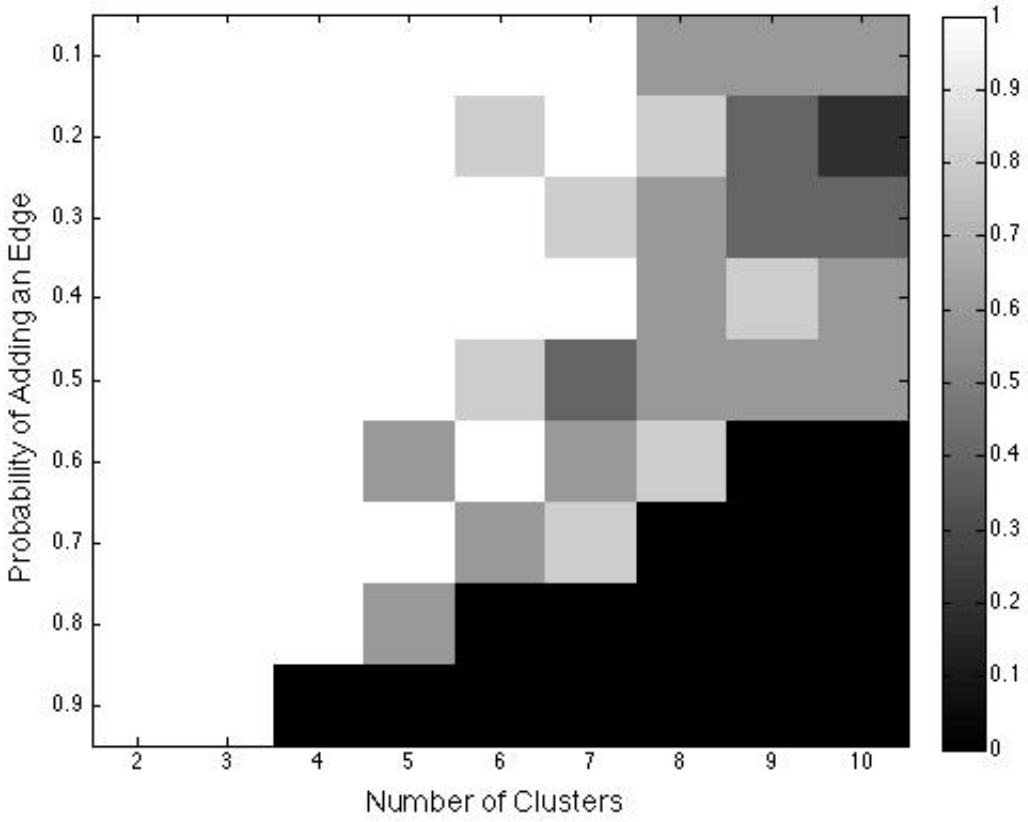

Figure 7.3: Recovery for `BCRMF` with $N = 250$

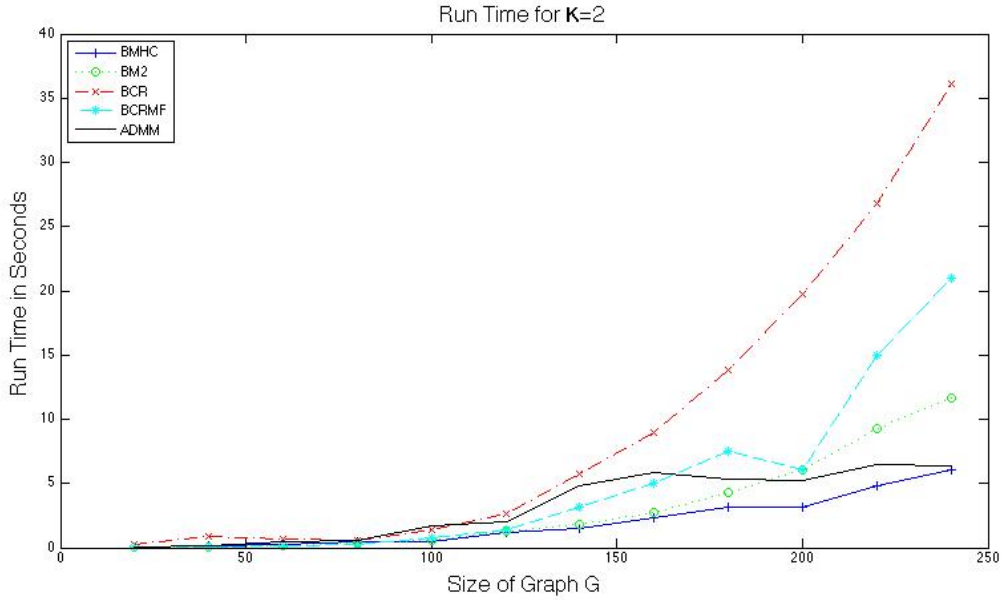

Figure 7.4: Run Time for Algorithms with $K = 2$

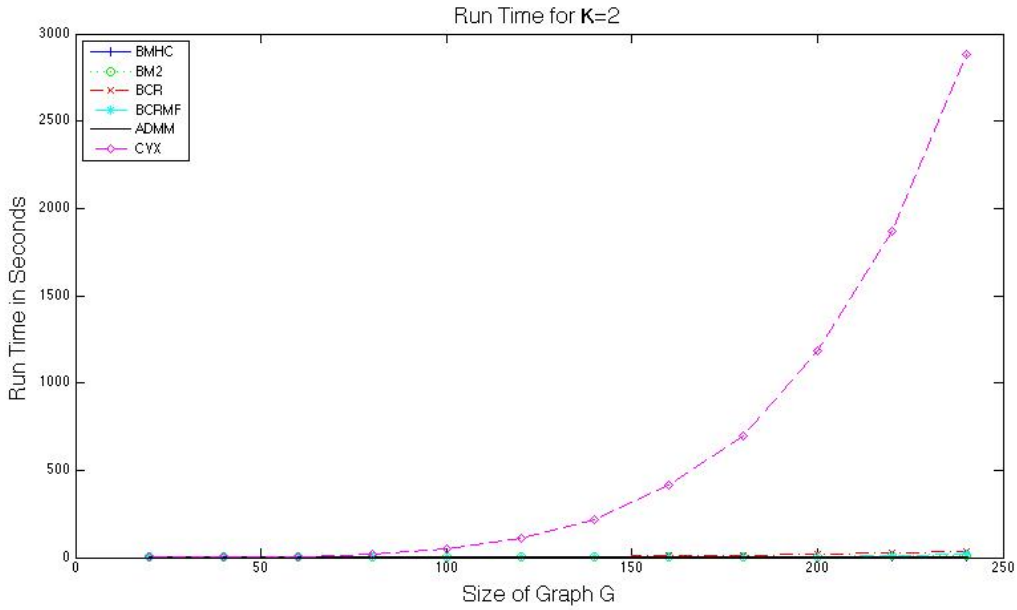

Figure 7.5: Run Time for Algorithms with $K = 2$

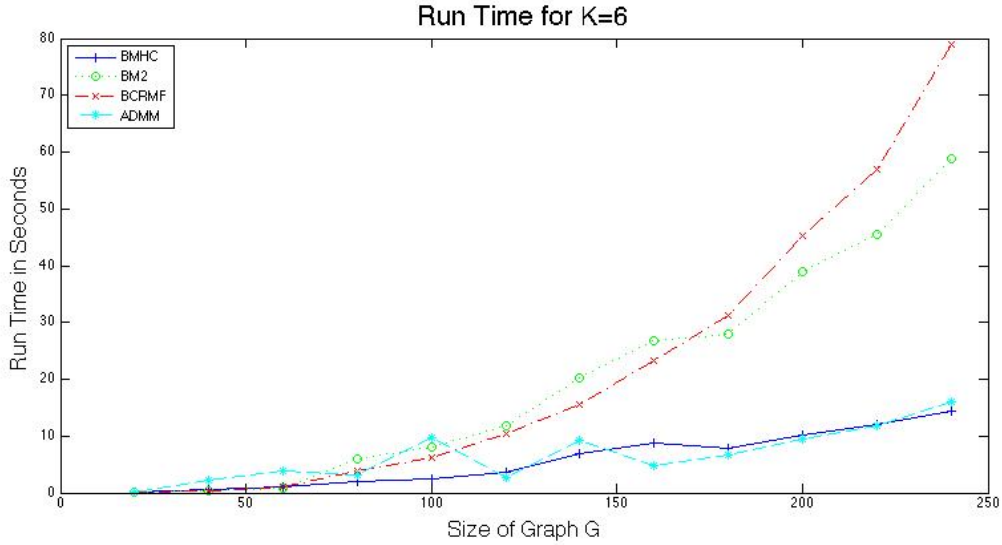

Figure 7.6: Run Time for Algorithms with $K = 6$

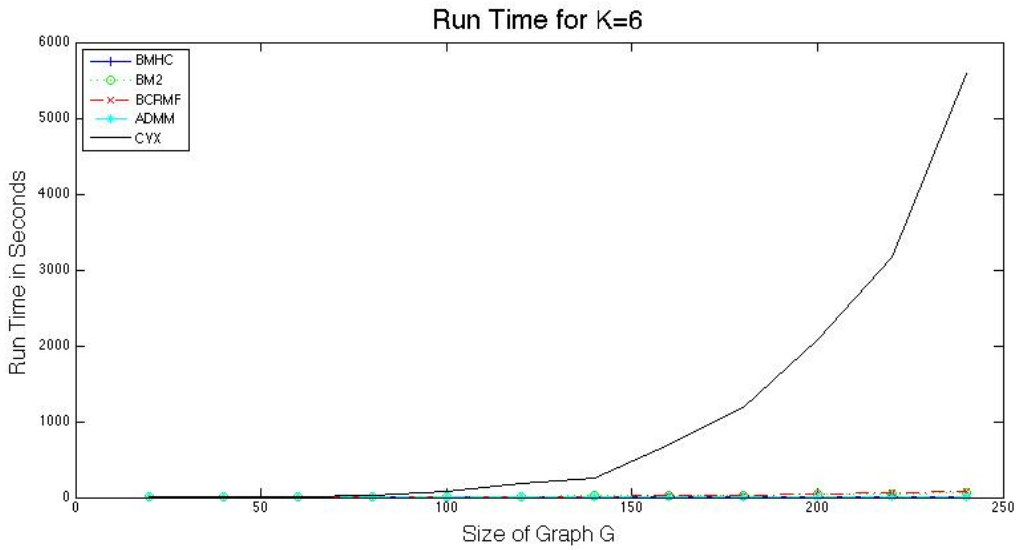

Figure 7.7: Run Time for Algorithms with $K = 6$

CHAPTER 8

CONCLUSIONS

In conclusion, we have presented material on three techniques to solve the K Disjoint Clique problem. We used Alternating Direction Method of Multipliers in Chapter 4, a low rank factorization with method of multipliers in Chapter 5, and a low rank factorization with alternating direction in Chapter 6. Each of which provides an improvement of per iteration computational cost without sacrificing accuracy of solution when compared to classical interior point methods. This can be seen in figures for timing and recovery.

Finally we discuss future work this prompts. One thing that limited our testing was memory constraints. We seek an algorithm that is more memory efficient and could be run for larger N. Next would be the question of recovery, and how to improve it as *prob* and K increase. One component of this is the rounding schemes, (7.2) and (7.3), and the factor or Γ used in each one. It is clear from our results that these constants depend on both K and *prob*. That is, finding a function to determine factor of Γ that would more accurately recovery the correct cliques would be of great interest. Even with a more optimized rounding scheme another feature of this would be to explore theoretical guarantees of convergence to the optimal solution of the K Disjoint Clique problem, (3.1). This would be continuing work from [Ames and Vavasis, 2014]. Another open question would be to continue to optimize run time or computational cost. One intriguing thought would be to decrease the number of flops per iteration in Algorithm 2 with the exploration of stochastic gradient methods rather than using gradient methods. Lastly, we would want to apply these techniques to similar SDP's. These techniques would have direct application to the weighted K Disjoint Clique problem.

We would want to explore it next followed by other optimization modeled as SDP's.

REFERENCES

[Abbe et al., 2016] Abbe, E., Bandeira, A. S., and Hall, G. (2016). Exact recovery in the stochastic block model. *Information Theory, IEEE Transactions on*, 62(1):471–487.

[Ailon et al., 2013] Ailon, N., Chen, Y., and Huan, X. (2013). Breaking the small cluster barrier of graph clustering. *arXiv preprint arXiv:1302.4549*.

[Ames, 2014] Ames, B. P. (2014). Guaranteed clustering and biclustering via semidefinite programming. *Mathematical Programming*, 147(1-2):429–465.

[Ames and Vavasis, 2014] Ames, B. P. and Vavasis, S. A. (2014). Convex optimization for the planted k-disjoint-clique problem. *Mathematical Programming*, 143(1-2):299–337.

[Amini and Levina, 2014] Amini, A. A. and Levina, E. (2014). On semidefinite relaxations for the block model. *arXiv preprint arXiv:1406.5647*.

[Berkhin, 2006] Berkhin, P. (2006). A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer.

[Boyd et al., 2011] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., et al. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[Burer and Monteiro, 2003] Burer, S. and Monteiro, R. D. (2003). A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(2):329–357.

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

[Cai et al., 2015] Cai, T. T., Li, X., et al. (2015). Robust and computationally feasible community detection in the presence of arbitrary outlier nodes. *The Annals of Statistics*, 43(3):1027–1059.

[Chen et al., 2016] Chen, C., He, B., Ye, Y., and Yuan, X. (2016). The direct extension of admm for multi-block convex minimization problems is not necessarily convergent. *Mathematical Programming*, 155(1-2):57–79.

[Chen et al., 2014a] Chen, Y., Jalali, A., Sanghavi, S., and Xu, H. (2014a). Clustering partially observed graphs via convex optimization. *The Journal of Machine Learning Research*, 15(1):2213–2238.

[Chen et al., 2014b] Chen, Y., Sanghavi, S., and Xu, H. (2014b). Improved graph clustering. *Information Theory, IEEE Transactions on*, 60(10):6440–6455.

[Chen and Xu, 2014] Chen, Y. and Xu, J. (2014). Statistical-computational phase transitions in planted models: The high-dimensional setting. In *ICML*, pages 244–252.

[Cherney et al., 2016] Cherney, D., Denton, T., Thomas, R., and Waldron, A. (2016). *Linear Algebra*.

[Douglas and Gunn, 1964] Douglas, J. and Gunn, J. E. (1964). A general formulation of alternating direction methods. *Numerische Mathematik*, 6(1):428–453.

[Duchi and Singer, 2009] Duchi, J. and Singer, Y. (2009). Efficient online and batch learning using forward backward splitting. *Journal of Machine Learning Research*, 10(Dec):2899–2934.

[Freund, 2004] Freund, R. M. (2004). Introduction to semidefinite programming (sdp). *Massachusetts Institute of Technology*.

[Gabay, 1983] Gabay, D. (1983). Chapter ix applications of the method of multipliers to variational inequalities. In *Studies in mathematics and its applications*, volume 15, pages 299–331. Elsevier.

[Gabay and Mercier, 1976] Gabay, D. and Mercier, B. (1976). A dual algorithm for the solution of nonlinear variational problems via finite element approximation. *Computers & Mathematics with Applications*, 2(1):17–40.

[Glowinski and Marroco, 1975] Glowinski, R. and Marroco, A. (1975). Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires. *Revue francaise d'automatique, informatique, recherche opérationnelle. Analyse numérique*, 9(R2):41–76.

[Gorski et al., 2007] Gorski, J., Pfeuffer, F., and Klamroth, K. (2007). Biconvex sets and optimization with biconvex functions: a survey and extensions. *Mathematical methods of operations research*, 66(3):373–407.

[Grant et al., 2008] Grant, M., Boyd, S., and Ye, Y. (2008). Cvx: Matlab software for disciplined convex programming.

[Guédon and Vershynin, 2015] Guédon, O. and Vershynin, R. (2015). Community detection in sparse networks via grothendieck's inequality. *Probability Theory and Related Fields*, pages 1–25.

[Hajek et al., 2015] Hajek, B., Wu, Y., and Xu, J. (2015). Achieving exact cluster recovery threshold via semidefinite programming. In *Information Theory (ISIT), 2015 IEEE International Symposium on*, pages 1442–1446. IEEE.

[Hestenes, 1969a] Hestenes, M. (1969a). Multipliers and gradient methods—in computing met h ods in optimization problems, vol. 2, la zadeh, lw neustadt, av balakrishnan eds.

[Hestenes, 1969b] Hestenes, M. R. (1969b). Multiplier and gradient methods. *Journal of optimization theory and applications*, 4(5):303–320.

[Holland et al., 1983] Holland, P. W., Laskey, K. B., and Leinhardt, S. (1983). Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137.

[Kulis et al., 2007] Kulis, B., Surendran, A. C., and Platt, J. C. (2007). Fast low-rank semidefinite programming for embedding and clustering. In *Artificial Intelligence and Statistics*, pages 235–242.

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

[Lin et al., 2015] Lin, T.-Y., Ma, S.-Q., and Zhang, S.-Z. (2015). On the sublinear convergence rate of multi-block admm. *Journal of the Operations Research Society of China*, 3(3):251–274.

[Mathieu and Schudy, 2010] Mathieu, C. and Schudy, W. (2010). Correlation clustering with noisy input. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 712–728. Society for Industrial and Applied Mathematics.

[Mirkin, 1998] Mirkin, B. (1998). Mathematical classification and clustering: From how to what and why. In *Classification, data analysis, and data highways*, pages 172–181. Springer.

[Nellore and Ward, 2013] Nellore, A. and Ward, R. (2013). Recovery guarantees for exemplar-based clustering. *arXiv preprint arXiv:1309.3256*.

[Nocedal and Wright, 2006] Nocedal, J. and Wright, S. J. (2006). *Conjugate gradient methods*. Springer.

[Oymak and Hassibi, 2011] Oymak, S. and Hassibi, B. (2011). Finding dense clusters via "low rank + sparse" decomposition. *Arxiv preprint arXiv:1104.5186*.

[Peng, 2005] Peng, J. (2005). 0-1 semidefinite programming for spectral clustering: Modeling and approximation. Technical report, Citeseer.

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

[Petersen et al., 2008] Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15):510.

[Powell, 1969] Powell, M. (1969). A method for nonlinear constraints in minimization problems in optimization ed. by r.

[Qin and Rohe, 2013] Qin, T. and Rohe, K. (2013). Regularized spectral clustering under the degree-corrected stochastic blockmodel. In *Advances in Neural Information Processing Systems*, pages 3120–3128.

[Renegar, 2014] Renegar, J. (2014). Efficient first-order methods for linear programming and semidefinite programming. *arXiv preprint arXiv:1409.5832*.

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

[Schmidt, 2005] Schmidt, M. (2005). minfunc: unconstrained differentiable multivariate optimization in matlab. *Software available at http://www. cs. ubc. ca/˜ schmidtm/Software/minFunc. htm*.

[Shah et al., 2016] Shah, S., Yadav, A. K., Castillo, C. D., Jacobs, D. W., Studer, C., and Goldstein, T. (2016). Biconvex relaxation for semidefinite programming in computer vision. In *European Conference on Computer Vision*, pages 717–735. Springer.

[Toh et al., 1999] Toh, K.-C., Todd, M. J., and Tütüncü, R. H. (1999). Sdpt3—a matlab software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581.

[Tu and Wang, 2014] Tu, S. and Wang, J. (2014). Practical first order methods for large scale semidefinite programming.

[Tuncel, 2016] Tuncel, L. (2016). *Polyhedral and semidefinite programming methods in combinatorial optimization*, volume 27. American Mathematical Soc.

[Vandenberghe and Boyd, 1996] Vandenberghe, L. and Boyd, S. (1996). Semidefinite programming. *SIAM review*, 38(1):49–95.

[Vinayak et al., 2014] Vinayak, R. K., Oymak, S., and Hassibi, B. (2014). Sharp performance bounds for graph clustering via convex optimization. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 8297–8301. IEEE.

[Von Luxburg, 2007] Von Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

[Wang and Carreira-Perpinán, 2013] Wang, W. and Carreira-Perpinán, M. A. (2013). Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. *arXiv preprint arXiv:1309.1541*.

[Zheng and Lafferty, 2015] Zheng, Q. and Lafferty, J. (2015). A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements. In *Advances in Neural Information Processing Systems*, pages 109–117.

APPENDIX

9.1 Proof for $\boldsymbol{X}$ Update in ADMM Algorithm

This section is to discuss in more detail the proof of our $\boldsymbol{X}$ update in the ADMM Algorithm 1, discussed earlier in Chapter 4 with update (4.8). The main substance and thought of the proof including Algorithm 5 used can be found in [Wang and Carreira-Perpinán, 2013]. Recall that we were trying to find the solution to (4.7):

$$\begin{aligned} \arg\min \ & ||\boldsymbol{X} - \boldsymbol{U_X}||_F^2 \\ \text{s.t} \ & tr(\boldsymbol{X}) = K, \\ & \boldsymbol{X} \succeq 0, \end{aligned}$$

with $\boldsymbol{U_X} = \frac{1}{2}(\frac{1}{\beta}\boldsymbol{\Lambda_{XY}} + \frac{1}{\beta}\boldsymbol{\Lambda_{XZ}} - \boldsymbol{Y} - \boldsymbol{Z})$. We will use the spectral decomposition of $\boldsymbol{U_X} = \boldsymbol{EDE}^T$, and reduce our problem to a problem of vectors:

$$\begin{aligned} \arg\min_{\boldsymbol{d}} \ & ||\boldsymbol{d} - \boldsymbol{u}||_2^2 \\ \text{s.t} \ & \sum_{i=1}^{N} d_i = K, \\ & d_i > 0 \ \text{ for } \ i = 1, ..., N, \end{aligned}$$

with vectors $\boldsymbol{d}$ and $\boldsymbol{u} = \boldsymbol{E}^T\boldsymbol{U_X}\boldsymbol{E}$ both N vectors which is equivalent to the problem solved in [Wang and Carreira-Perpinán, 2013] and we can formulate:

$$\begin{aligned} \arg\min_{\boldsymbol{d}} \ & \frac{1}{2}||\boldsymbol{d} - \boldsymbol{u}||^2 \\ \text{s.t} \ & \boldsymbol{d}^T\boldsymbol{e} = K, \\ & d_i \geq 0. \end{aligned} \tag{9.1}$$

We will prove this using the KKT Conditions on the Lagrangian of (9.1):

$$\mathcal{L}(\boldsymbol{d}, \lambda, \boldsymbol{\mu}) = \frac{1}{2}||\boldsymbol{d} - \boldsymbol{u}||^2 - \lambda(\boldsymbol{d}^T\boldsymbol{e} - K) - \boldsymbol{\mu}^T\boldsymbol{d}$$

where $\lambda \in \mathbf{R}$ and $\boldsymbol{\mu} \in \mathbf{R}^N$ are dual variables. Using this Lagrangian we get the following KKT Conditions:

$$\begin{aligned} \boldsymbol{d} - \boldsymbol{u} - \lambda\boldsymbol{e} - \boldsymbol{\mu} &= \mathbf{0}, \\ d_i \geq 0, \quad \boldsymbol{d}^T\boldsymbol{e} &= K, \\ \mu_i &\geq 0, \\ \boldsymbol{d}^T\boldsymbol{\mu} &= 0, \end{aligned} \tag{9.2}$$

which we can reduce to focus on each $i = 1, ..., N$ and then solve for d_i to find $\boldsymbol{d}$

$$\begin{aligned} d_i - u_i - \lambda - \mu_i &= 0, \\ d_i \geq 0, \quad \boldsymbol{d}^T\boldsymbol{e} &= K, \\ \mu_i &\geq 0, \\ d_i\mu_i &= 0. \end{aligned} \tag{9.3}$$

@e break this down into cases, consider the following cases.

CASE I: Where $d_i > 0$ then by complementary slackness $\mu_i = 0$ and $d_i = u_i + \lambda > 0$ where we still need to find λ.

CASE II: Where $d_i = 0$ then $\mu_i \geq 0$ giving us $u_i + \lambda = -\mu_i \leq 0$ by stationary and dual feasibility of $\boldsymbol{\mu}$.

From here we can easily see that $d_i = 0$ corresponds to the smaller components of $\boldsymbol{u}$ so our objective is minimized. Then without loss of generality we can order $\boldsymbol{u}$ and $\boldsymbol{d}$ with the same ordering:

$$u_1 \geq u_2 \geq ... \geq u_\rho \geq u_{\rho+1} \geq ... \geq u_N$$

$$d_1 \geq d_2 \geq ... \geq d_\rho \geq d_{\rho+1} \geq ... \geq d_N,$$

where $d_{\rho+1}, ..., d_N = 0$, and ρ is the number of positive components of $\boldsymbol{d}$ with the rest equal to zero since $d_i \geq 0$. Now consider primal feasibility $\sum_{i=1}^{N} d_i = K$, and verify the following chain of equalities using our knowledge on $\boldsymbol{d}$ with some equal to zero and others greater than zero:

$$K = \sum_{i=1}^{N} d_i = \sum_{i=1}^{\rho} d_i = \sum_{i=1}^{\rho} (u_i + \lambda) = \sum_{i=1}^{\rho} (u_i) + \rho\lambda.$$

Recall that we know u_i so we need to solve this for λ to get:

$$\lambda = \frac{1}{\rho}\left(K - \sum_{i=1}^{\rho} u_i\right),$$

from here if we can find ρ we can find $\boldsymbol{d}$, and get our solution $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$ with $\boldsymbol{D} = \text{Diag}(\boldsymbol{d})$ where $\text{Diag}(\boldsymbol{d})$ takes the vector $\boldsymbol{d}$ and place it in the diagonal of $\boldsymbol{D}$.

To complete our proof we need to show that ρ is the largest number $j = 1, ..., N$ such that $u_j + \lambda_j > 0$ so $d_j = u_j + \lambda_j > 0$ if $j \leq \rho$ otherwise $u_j\lambda_j < 0$ corresponding to $d_j = 0$. Where

$$\lambda_j = \frac{1}{j}\left(K - \sum_{i=1}^{j} u_i\right),$$

we will once again consider cases:

CASE I: Let $j = \rho$ then we need to show $u_j + \lambda_j > 0$:

$$u_j + \lambda_j = u_\rho + \lambda_\rho = u_\rho + \frac{1}{\rho}\left(K - \sum_{i=1}^{\rho} u_i\right) = d_\rho > 0$$

by previous assumption of $\boldsymbol{d}$.

CASE II: Let $j < \rho$ then we need to show $u_j + \lambda_j > 0$:

$$\begin{aligned}
u_j + \lambda_j &= u_j + \frac{1}{j}\left(K - \sum_{i=1}^{j} u_i\right) \\
&= \frac{1}{j}\left(ju_j + K - \sum_{i=1}^{j} u_i\right) \\
&= \frac{1}{j}\left(ju_j + \sum_{i=j+1}^{\rho} u_i + K - \sum_{i=1}^{\rho} u_i\right) \\
&= \frac{1}{j}\left(ju_j + \sum_{i=j+1}^{\rho} u_i + \lambda\rho\right) \\
&= \frac{1}{j}\left(j(u_j + \lambda) + \sum_{i=j+1}^{\rho} (u_i + \lambda)\right) > 0,
\end{aligned}$$

since $u_i + \lambda > 0$ for $i = j, ..., \rho$ and $u_j + \lambda > 0$ when $j < \rho$.

CASE III: Let $j > \rho$ then we need to show $u_j + \lambda_j < 0$:

$$\begin{aligned}
u_j + \lambda_j &= u_j + \frac{1}{j}\left(K - \sum_{i=1}^{j} u_i\right) \\
&= \frac{1}{j}\left(ju_j + K - \sum_{i=1}^{j} u_i\right) \\
&= \frac{1}{j}\left(ju_j + \sum_{i=\rho+1}^{j} u_i + K - \sum_{i=1}^{\rho} u_i\right) \\
&= \frac{1}{j}\left(ju_j + \sum_{i=\rho+1}^{j} u_i + \lambda\rho\right) \\
&= \frac{1}{j}\left(\rho(u_j + \lambda) + \sum_{i=\rho+1}^{j} (u_j - u_i)\right) < 0,
\end{aligned}$$

since $u_i + \lambda < 0$ for $j > \rho$ and $u_j - u_i < 0$ since $\boldsymbol{u}$ is sorted in descending order.

Hence if ρ is the number of positive components of $\boldsymbol{d}$ then

$$\rho = \max\left\{1 \leq j \leq N : u_j + \frac{1}{j}(K - \sum_{i=1}^{j} u_i) > 0)\right\},$$

Algorithm 5: Finding $\boldsymbol{d}$, ρ, to find $\boldsymbol{X}$ in $\boldsymbol{X}$ update of ADMM

Data: Input $\boldsymbol{u} = \boldsymbol{E}^T \boldsymbol{U_X} \boldsymbol{E}$

for

sort $\boldsymbol{u}$ into $\boldsymbol{x} : x_1 \geq x_2 \geq ... \geq x_N$

$\rho = \max\{1 \leq j \leq N : x_j + \frac{1}{j}(K - \sum_{i=1}^{j} x_i) > 0)\}$

$\lambda = \frac{1}{\rho}(K - \sum_{i=1}^{\rho} u_i)$

$y_i^{k+1} = y_k^k - \sigma_k(tr(\boldsymbol{A_i} \boldsymbol{V} \boldsymbol{V}^T) - b_i) \ \ \forall \ \ i$

$d_i = \max\{u_i + \lambda, 0\}, \ i = 1, ..., N$

end

Data: Output $\boldsymbol{d}, \ \rho$

which gives us an algorithm discussed in [Wang and Carreira-Perpinán, 2013] on how to find ρ giving us λ and $\boldsymbol{d}$ which gets our solution $\boldsymbol{X} = \boldsymbol{E}\boldsymbol{D}\boldsymbol{E}^T$ with $\boldsymbol{D} = \text{Diag}(\boldsymbol{d})$.

9.2 Finding $\boldsymbol{C}$, and $\boldsymbol{A}$'s

In this section we will explore in detail how to find the data matrices $\boldsymbol{C}$, and $\boldsymbol{A}$'s needed in Algorithm 2. First let us recall semidefinite program, (3.6) which is equivalent to the optimization problem (3.1):

$$\begin{aligned}
\min \quad & -\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij} \\
\text{s.t.} \quad & \boldsymbol{X}\boldsymbol{e} \leq \boldsymbol{e}, \\
& X_{ij} = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0,
\end{aligned}$$

where $\boldsymbol{e} \in \mathbf{R}^N$ is the vector of ones, our graph G has N vertices, E is the set of edges in our graph G, K is the desired number of clusters, and $\boldsymbol{X} \in \mathbf{R}^{N\times N}$ is our variable. From here we rewrite this semidefinite program using the trace, tr, or inner product function to get (5.1):

$$\begin{aligned}
\min \quad & -tr(\boldsymbol{C}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\boldsymbol{A_i}\boldsymbol{X}) \leq 1 \ \ \forall\, i = 1, ...N, \\
& tr(\boldsymbol{A_i}\boldsymbol{j}\boldsymbol{X}) = 0, \ \ \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0,
\end{aligned}$$

where the $\boldsymbol{A}$'s are data matrices which we strive to make symmetric, and the right hand side of our constraints can be grouped into vector $\boldsymbol{b}$ for ease of notation. These are the $\boldsymbol{A}$'s which we will construct below. We begin with the general thought for all before we find them in detail. Recall that the $tr(\boldsymbol{X})$ for any square matrix $\boldsymbol{X}$ is the sum of the diagonal entries or $tr(\boldsymbol{X}) = \sum_{i=1}^{N} X_{ii}$ for an N by N matrix $\boldsymbol{X}$.

We will look at finding each data matrix one by one with the goal of them all being symmetric.

CASE I: So let us start with the objective function where we want to rewrite $-\sum_{i=1}^{N}\sum_{j=1}^{N} X_{ij}$ as $-tr(\boldsymbol{CX})$. Notice in (3.6) this is simply the sum of all entries in $\boldsymbol{X}$. So we need to find $\boldsymbol{C}$ that moves entries of a corresponding row or column to the diagonal so that when we take the trace all entries will be included. For this to be true, one can easily see that $\boldsymbol{C}$ should be the matrix of all ones with size N by N. Since, we only need to consider the diagonal entries, $\boldsymbol{CX}$ sums a column of $\boldsymbol{X}$, and places that value in the corresponding diagonal entry for that column. For example the i-th column of $\boldsymbol{X}$ is added, and placed in $[CX]_{ii}$ which will be included in $tr(\boldsymbol{CX})$.

CASE II: Consider the first constraint corresponding to the row sum of $\boldsymbol{X}$ for each row. We want to rewrite $\boldsymbol{Xe} \leq \boldsymbol{e}$ as $tr(\boldsymbol{A_iX}) \leq 1$ for $i = 1, ...N$. So there will be N data matrices $\boldsymbol{A_i}$, let us consider the i-th row sum. We need $\boldsymbol{A_iX}$ to put only the i-th row of $\boldsymbol{X}$ into the diagonal. To do this consider the following data matrix $\boldsymbol{A_i}$:

$$\boldsymbol{A_i} = \begin{cases} 1 & \text{for the } (i,i) \text{ entry} \\ \frac{1}{2} & \text{for the } i\text{th row and column excluding the diagonal entry} \\ 0 & \text{otherwise;} \end{cases}$$

If one works out the multiplication of $\boldsymbol{A_iX}$ one will see it sums the i-th row of $\boldsymbol{X}$ and places it in the i-th diagonal entry. This will work for all $i = 1, ..., N$ and is symmetric consistent with what we desire.

CASE III: Now looking at the second constraint, concerning the (i, j) entry being equal to zero if (i, j) is not an edge in our graph G. We want to rewrite $X_{ij} = 0, \ \forall\ (i,j) \notin E$ s.t. $i \neq j$ as $tr(\boldsymbol{A_{ij}X}) = 0, \ \forall\ (i,j) \notin E$ s.t. $i \neq j$. Therefore we need $\boldsymbol{A_{ij}}$, when multiplied by $\boldsymbol{X}$, to put the (i, j)-th entry in the diagonal. To

accomplish this consider the data matrix $\boldsymbol{A_{ij}}$:

$$\boldsymbol{A_{ij}} = \begin{cases} \frac{1}{2} & \text{for the } (i,j) \text{ entry} \\ \frac{1}{2} & \text{for the } (j,i) \text{ entry} \\ 0 & \text{otherwise.} \end{cases}$$

With careful calculations one can see that $\boldsymbol{A_{ij}X}$ takes the (i,j)-th entry from $\boldsymbol{X}$ and puts half in the i-th diagonal and the other half in the j-th diagonal by virtue of $\boldsymbol{X}$ begin positive semidefinite. This will work for all (i,j) not in E. The number of data matrices of this form will be the number of zeros in our initial graph G above the diagonal.

CASE IV: If we now consider the last constraint, in which the rewrite is trivial, we want to rewrite $tr(\boldsymbol{X}) = K$ as $tr(\boldsymbol{AX}) = K$. This is clearly the data matrix $\boldsymbol{A}$ which is equal to the identity of size N by N since we desire to have the $tr(\boldsymbol{X}) = K$.

With the above calculations, we can rewrite our SDP (3.6) as an SDP with the trace inner product as (5.1). These calculations allow us to apply a low rank factorization with augmented Lagrangian detailed in the Chapter 5.

9.3 Finding $\boldsymbol{CV}$ and $\boldsymbol{AV}$'s

After finding analytic formulas the $\boldsymbol{A}$'s as sparse matrices, we can use these formulas to find any multiplications involving $\boldsymbol{A}$ during the subproblem. In the analytic formula verison of Algorithm 2, `BMHC`, we have used these analytic formulas for $\boldsymbol{A_i V}$ for each i to speed up calculations as N increases and eliminate the need to store matrix $\boldsymbol{A}$. Though the multiplication $\boldsymbol{CV}$ and $\boldsymbol{AV}$ are used sparingly, only in the gradient calculation, it is dense matrix, and time consuming as we work with large scale problems. Even with $\boldsymbol{AV}$ a dense matrix finding a formula for it is rather simple with our sparse $\boldsymbol{A}$. Consider each of the following cases for our different matrices $\boldsymbol{A}$:

CASE I: First start with the matrix multipilication of $\boldsymbol{CV}$ which is part of the first term in our gradient of the augmented Lagrangain (4.5). We have $\boldsymbol{C}$ as the matrix of all ones so we can find $\boldsymbol{CV}$:

$$[CV]_{ij} = \begin{cases} \sum_{l=1}^{N} V_{lj} \text{ for } i = 1, ..., N \text{ and } j = 1, ..., N \end{cases}$$

CASE II: With $\boldsymbol{A_i}$ concerning the row sum constraint: We have the data matrix $\boldsymbol{A_i}$ for $i = 1, ..., N$:

$$\boldsymbol{A_i} = \begin{cases} 1 & \text{for the } (i,i) \text{ entry} \\ \frac{1}{2} & \text{for the } i\text{-th row and column excluding the diagonal entry} \\ 0 & \text{otherwise;} \end{cases}$$

which allows us to find $\boldsymbol{A_i V}$ for $i = 1, ..., N$:

$$\boldsymbol{A_i V} = \begin{cases} \frac{1}{2}(\sum_{l=1}^{N} V_{lj}) + \frac{1}{2}V_{ij} & \text{for the } i\text{-th row } j\text{-th column for all columns } \boldsymbol{V} \\ \frac{1}{2}V_{ij} & \text{for the non } i\text{-th row } j\text{-th column for all columns of } \boldsymbol{V} \end{cases}$$

which is dense matrix that would take much more time to compute than define. Also

$\boldsymbol{AV}$ is smaller less memory exhaustive than storing $\boldsymbol{A}$ during Algorithm 2.

CASE III: With $\boldsymbol{A_{ij}}$ corresponding to the missing edge constraint: We have the data matrix $\boldsymbol{A_{ij}}$ for $(i,j) \notin E$ s.t. $i \neq j$:

$$\boldsymbol{A_{ij}} = \begin{cases} \frac{1}{2} & \text{for the } (i,j)\text{-th entry} \\ \frac{1}{2} & \text{for the } (j,i)\text{-th entry} \\ 0 & \text{otherwise;} \end{cases}$$

which allows us to find $\boldsymbol{A}_{ij}\boldsymbol{V}$ for $(i,j) \notin E$ s.t. $i \neq j$:

$$\boldsymbol{A_{ij}V} = \begin{cases} \frac{1}{2}V_{jl} & \text{for the } (i,l)\text{-th entry for } l = 1, ..., K \\ \frac{1}{2}V_{il} & \text{for the } (j,l)\text{-th entry for } l = 1, ..., K \\ 0 & \text{otherwise;} \end{cases}$$

which saves use many flops as we only need to focus on a few entrys.

CASE IV: With $\boldsymbol{A} = \boldsymbol{I}$ based on the constraint where the trace is equal to K. We trivially get $\boldsymbol{AV} = \boldsymbol{V}$ since $\boldsymbol{A} = \boldsymbol{I}$.

This has allowed us to exploit sparsity of our problem even further, and we show this by comparing `BM2`, without analytic formulas of $\boldsymbol{AV}$, and `BMHC`, with analytic formulas of $\boldsymbol{AV}$, in Chapter 7 for chosen values of N.

9.4 Finding $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{A}}$'s

This section will be focused on finding $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{A}}$'s required to use the Biconvex Relaxation method of alternating direction in Chapter 6. We have previously found $\boldsymbol{A}$'s in the appendix, which are very similar to these $\hat{\boldsymbol{A}}$'s with one exception. We require $\hat{\boldsymbol{A}} \succeq 0$ or each $\hat{\boldsymbol{A}}$ is positive semidefinite. Let us start by looking at SDP (5.1) with $\boldsymbol{A}$'s:

$$\begin{aligned}
\min \quad & -tr(\boldsymbol{C}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\boldsymbol{A_i}\boldsymbol{X}) \leq 1 \;\; \forall\, i = 1, ...N, \\
& tr(\boldsymbol{A_{ij}}\boldsymbol{X}) = 0, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0,
\end{aligned}$$

and now consider SDP (6.1) we desire to create with $\hat{\boldsymbol{A}}$'s:

$$\begin{aligned}
\min \quad & -tr(\hat{\boldsymbol{C}}\boldsymbol{X}) \\
\text{s.t.} \quad & tr(\hat{\boldsymbol{A}}_i\boldsymbol{X}) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\hat{\boldsymbol{A}}_{\boldsymbol{ij}}\boldsymbol{X}) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{X}) = K, \\
& \boldsymbol{X} \succeq 0.
\end{aligned}$$

As discussed above the $\boldsymbol{C}$, $\hat{\boldsymbol{C}}$, $\boldsymbol{A}$'s and $\hat{\boldsymbol{A}}$'s are data matrices which we strive to make symmetric, and the right hand side of our constraints can be grouped into vector $\boldsymbol{b}$ for ease of notation. One thing of note is that as we change our data matrices from $\boldsymbol{A}$ to $\hat{\boldsymbol{A}}$ we also change our vector $\boldsymbol{b}$.

First, the general thought of this transformation is there is a minimum eigenvalue, with value -ζ, from all data matrices which we need to ensure is greater than or equal to zero. We can assume that $\zeta > 0$ because otherwise our minimum

Algorithm 6: Finding $\hat{\boldsymbol{A}}$'s

Data: Input $\boldsymbol{C}$, $\boldsymbol{A}$'s, and $\boldsymbol{b}$
for $i = 1, 2, \ldots, N,\ ij = 1, 2, \ldots K$
  Find $\zeta = \min\{\text{eigenvalue of all data matrices}\}$
  $\hat{\boldsymbol{C}} = \boldsymbol{C} + \zeta \boldsymbol{I}$
  $\hat{\boldsymbol{A}}_{\boldsymbol{i}} = \boldsymbol{A}_{\boldsymbol{i}} + \zeta \boldsymbol{I}$
  $\hat{b_i} = 1 + \zeta K$
  $\hat{\boldsymbol{A}}_{\boldsymbol{ij}} = \boldsymbol{A}_{\boldsymbol{ij}} + \zeta \boldsymbol{I}$
  $\hat{b_{ij}} = \zeta K$
end
Data: Output $\hat{\boldsymbol{C}}$, $\hat{\boldsymbol{A}}$'s, and $\hat{\boldsymbol{b}}$

eigenvalue is greater than or equal to zero, and our data matrices are already positive semidefinite. If this is true then our shift is not needed. The way to ensure that all data matrices are positive semidefinite is to add $\zeta \boldsymbol{I}$ where $\boldsymbol{I}$ is the identity of same size as our data matrices, N by N. This gives us Algorithm 6 to find $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{A}}$ after we have found $\boldsymbol{C}$ and $\boldsymbol{A}$ as previously discussed. Which gives us the following matrices:

$$\hat{\boldsymbol{C}} = \begin{cases} \zeta - 1 & \text{along the diagonal} \\ -1 & \text{otherwise;} \end{cases}$$

concerning the matrix $\boldsymbol{C}$ in our objective,

$$\hat{\boldsymbol{A}}_{\boldsymbol{i}} = \begin{cases} 1 + \zeta & \text{for the } (i, i) \text{ entry} \\ \zeta & \text{along the diagonal except the } (i, i) \text{ entry} \\ \frac{1}{2} & \text{for the } i\text{-th row and column excluding the diagonal entry} \\ 0 & \text{otherwise;} \end{cases}$$

for matrices $\boldsymbol{A_i}$ concerning the constraints about row sum for $i = 1, 2, \ldots N$,

$$\boldsymbol{A_{ij}} = \begin{cases} \frac{1}{2} & \text{for the } (i,j)\text{-th entry} \\ \zeta & \text{along the diagonal} \\ 0 & \text{otherwise.} \end{cases}$$

for the matrices $\boldsymbol{A_{ij}}$ concerning the missing edges constraint for $ij \notin E$ such that $i \neq j$ in graph, G.

Since we have discussed how to find $\hat{\boldsymbol{C}}$ and $\hat{\boldsymbol{A}}$'s, let us now look at why when we do this it changes the value of our vector $\boldsymbol{b}$. Without loss of generality we will consider one of the $\hat{\boldsymbol{A}}$'s, $\hat{\boldsymbol{A}}_{\boldsymbol{i}} = \boldsymbol{A_i} + \zeta \boldsymbol{I}$ and calculate how this changes our right hand side of the constraint inside the trace inner product.

$$\begin{aligned} tr(\hat{\boldsymbol{A}}_{\boldsymbol{i}}\boldsymbol{X}) &= tr((\boldsymbol{A_i} + \zeta \boldsymbol{I})\boldsymbol{X}) \\ &= tr(\boldsymbol{A_i}\boldsymbol{X}) + tr(\zeta \boldsymbol{I}\boldsymbol{X}) \\ &= tr(\boldsymbol{A_i}\boldsymbol{X}) + \zeta tr(\boldsymbol{I}\boldsymbol{X}) \\ &= b_i + \zeta K, \end{aligned}$$

This shows how for each data matrix $\boldsymbol{A}$ that we change to $\hat{\boldsymbol{A}}$ we need to add ζK to the right hand side of the constraint to account for the shifting of making our matrices positive semidefinite.

9.5 Finding $\boldsymbol{L}$'s

In this section, we will discuss how to find $\boldsymbol{L}$'s such that $\hat{\boldsymbol{A}} = \boldsymbol{L}^T\boldsymbol{L}$ for lower triangular $\boldsymbol{L}$. Recall from the previous section that we found $\hat{\boldsymbol{A}}$ so that it was positive semidefinite to ensure it had the factorization which we now discuss in detail. Since these matrices are positive semidefinite, this factorization is not unique so we will attempt to do this in the most efficient way possible. First, note that we will be using the Cholesky decomposition to do this which can be rather expensive, $\mathcal{O}(\frac{N^3}{3})$ in terms of flop counts for a N by N matrix. Which is why we have formed analytic formulas, or created an algorithm to find $\boldsymbol{L}$ rather than allowing it to be computed during the setup of our K Disjoint Clique problem. Another important note is that the Cholesky decomposition does not normally produce a sparse matrix, but rather a dense matrix. We will use permutations of our matrices to avoid dense matrices when possible.

We will describe how to take SDP (6.2) with $\hat{\boldsymbol{A}}$'s:

$$\begin{aligned}
\min \quad & -tr(\hat{\boldsymbol{C}}\boldsymbol{V}\boldsymbol{V}^T) \\
\text{s.t.} \quad & tr(\hat{\boldsymbol{A}}_{\boldsymbol{i}}\boldsymbol{V}\boldsymbol{V}^T) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\hat{\boldsymbol{A}}_{\boldsymbol{ij}}\boldsymbol{V}\boldsymbol{V}^T) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T) = K,
\end{aligned}$$

and transform it into SDP (6.3) with $\boldsymbol{L}$'s:

$$\begin{aligned}
\min \quad & -tr(\boldsymbol{V}^T\hat{\boldsymbol{C}}\boldsymbol{V}) \\
\text{s.t.} \quad & tr(\boldsymbol{V}^T\boldsymbol{L}_{\boldsymbol{i}}^T\boldsymbol{L}_{\boldsymbol{i}}\boldsymbol{V}) \leq 1 + K\zeta \;\; \forall\, i = 1, ...N, \\
& tr(\boldsymbol{V}^T\boldsymbol{L}_{\boldsymbol{ij}}^T\boldsymbol{L}_{\boldsymbol{ij}}\boldsymbol{V}) = K\zeta, \;\; \forall\, (i,j) \notin E \text{ s.t. } i \neq j, \\
& tr(\boldsymbol{V}^T\boldsymbol{L}^T\boldsymbol{L}\boldsymbol{V}) = K,
\end{aligned}$$

using the general framework discussed above.

First, let us look at the trivial case when $\boldsymbol{A} = \hat{\boldsymbol{A}} = \boldsymbol{I}$ based on the constraint

$tr(\boldsymbol{A}\boldsymbol{V}\boldsymbol{V}^T) = K$. In this case $\boldsymbol{L} = \boldsymbol{L}^T = \boldsymbol{I}$, and note there is no shift of ζ needed in this case to find $\hat{\boldsymbol{A}}$ as $\boldsymbol{A} \succeq 0$.

Now looking at the constraints based on the row sum $tr(\hat{\boldsymbol{A}}_{\boldsymbol{i}}\boldsymbol{V}\boldsymbol{V}^T) \leq 1 + K\zeta \;\; \forall\, i = 1, ..., N$ where we will factor $\hat{\boldsymbol{A}}_{\boldsymbol{i}} = \boldsymbol{L}_{\boldsymbol{i}}^T \boldsymbol{L}_{\boldsymbol{i}}$. We will start with $i = N$ and then use permutations after that to avoid dense matrices. Recall with previous calculations that we have the data matrix $\hat{\boldsymbol{A}}_{\boldsymbol{N}}$:

$$\hat{\boldsymbol{A}}_{\boldsymbol{N}} = \begin{cases} 1 + \zeta & \text{for the } (N, N) \text{ entry} \\ \zeta & \text{along the diagonal excluding the } (N, N) \text{ entry} \\ \frac{1}{2} & \text{for the } N\text{th row and column excluding the diagonal entry} \\ 0 & \text{otherwise.} \end{cases}$$

We will factor this into $\hat{\boldsymbol{A}}_{\boldsymbol{N}} = \boldsymbol{L}_{\boldsymbol{N}}^T \boldsymbol{L}_{\boldsymbol{N}}$ using Cholesky decomposition. One thing to be clear on is we purposefully started with this matrix because the Cholesky decomposition of this matrix is not dense, but instead sparse. We will get the following $\boldsymbol{L}_{\boldsymbol{N}}$:

$$\boldsymbol{L}_{\boldsymbol{N}} = \begin{cases} \sqrt{1 + \zeta - \frac{N-1}{4\zeta}} & \text{for the } (N, N) \text{ entry} \\ \sqrt{\zeta} & \text{along the diagonal excluding the } (N, N) \text{ entry} \\ \frac{1}{2\sqrt{\zeta}} & \text{for the } N\text{th row excluding the diagonal entry} \\ 0 & \text{otherwise,} \end{cases}$$

which one can do the matrix multiplication to check that $\hat{\boldsymbol{A}}_{\boldsymbol{N}} = \boldsymbol{L}_{\boldsymbol{N}}^T \boldsymbol{L}_{\boldsymbol{N}}$. We now find the rest of our $\boldsymbol{L}_i$ for $i = 1, 2, \ldots N - 1$ using a permutation of $\boldsymbol{L}_{\boldsymbol{N}}$ to avoid a dense matrix that would be created by blindly doing Cholesky decomposition to each $\hat{\boldsymbol{A}}_{\boldsymbol{i}}$. To do this start by considering the permutation matrix $\boldsymbol{P}_{\boldsymbol{i} \to \boldsymbol{j}}$ equal to the identity matrix,

$\boldsymbol{I}$ with the i-th row swapped with the j-th row. For example, $\boldsymbol{P_{1\to N}}$ would be:

$$\boldsymbol{P_{1\to N}} = \begin{cases} 1 & \text{for the } (1,N) \text{ entry} \\ 1 & \text{for the } (N,1) \text{ entry} \\ 1 & \text{along the diagonal excluding the } (1,1) \text{ and } (N,N) \text{ entry} \\ 0 & \text{otherwise,} \end{cases}$$

which can be used to find $\hat{\boldsymbol{A}}_{\boldsymbol{1}}$ by switching row 1 and row N of $\hat{\boldsymbol{A}}_{\boldsymbol{N}}$ to get $\hat{\boldsymbol{A}}_{\boldsymbol{1}} = \boldsymbol{P_{1\to N}}\boldsymbol{L_N^T}\boldsymbol{L_N}\boldsymbol{P_{1\to N}^T}$ so that $\boldsymbol{L_1^T} = \boldsymbol{P_{1\to N}}\boldsymbol{L_N^T}$ giving us:

$$\boldsymbol{L_1} = \begin{cases} \sqrt{1+\zeta-\frac{N-1}{4\zeta}} & \text{for the } (1,N) \text{ entry} \\ \sqrt{\zeta} & \text{along the diagonal excluding the } (N,N) \text{ and } (1,1) \text{ entries} \\ \frac{1}{2\sqrt{\zeta}} & \text{for the 1st row excluding the } (1,N) \text{ entry} \\ \sqrt{\zeta} & \text{for the } (N,1) \text{ entry} \\ 0 & \text{otherwise.} \end{cases}$$

Using similar methods of permutations and our $\hat{\boldsymbol{A}}_{\boldsymbol{N}}$ one can find $\boldsymbol{L_i}$ for each $i = 1, \ldots N$ to have the following form:

$$\boldsymbol{L_i} = \begin{cases} \sqrt{1+\zeta-\frac{N-1}{4\zeta}} & \text{for the } (i,N) \text{ entry} \\ \sqrt{\zeta} & \text{along the diagonal excluding the } (N,N) \text{ and } (i,i) \text{ entries} \\ \frac{1}{2\sqrt{\zeta}} & \text{for the } i\text{th row excluding the } (i,N) \text{ entry} \\ \sqrt{\zeta} & \text{for the } (N,i) \text{ entry} \\ 0 & \text{otherwise.} \end{cases}$$

Now we will consider finding $\boldsymbol{L_{ij}}$, such that $\hat{\boldsymbol{A}}_{\boldsymbol{ij}} = \boldsymbol{L_{ij}^T}\boldsymbol{L_{ij}}$, for the constraint

concerning the missing edges from graph G. The number of $\boldsymbol{L_{ij}}$'s will be equal to the number of zeros above the diagonal in graph G. We will again use a permutation of the most sparse matrix $\boldsymbol{L_{ij}}$. To do this first consider a slightly different permutation matrix $\boldsymbol{P_{ij\to N-1N}}$:

$$\boldsymbol{P_{ij\to N-1N}} = \begin{cases} 1 & \text{for the } (N,i) \text{ entry} \\ 1 & \text{for the } (N-1,j) \text{ entry} \\ 1 & \text{for the } (i,N) \text{ entry} \\ 1 & \text{for the } (j,N-1) \text{ entry} \\ 1 & \text{along the diagonal excluding the } (i,i)\ ,\ (j,j)\ (N-1,N-1) \text{ and } (N,N) \text{ entries} \\ 0 & \text{otherwise;} \end{cases}$$

this permutation matrix takes the i-th row to the N-th row, the j-th row to the $N-1$-th row, the $N-1$-th row to the j-th row and the N-th to the i-th row and is derived using the knowledge that $\boldsymbol{L_{N-1N}}$ will be the most sparse $\boldsymbol{L_{ij}}$ of our Cholesky decomposition. First let us find $\boldsymbol{L_{N-1N}}$ so we can use the permutation matrix just discussed and $\boldsymbol{L_{N-1N}}$ to find all $\boldsymbol{L_{ij}}$. Using Cholesky decomposition we can find $\boldsymbol{L_{N-1N}}$:

$$\boldsymbol{L_{N-1N}} = \begin{cases} \sqrt{\zeta - \frac{1}{4\zeta}} & \text{for the } (N,N) \text{ entry} \\ \sqrt{\zeta} & \text{along the diagonal excluding the } (N,N) \text{ entry} \\ \frac{1}{2\sqrt{\zeta}} & \text{for the } (N,N-1) \text{ entry} \\ 0 & \text{otherwise,} \end{cases}$$

from here we can use this and our permutation matrices to find the rest of our $\boldsymbol{L_{ij}}$. We

will now find a general form for $\boldsymbol{L_{ij}}$:

$$\boldsymbol{L_{ij}} = \begin{cases} \sqrt{\zeta - \frac{1}{4\zeta}} & \text{for the } (i, N) \text{ entry} \\ \sqrt{\zeta} & \text{along the diagonal excluding the } (i,i),\ (j,j),\ (N-1, N-1),\ \text{and } (N,N) \text{ entries} \\ \frac{1}{2\sqrt{\zeta}} & \text{for the } (i, N-1) \text{ entry} \\ \sqrt{\zeta} & \text{for the } (j, N-1) \text{ entry} \\ \sqrt{\zeta} & \text{for the } (N-1, j) \text{ entry} \\ \sqrt{\zeta} & \text{for the } (N, i) \text{ entry} \\ 0 & \text{otherwise,} \end{cases}$$

which is more sparse than just using Cholesky decomposition on each $\hat{\boldsymbol{A}}_{\boldsymbol{ij}}$, and comes from the following equality: $\hat{\boldsymbol{A}}_{\boldsymbol{ij}} = \boldsymbol{L}_{\boldsymbol{ij}}^T \boldsymbol{L}_{\boldsymbol{ij}} = \boldsymbol{P}_{\boldsymbol{ij \to N-1N}} \boldsymbol{L}_{\boldsymbol{N-1N}}^T \boldsymbol{L}_{\boldsymbol{N-1N}} \boldsymbol{P}_{\boldsymbol{ij \to N-1N}}^T$.

We have now found formulas, which can be converted into algorithms, for finding each $\boldsymbol{L}$ such that $\hat{\boldsymbol{A}} = \boldsymbol{L}^T \boldsymbol{L}$. This work decreases the overall flop count in Algorithm 3, and helps to maximize efficiency with the goal of making our algorithm comparable to current Alternating Direction Method of Multiplier methods detailed in Chapter 4. One should also note by keeping the matrices sparse that any future multiplication by $\boldsymbol{L}$, for example $\boldsymbol{LV}$ needed in gradient descent to find $\boldsymbol{V}$ in BCR, would be much more manageable, or even find analytic formulas to decrease flops more.

9.6 Extra Figures

9.6.1 Recovery

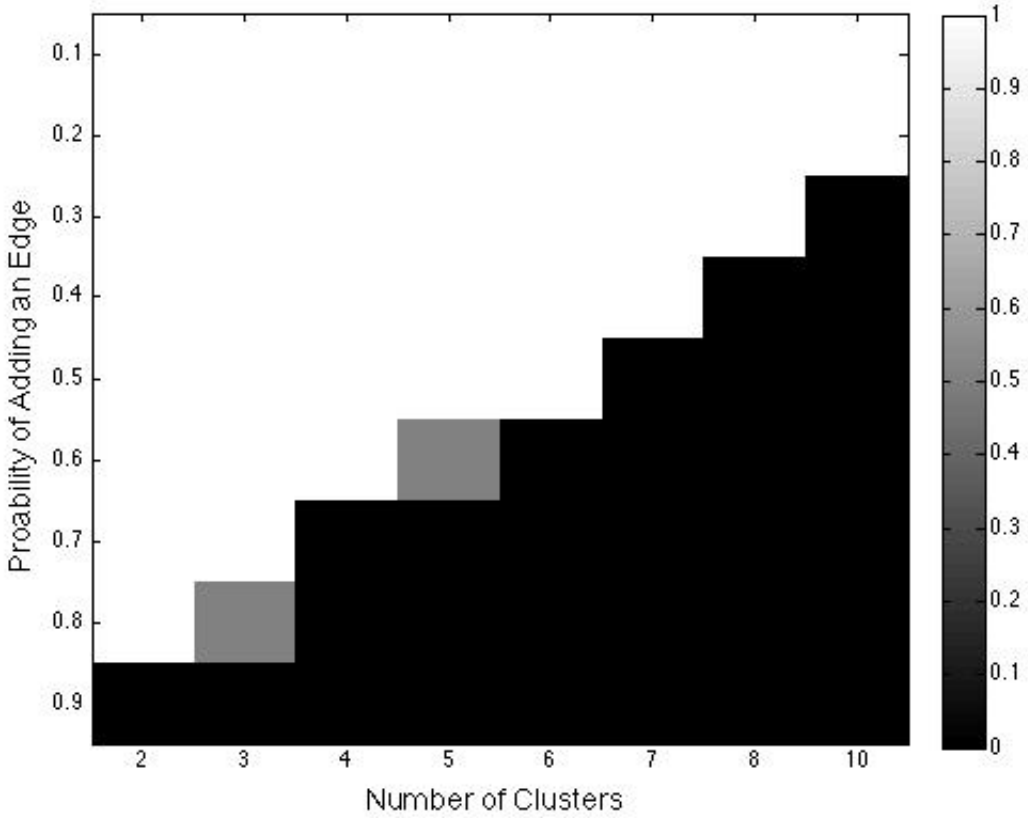

Figure 9.1: Recovery `ADMM` with $N = 50$

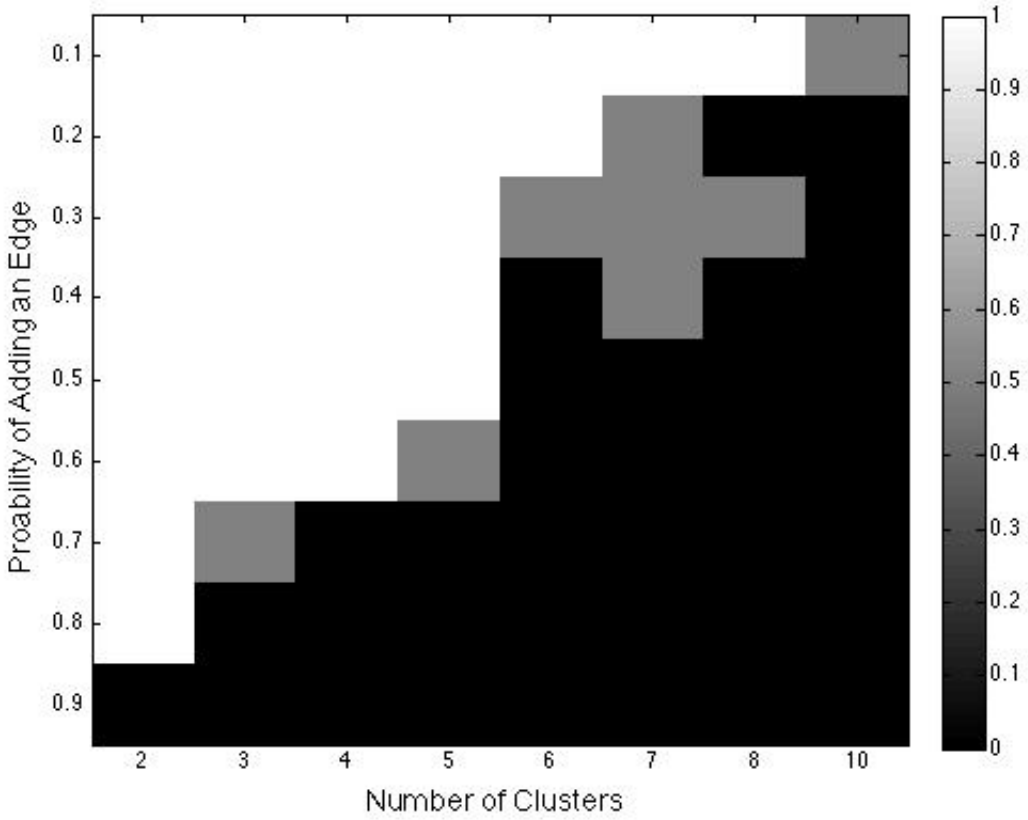

Figure 9.2: Recovery `BMHC` with $N = 50$

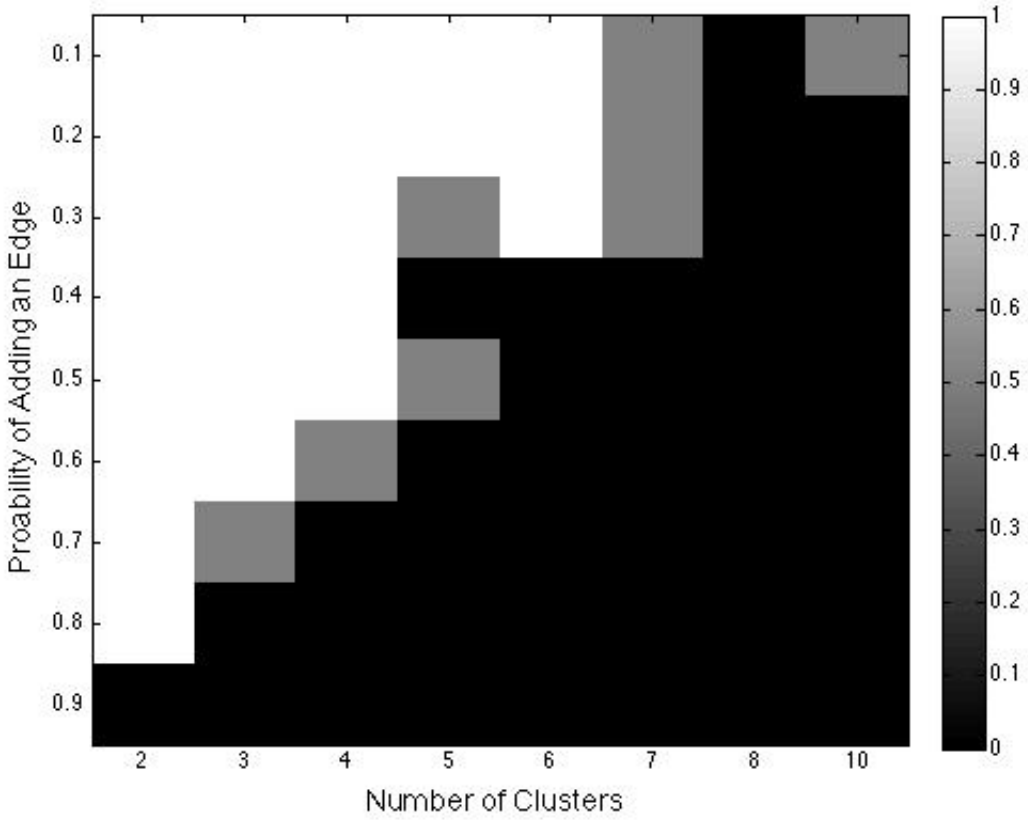

Figure 9.3: Recovery BM2 with $N = 50$

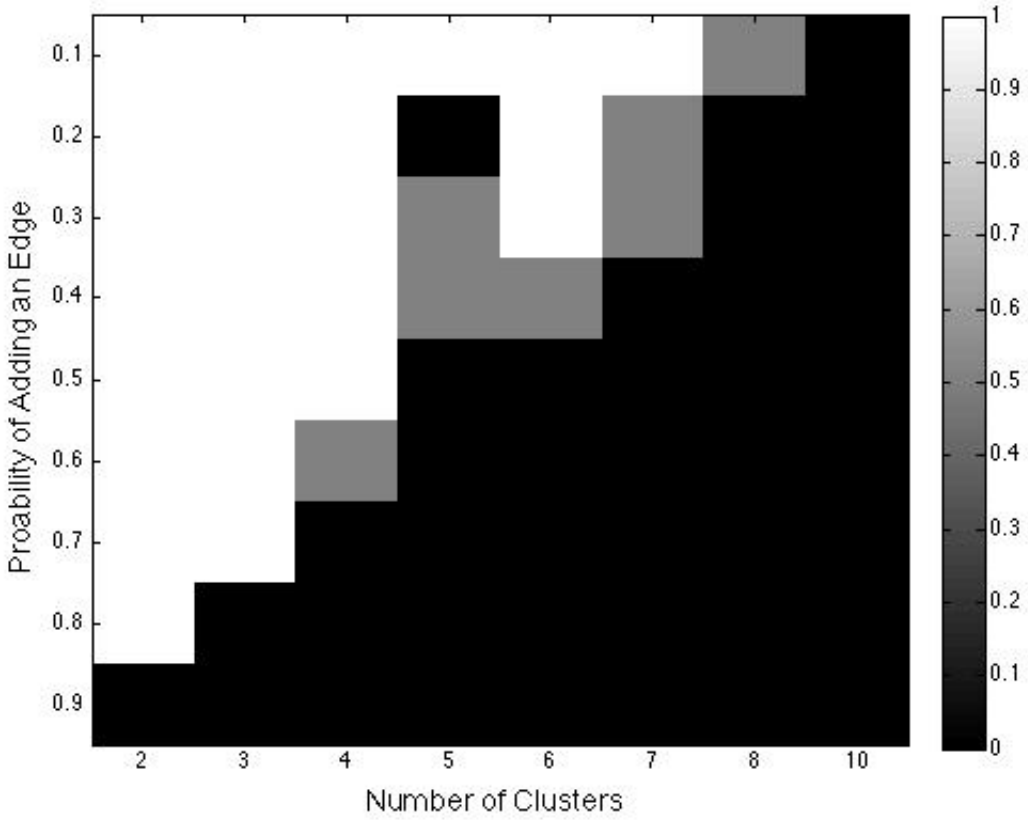

Figure 9.4: Recovery `BCRMF` with $N = 50$

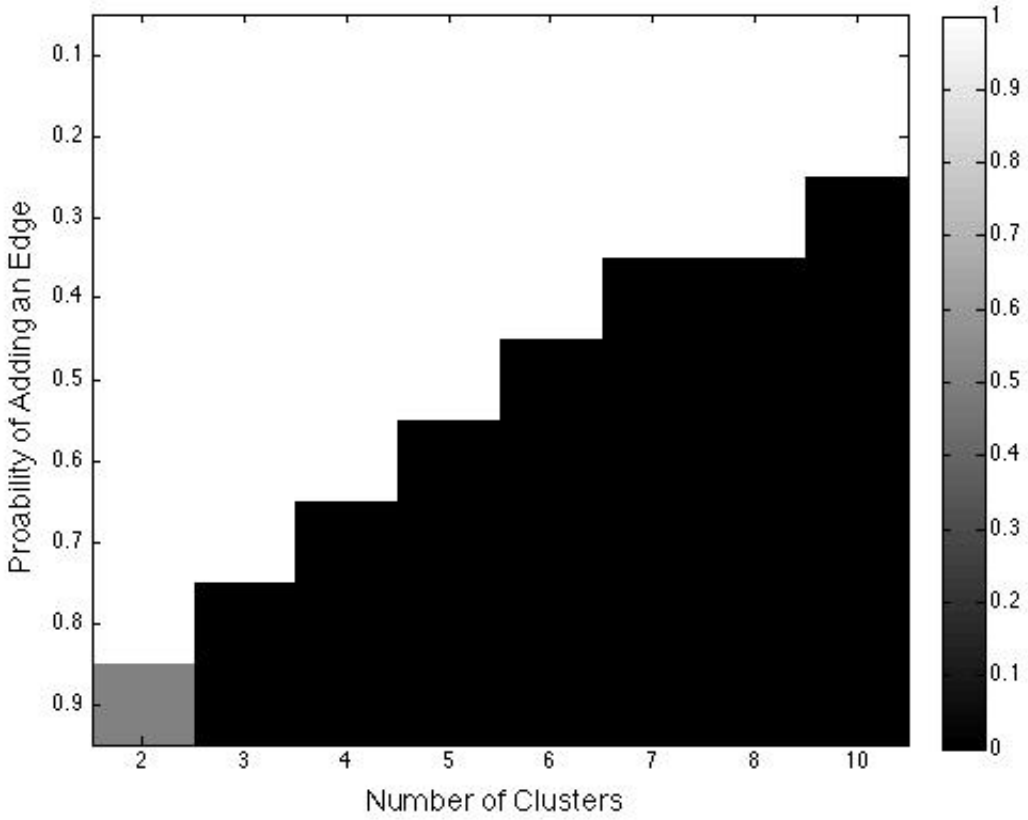

Figure 9.5: Recovery `CVX` with $N = 50$

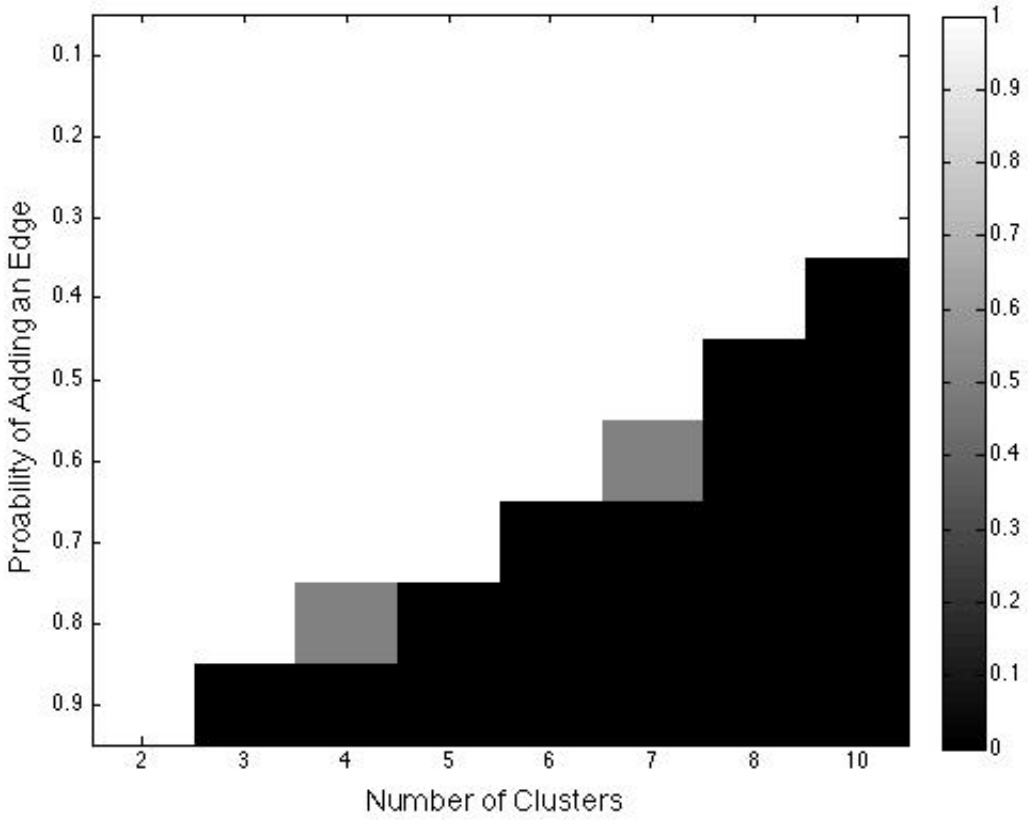

Figure 9.6: Recovery `ADMM` with $N = 80$

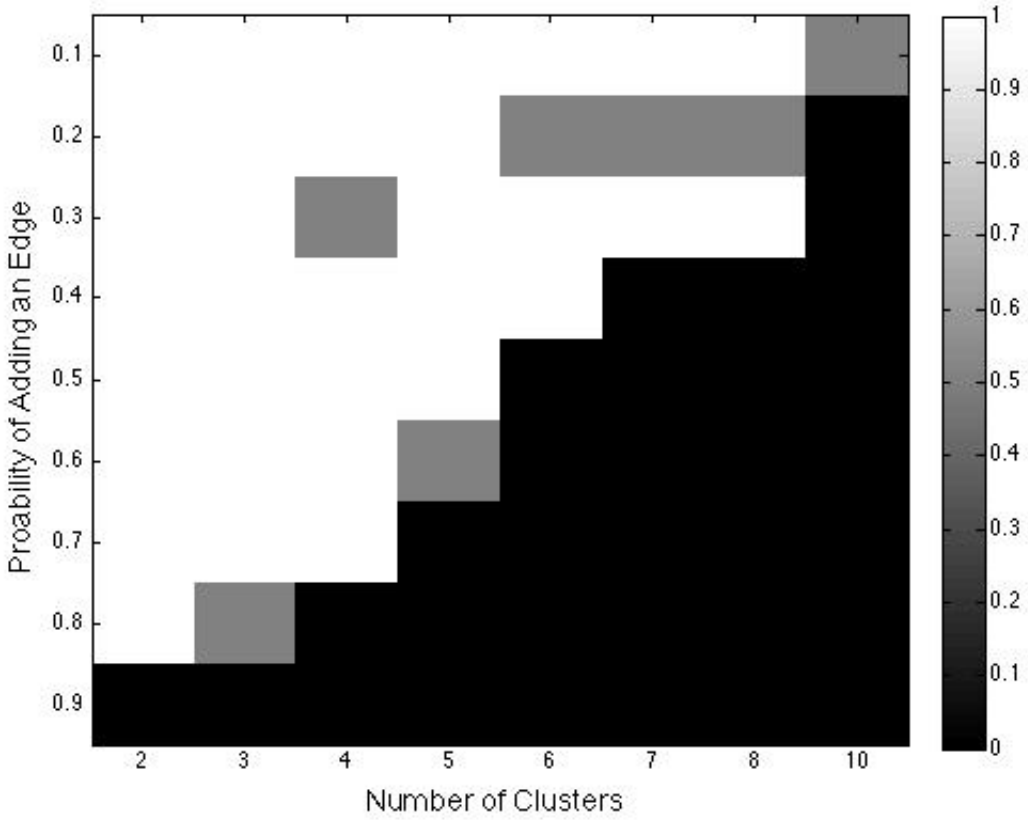

Figure 9.7: Recovery `BMHC` with $N = 80$

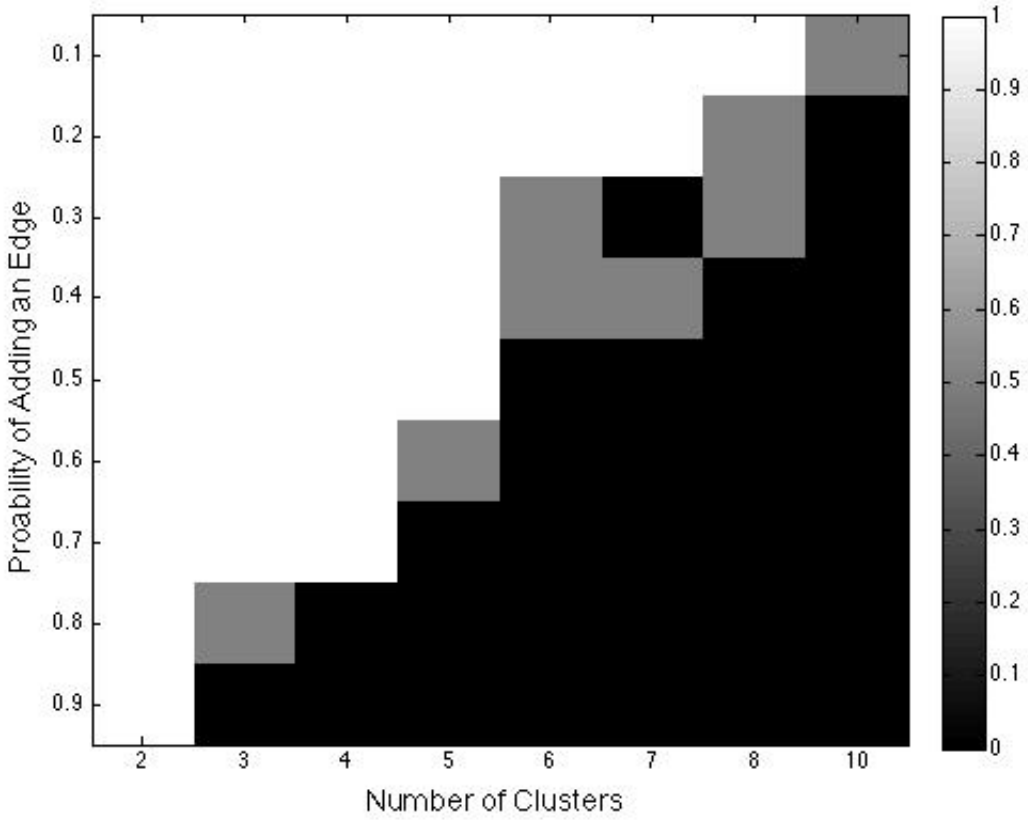

Figure 9.8: Recovery BM2 with $N = 80$

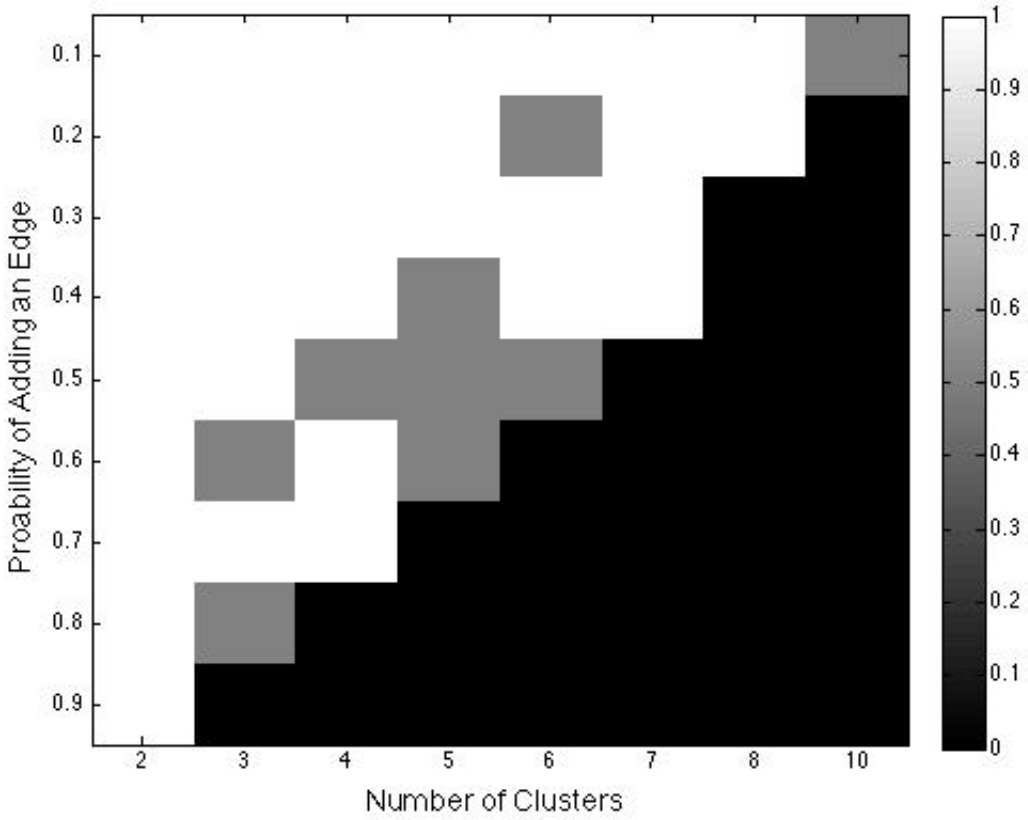

Figure 9.9: Recovery `BCRMF` with $N = 80$

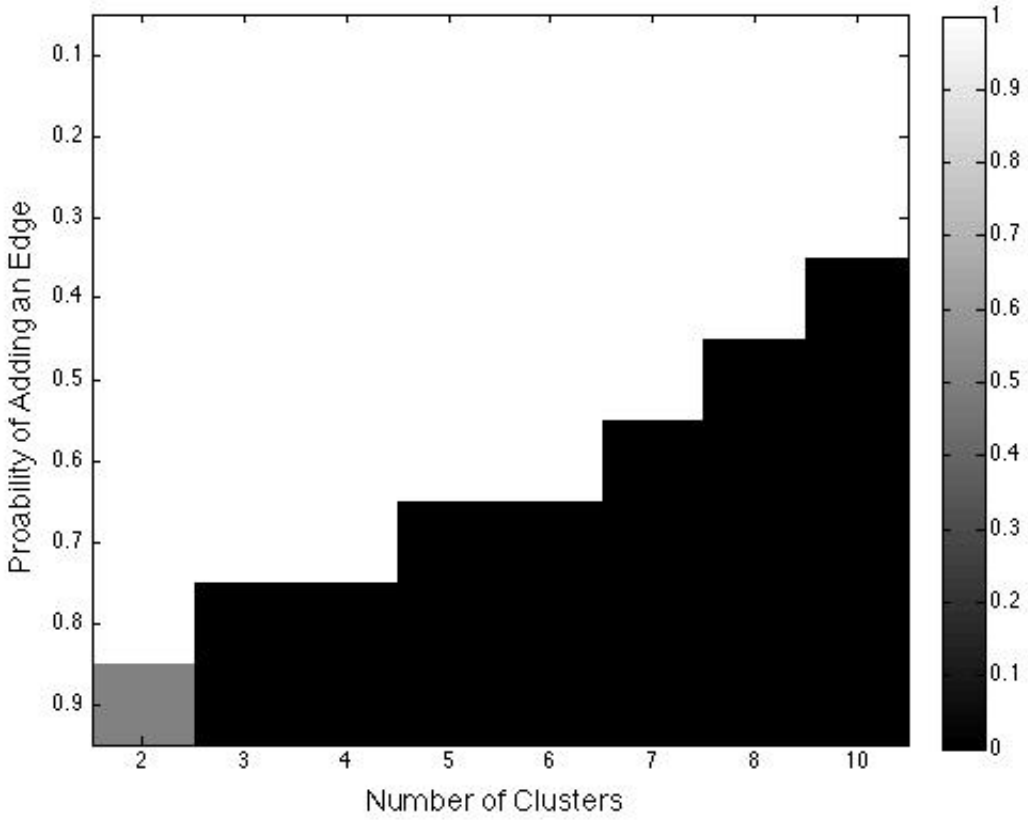

Figure 9.10: Recovery `CVX` with $N = 80$

9.6.2 Unrounded Results

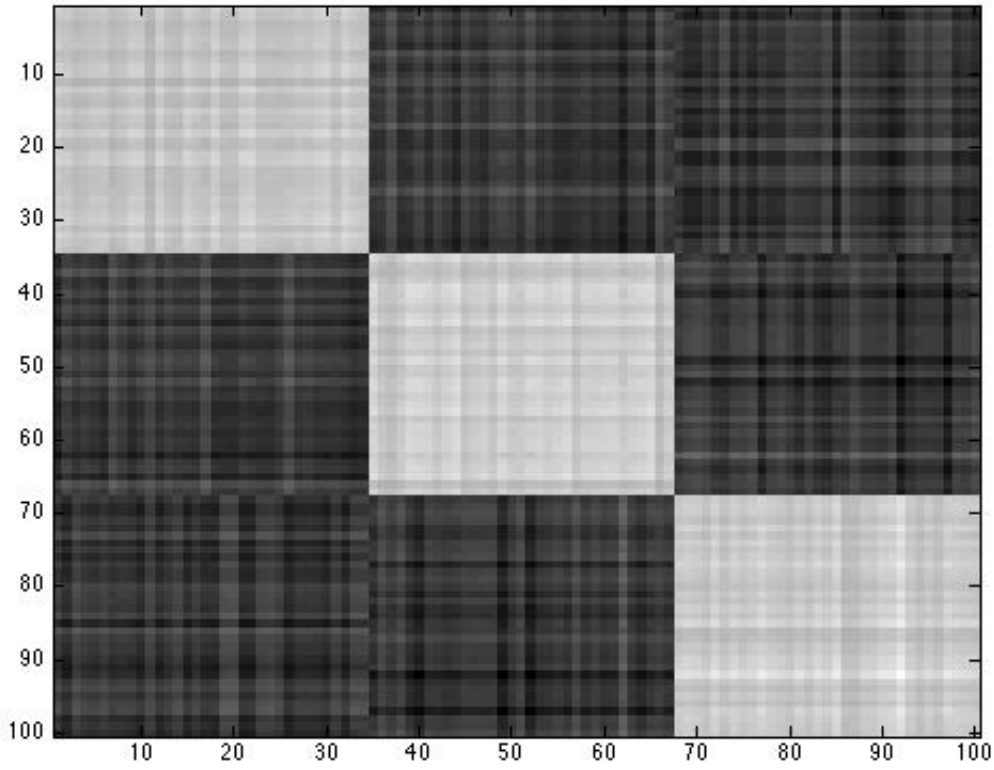

Figure 9.11: Unrounded Approximate Solution with $K = 3$ and $prob = 0.2$

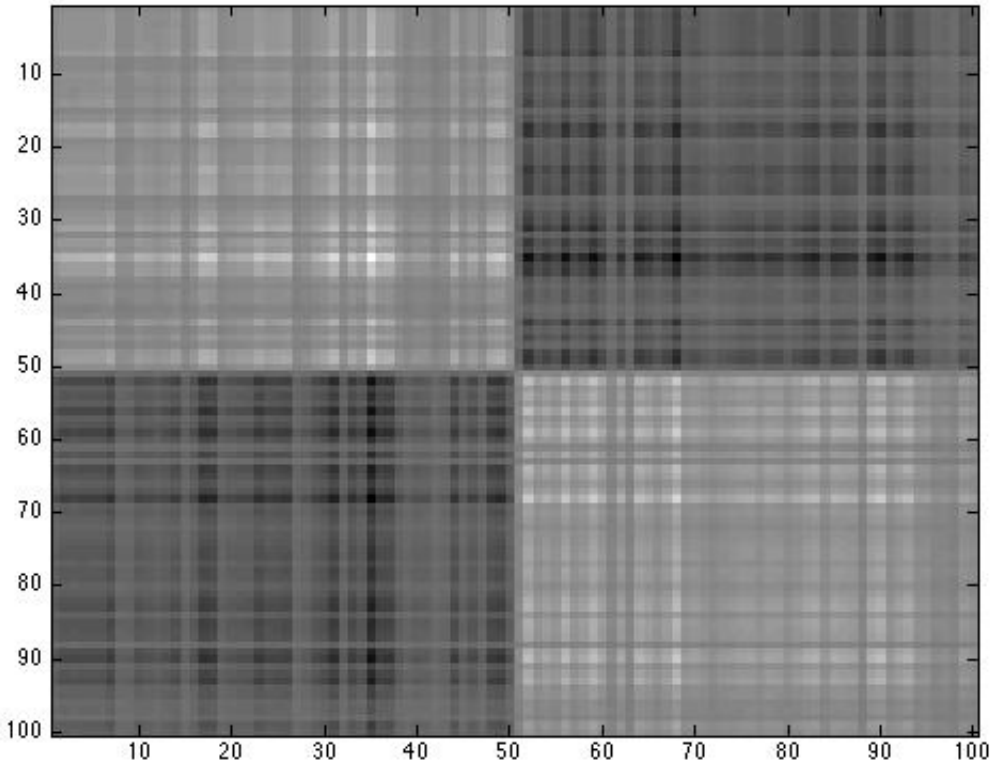

Figure 9.12: Unrounded Approximate Solution with $K = 2$ and $prob = 0.8$

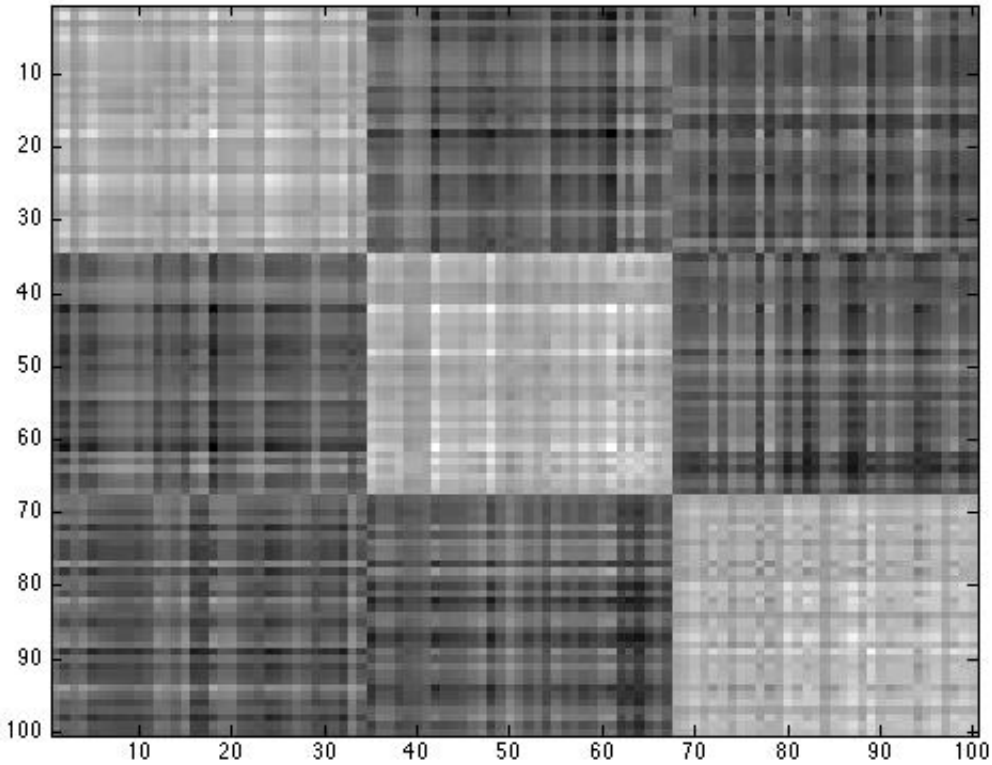

Figure 9.13: Unrounded Approximate Solution with $K = 3$ and $prob = 0.7$

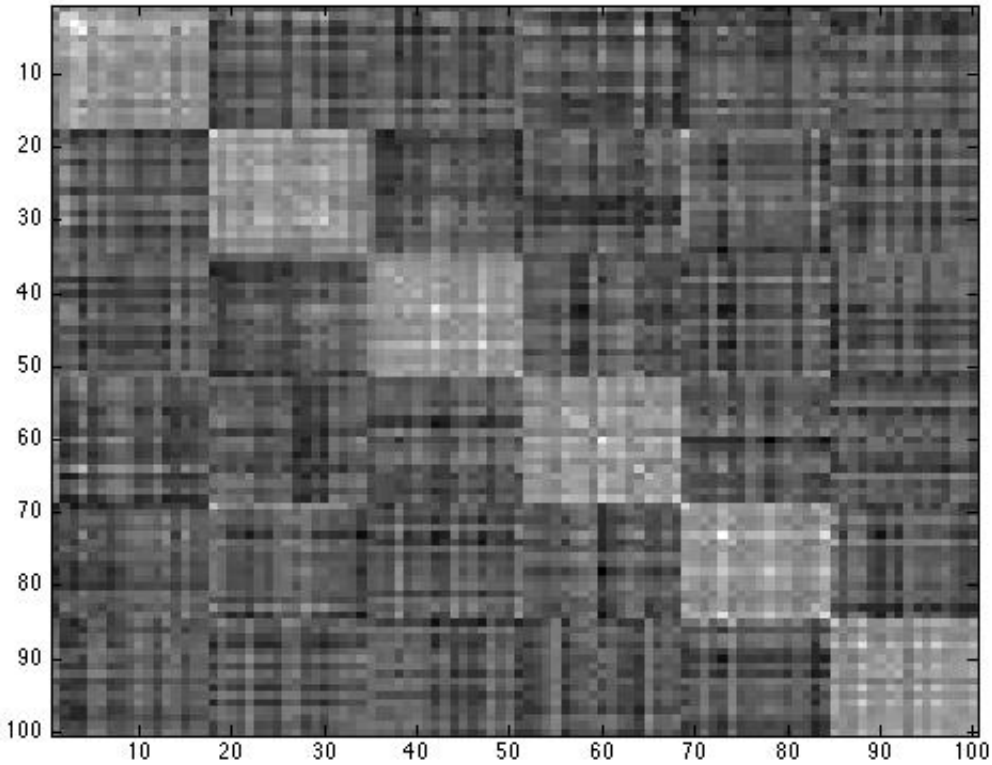

Figure 9.14: Unrounded Approximate Solution with $K = 6$ and $prob = 0.5$

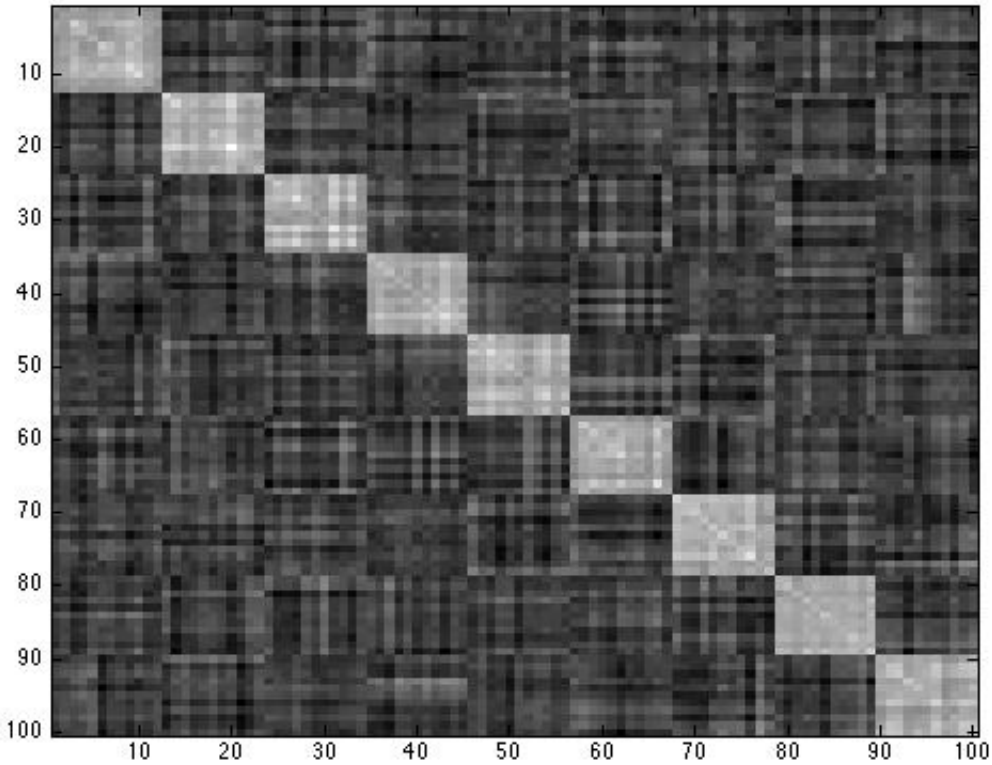

Figure 9.15: Unrounded Approximate Solution with $K = 9$ and $prob = 0.2$

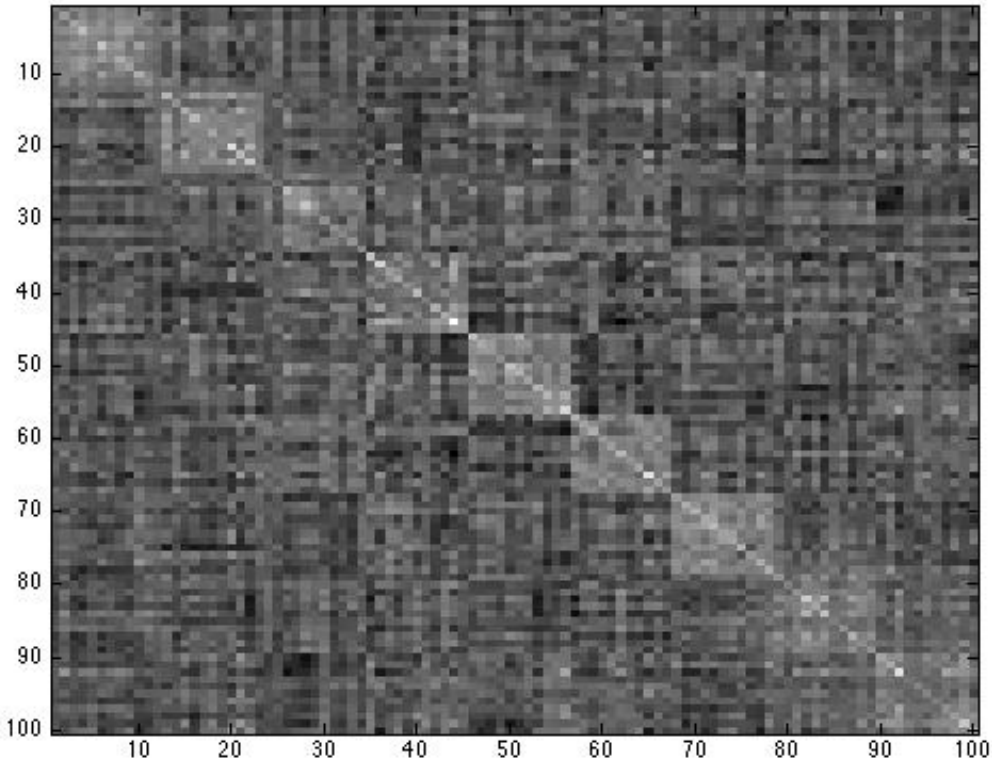

Figure 9.16: Unrounded Approximate Solution with $K = 9$ and $prob = 0.5$

9.6.3 Rounded Results

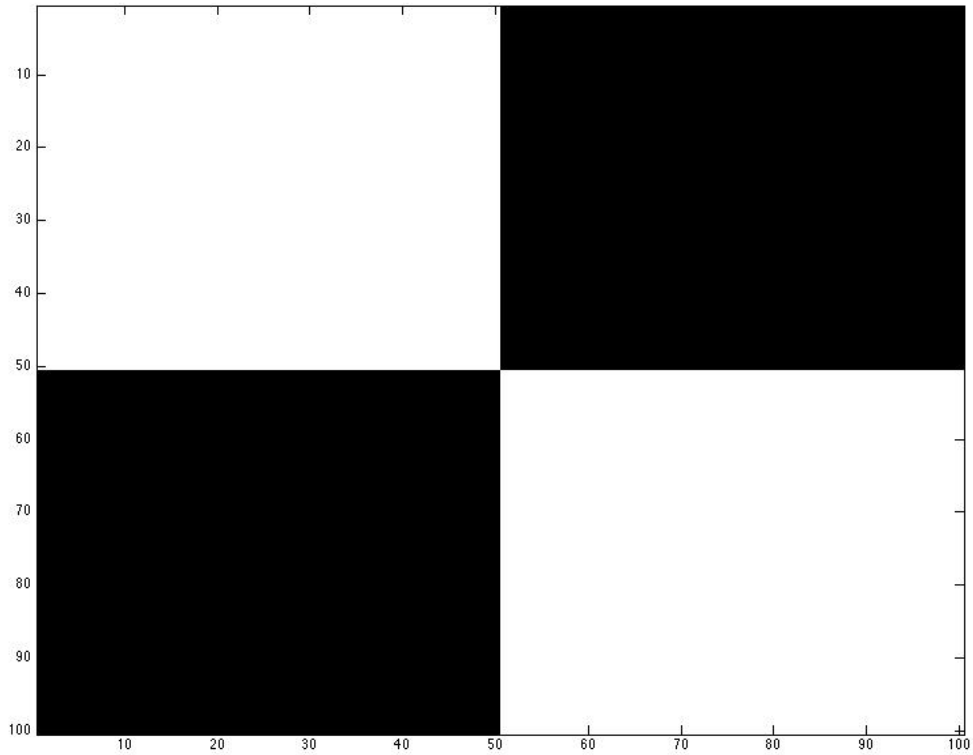

Figure 9.17: Rounded Approximate Solution from `ADMM` with $K = 2$ and $prob = 0.9$

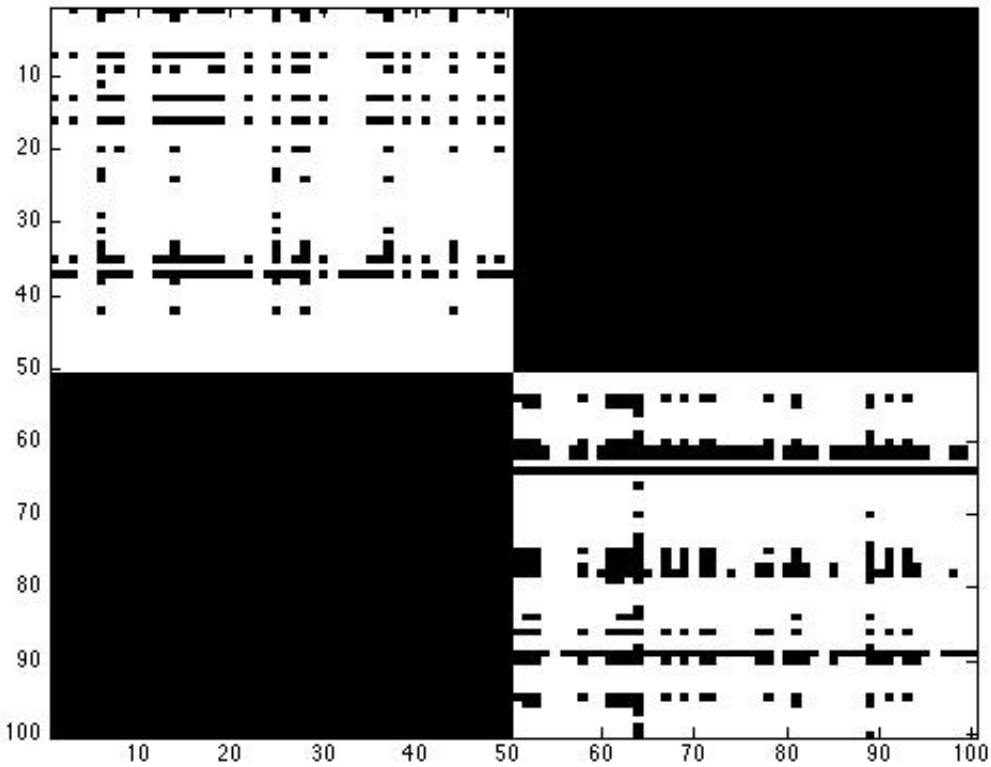

Figure 9.18: Rounded Approximate Solution from `BMHC` with $K = 2$ and $prob = 0.9$

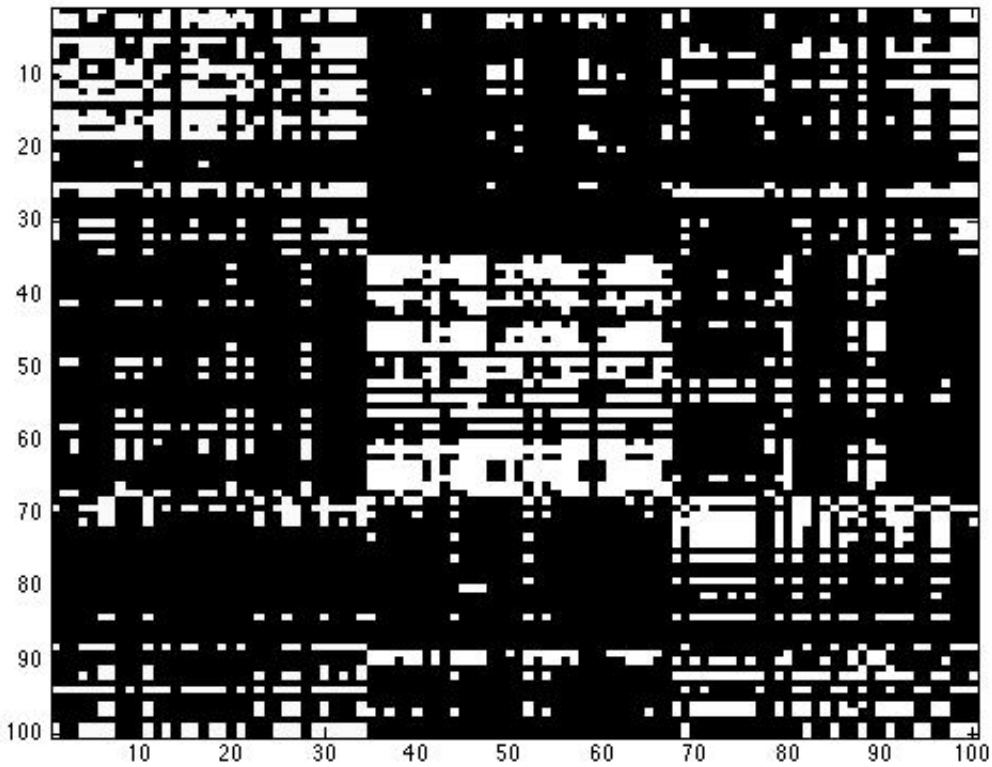

Figure 9.19: Rounded Approximate Solution with $K = 3$ and $prob = 0.9$

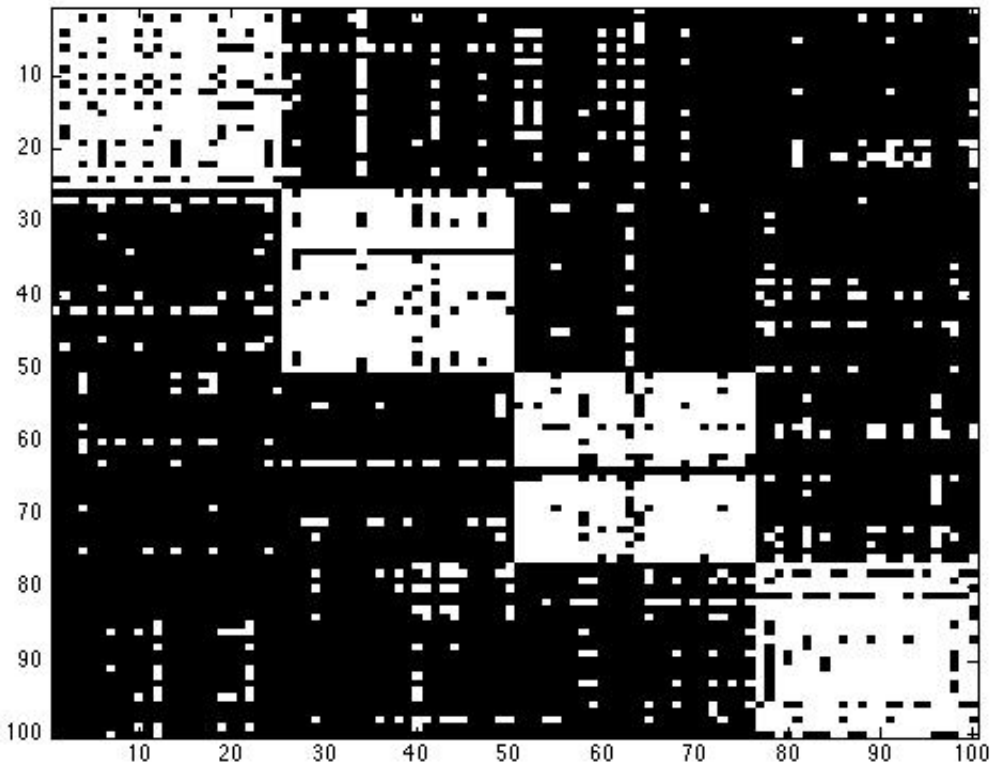

Figure 9.20: Rounded Approximate Solution with $K = 4$ and $prob = 0.6$

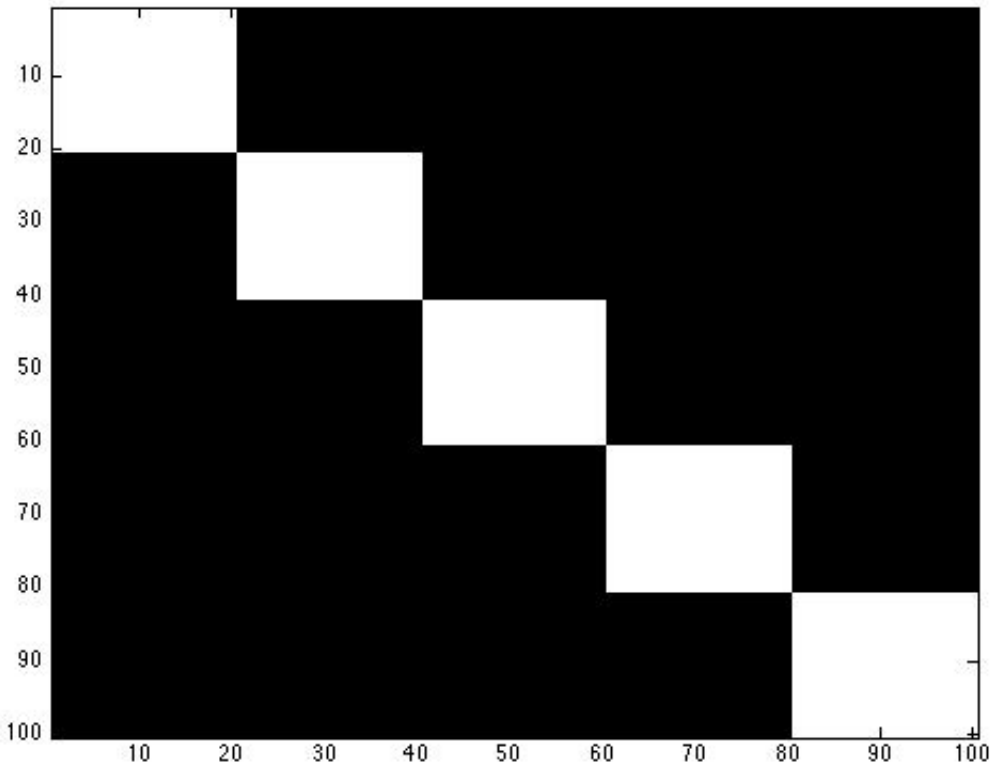

Figure 9.21: Rounded Approximate Solution from `ADMM` with $K = 5$ and $prob = 0.5$

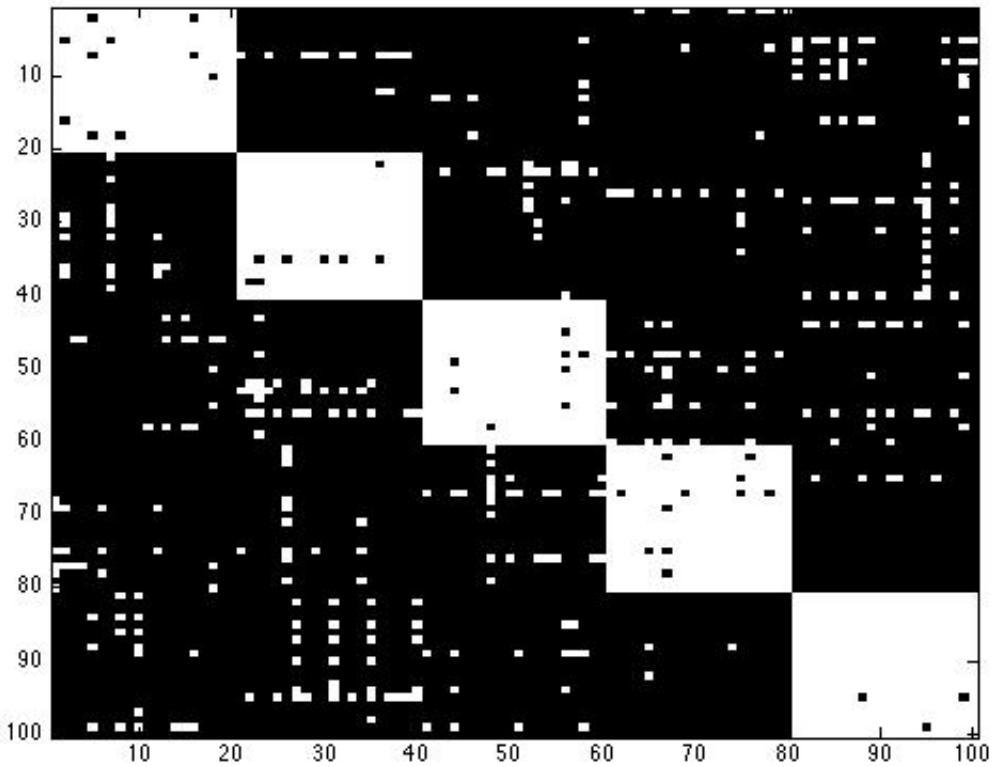

Figure 9.22: Rounded Approximate Solution from `BMHC` with $K = 5$ and $prob = 0.5$

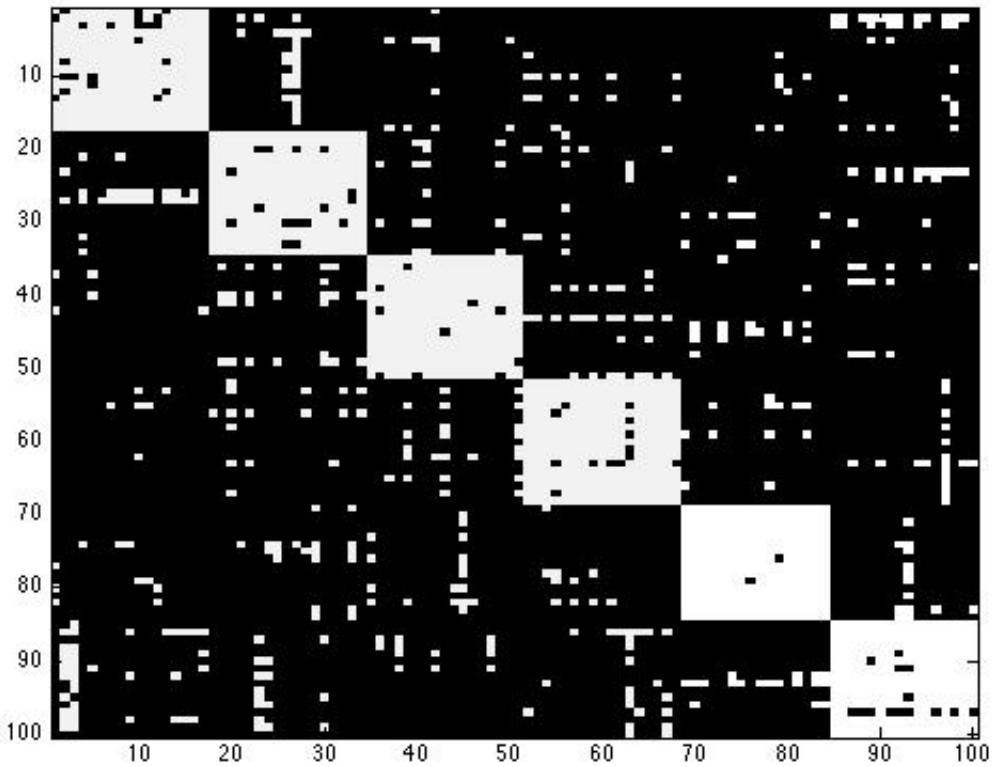

Figure 9.23: Rounded Approximate Solution with $K = 6$ and $prob = 0.6$

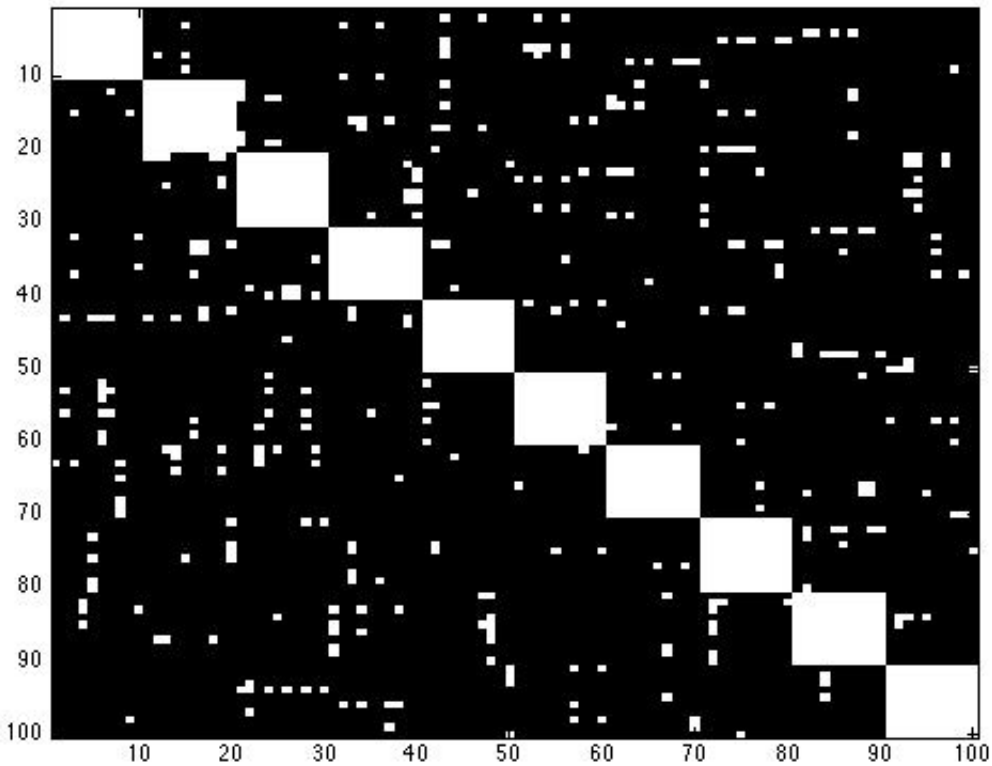

Figure 9.24: Rounded Approximate Solution with $K = 10$ and $prob = 0.6$