A Branch-and-Cut Algorithm based on Semidefinite Programming for the Minimum k-Partition Problem

by

Bissan Ghaddar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Management Sciences

Waterloo, Ontario, Canada, 2007

Abstract

The minimum k-partition (MkP) problem is a well-known optimization problem encountered in various applications most notably in telecommunication and physics. Formulated in the early 1990s by Chopra and Rao, the MkP problem is the problem of partitioning the set of vertices of a graph into k disjoint subsets so as to minimize the total weight of the edges joining vertices in different partitions.

In this thesis, we design and implement a branch-and-cut algorithm based on semidefinite programming (SBC) for the MkP problem. We describe and study the properties of two relaxations of the MkP problem, the linear programming and the semidefinite programming relaxations. We then derive a new strengthened relaxation based on semidefinite programming. This new relaxation provides tighter bounds compared to the other two discussed relaxations but suffers in term of computational time. We further devise an iterative clustering heuristic (ICH), a novel heuristic that finds feasible solution to the MkP problem and we compare it to the hyperplane rounding techniques of Goemans and Williamson and Frieze and Jerrum for k=2 and for k=3 respectively. Our computational results support the conclusion that ICH provides a better feasible solution for the MkP. Furthermore, unlike the hyperplane rounding, ICH remains very effective in the presence of negative edge weights. Next we describe in detail the design and implementation of a branch-and-cut algorithm based on semidefinite programming (SBC) to find optimal solution for the MkP problem. The ICH heuristic is used in our SBC algorithm to provide feasible solutions at each node of the branch-and-cut tree. Finally, we present computational results for the SBC algorithm on several classes of test instances with k=3, 5, and 7. Complete graphs with up to 60 vertices and sparse graphs with up to 100 vertices arising from a physics application were considered.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Professor Miguel Anjos. This thesis would not have been possible without his encouragement, his enthusiasm, and his guidance. It was a great pleasure to me to conduct this thesis under his supervision.

I would like to thank my readers, Professor Anthony Vannelli and Professor Samir Elhedhli, for their helpful feedback and comments. I thank Frauke Liers for her interest in this research and for her insightful remarks and suggestions. I would like to thank Michael Jünger and Frauke Liers for giving me the opportunity to visit University of Cologne for a three month research position to pursue further research on this topic. I would like also to thank Brian Borchers for providing technical assistance with the CSDP solver and Franz Rendl for his comments and suggestions. Thanks also goes to Johannes Hatzl for his valuable remarks and discussions on this subject. I am also grateful to the Department of Management Science, University of Waterloo for assisting me in many different ways and for their financial support. The financial support of NSERC Discovery Grant 312125 and of the MITACS Network of Centres of Excellence through the project *High Performance Optimization: Theory, Algorithm Design and Engineering Applications* was very appreciated.

I wish to thank my beloved Joe Naoum-Sawaya for making this experience enjoyable and productive. I am grateful for his love, support, and motivation. I am also thankful for his valuable suggestions in implementing the branch-and-cut algorithm.

I would like also to acknowledge my friends, especially my best friend Loulou El-Azar for all the emotional support, entertainment, and caring she provided.

Lastly and most importantly, I am indebted to my parents Taha and Fatima Ghaddar and my two sisters Ninar and Ruha Ghaddar for providing me with a loving and supportive environment. It is to them that I dedicate this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Minimum k-Partition Problem

The minimum k-partition problem (MkP) is the problem of partitioning the set of vertices of a graph into k disjoint subsets so as to minimize the total weight of the edges joining vertices in different partitions. This problem is known to be $\mathcal{NP}$-hard in general and hence very difficult to solve. The MkP is equivalent to finding a maximum k-cut, where the weighted sum of all edges with their endpoints in distinct sets is maximized. Several authors, including Barahona and Mahjoub [5] and Deza and Laurent [12], studied the problem of partitioning a graph into two subsets, the special case with k=2 known as the max-cut problem. The maximum k-cut problem has received more attention in the literature than the minimum k-partition problem, such as in Deza, Grötschel, and Laurent [10], Chopra and Rao [9], and the book by Deza and Laurent [12]. Results on the approximation of the maximum k-cut problem were obtained by Karger, Motwani, and Sudan [25] as well as by Frieze and Jerrum [18]. In [18], a polynomial-time approximation algorithm using the semidefinite relaxation with a performance guarantee is presented for the maximum k-partition problem. However, the optimal cut value in the maximum k-partition is underestimated so that the value of the minimum k-partition is overestimated, hence no lower bound is obtained that way. The minimum k-partition problem is formulated by Deza et al. [10, 11] and Chopra and Rao in [8] where several valid and facet-defining inequalities are identified.

The minimum k-partition has applications in network planning [16], VLSI layout design [3], micro-aggregation of statistical data [13], sports team scheduling [36], physics [23, 34, 17], and other areas. We briefly describe two of those applications in the next section.

1.2 Applications of MkP

1.2.1 Physics: Spin Glass Problem

A spin glass is a set of magnetic spins that possesses various interactions between them [34, 17]. Each spin can be in one of a finite number of orientations. Spins i and j are coupled with coupling strength J_{ij}, where the couplings are assumed to follow a Gaussian distribution or to take on values $\{\pm J\}$ (for a given value J) in equal (or nearly equal) numbers.

The special case in which the spin is one-dimensional and can take only one of two orientations, represented by +1 and -1, is called the Ising model. The interactions between the spins describe how the orientation of a given spin and those of its neighboring spins affect the overall energy of the spin glass. The ground state, or minimum-energy state, of a spin glass occurs when the orientations of the spins are chosen so as to minimize the overall energy of the spin glass, that is, to minimize the Hamiltonian representing the total energy of the system.

To formulate the optimization problem, suppose that we have n spins and let v_i be the orientation of spin i, where $v_i = 1$ if the spin is oriented upwards and $v_i = -1$ if it is oriented downwards. Let v_0 indicate the orientation of the exterior magnetic field of strength h. Then the total energy of the system is given by the Hamiltonian [30]

$$H := -\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} J_{ij}v_iv_j - h\sum_{j=1}^{n} v_0v_j.$$

This problem can be represented using an edge-weighted graph $G = (V, E)$, where the vertex set $V = \{1, \ldots, n\}$ is the set of spins, the edge set E describes the pairwise interactions between spins, and the edge weight $w_{ij} = -J_{ij}$ and $w_{0j} = -h$. In the case of the Ising model $w_{ij} \in \{-1, 0, 1\}$ where

$$w_{ij} = \begin{cases} 1, & \text{if } i \text{ and } j \text{ interact positively} \\ -1, & \text{if } i \text{ and } j \text{ interact negatively} \\ 0, & \text{if } i \text{ and } j \text{ have negligible or no interaction.} \end{cases}$$

Let the variable $z_i \in \{1, -1\}$ represent the orientation of spin i, then to find the ground state of the Ising spin glass we want to minimize H. Since

$$\begin{aligned} H &= \sum_{ij \in E,\, i<j} w_{ij} z_i z_j \\ &= \sum_{ij \in E,\, i<j:\, z_i = z_j} w_{ij} z_i z_j + \sum_{ij \in E,\, i<j:\, z_i \neq z_j} w_{ij} z_i z_j \\ &= \sum_{ij \in E,\, i<j} w_{ij} - 2 \sum_{ij \in E,\, i<j:\, z_i \neq z_j} w_{ij}, \end{aligned}$$

minimizing H is equivalent to maximizing $-H$, and since $\sum_{ij \in E,\, i<j} w_{ij}$ is a constant, then maximizing $\sum_{ij \in E,\, i<j:\, z_i \neq z_j} w_{ij}$ is equivalent to maximizing $\sum_{ij \in E,\, i<j} w_{ij} \frac{1 - z_i z_j}{2}$. Therefore, the optimization problem is as follows:

$$\begin{aligned} \max \quad & \frac{1}{2} \sum_{ij \in E,\, i<j} w_{ij}(1 - z_i z_j) \\ \text{s.t.} \quad & |z_i| = 1. \end{aligned}$$

This optimization problem is an instance of the max-cut problem, the minimum $k-$partition problem with $k = 2$. Hence, the problem of determining a ground state of an Ising spin glass is equivalent to the max-cut problem. The spin-glass server website [24] can solve these types of instances using a linear programming-based branch-and-cut algorithm [31].

The spin glass model can be extended to the case where the spins can take more than 2 positions. This is known as the Potts model. The Potts model with p states is a system of spins where each spin i can be in one of the p different states, $\{1, 2, \ldots, p\}$. Let the variable z_{ij} be one if neighbouring spins i and j are in the same state, and zero if they are in different states. The Hamiltonian is defined as follows [29]:

$$H = -\sum_{ij\in E,\, i<j} J_{ij} z_{ij}.$$

Let x_i be a unit vector which takes one of p values. Then x_i satisfies:

$$x_i \cdot x_j = \frac{pz_{ij} - 1}{p - 1}$$

Thus the Hamiltonian becomes:

$$H = -\sum_{ij\in E,\, i<j} J_{ij} \left[\frac{p-1}{p} x_i \cdot x_j + \frac{1}{p}\right].$$

Let $w_{ij} = -J_{ij}$ and let $X_{ij} = x_i \cdot x_j$, and since we want to minimize the Hamiltonian, then the objective function of the optimization problem is as follows:

$$\min \sum_{ij\in E,\, i<j} w_{ij} \frac{(p-1)X_{ij} + 1}{p}.$$

This is equivalent to the minimum k-partition problem as discussed in Section 2.3.3.

1.2.2 Fixed-Spectrum Frequency Assignment Problem

The radio spectrum is a limited resource to be shared by all users. The Global System for Mobile Communications (GSM) divides up the bandwidth among the users in such away that minimizes the interference among users.

Given a list of transmitter/receiver units (TRX), a range of channels, a list of locally blocked channels for each TRX, as well as the minimum separation, the co-channel interference, and the adjacent channel interference matrices, the frequency assignment problem (FAP) is to assign to each TRX one channel from the spectrum which is not locally blocked such that the sum over all interferences occurring between pairs of TRXs is minimized, and all separation requirements are met.

In the fixed-spectrum frequency assignment problem (FS-FAP), the frequencies are assigned from a limited number of available frequencies in such a way that the interference is minimized. The FS-FAP can be represented using an edge-weighted undirected graph. The representation of the problem is based on a 5-tuple $\{V, E, D, P, B\}$ where:

- V is the vertex set of the undirected graph. Every vertex represents a transmitter.
- E is the set of edges of the undirected graph. Edges represent those transmitters that are constrained, i.e., pairs of potentially interfering transmitters.
- D is a set of labels $d_{vw} \in \mathbb{N}_0^+$, such that d_{vw} is the highest separation between the frequency assigned to the transmitter v and the one assigned to w that generates unacceptable interference. Let $f(v)$ be the frequency assigned to transmitter v, then if $|f(v) - f(w)| > d_{vw}$, the interference involving the two transmitters is acceptable.
- P is a set of labels $p_{vw} \in \mathbb{N}^+$ such that p_{vw} is the cost to be paid if the separation between the frequencies of transmitters v and w is less than or equal to d_{vw}, that is, if $|f(v) - f(w)| \leq d_{vw}$.
- $B_v \subsetneq F$ is the set of locally blocked frequencies for each vertex $v \in V$.

The objective of the FS-FAP is to find an assignment of frequencies to transmitters that minimizes the sum of p_{vw} over all pairs $vw \in E$ for which $|f(v) - f(w)| \leq d_{vw}$.

Eisenblätter's Problem Formulation

Let $G = (V, E)$ be an undirected graph. The vertices of the graph are called carriers and represent the TRXs. The frequency spectrum F is a finite carrier spectrum interval in $\mathbb{Z}^+$, the set of nonnegative integers, representing the range of channels. For every carrier $v \in V$, a set $B_v \subsetneq F$ of blocked channels is specified (where B_v may be empty). The channels in $F \setminus B_v$ are available at transmitter v. For an edge $vw \in E$, d_{vw} gives the separation necessary between the channels assigned to v and w. c_{vw}^{co} and c_{vw}^{ad} respectively denote the co-channel and adjacent channel interference between v and w. Eisenblätter in [15] formulates the FS-FAP using a function $y : V \to F$. An assignment is feasible if every carrier $v \in V$ is assigned an available frequency and all separation requirements are met, i.e.,

$$y_v \in F \setminus B_v \qquad \forall v \in V, \tag{1.1}$$

$$|y_v - y_w| \geq d_{vw} \qquad \forall vw \in E. \tag{1.2}$$

Therefore, given a carrier network, the FS-FAP problem is as follows:

$$\min_{y\ feasible} \sum_{\substack{vw \in E \\ y_v = y_w}} c_{vw}^{co} + \sum_{\substack{vw \in E \\ |y_v - y_w| = 1}} c_{vw}^{ad} \tag{1.3}$$

In [15], Eisenblätter relaxed the FS-FAP by modifying the objective function and weakening the constraints. The relaxed frequency planning is done by dropping the adjacent channel interference, ignoring the locally blocked channels, and restricting the minimum separation d_{vw} to equal at most 1. A partition of V into at most $|F| = k$ disjoint carriers has to be determined. Thus, the frequency assignment problem is reduced to a minimum k-partition problem where the graph is $G = (V, E)$ and the edge weights of G are derived from c^{co} and the modified d_{vw}.

1.3 Contribution of this Thesis

The main contribution of this thesis is the design and implementation of a semidefinite programming-based branch-and-cut (SBC) algorithm to solve the minimum k-partition problem. Experimental results show that the semidefinite programming (SDP) relaxation provides tighter lower bounds for the MkP problem when compared to the LP relaxation. Additionally, the SDP bound can be tightened by the addition of valid inequalities. Hence, a sequence of SDP relaxations with separation of inequalities is solved at each node of the branch-and-cut tree. To get upper bounds, we devise an iterative clustering heuristic (ICH), a novel primal heuristic to find a feasible solution at each node of the branch-and-cut tree. Computational experiments show that the ICH heuristic provides good feasible solutions and tight upper bounds. This motivates the use of ICH in the SBC algorithm.

In Chapter 2, the technical definition and literature background about the minimum k-partition problem is given, and previous approaches proposed to tackle it and closely related problems are discussed. In Chapter 3, a new SDP relaxation is described. This new relaxation is a lifting that experimentally provides in most cases tighter bounds than the previous SDP relaxation in the literature but requires more computational time. The branch-and-cut algorithm and the novel primal heuristic are presented in Chapter 4. The novel heuristic is compared to the hyperplane rounding of Frieze and Jerrum [18] in terms of bounds. Computational results for the branch-and-cut algorithm on several important classes of instances, and for different values of k, are presented in Chapter 5. The computational results show the potential of SBC for tackling the MkP problem. Finally, conclusions and future research directions are discussed in Chapter 6.

Chapter 2

Formulations and Relaxations of MkP

In this chapter, we begin by giving some definitions related to graph theory. We then present well-known formulations of the MkP in the literature. Furthermore, we introduce two types of relaxations, the linear programming and the SDP relaxations. We strengthen these two relaxations using valid inequalities. Next using experimental analysis we show that the SDP relaxation is stronger which motivates its use in a branch-and-cut algorithm. Finally, we give an overview of several approaches related to this problem. However, none of these approaches address the same version of the minimum k-partition problem that our method does.

2.1 Mathematical Preliminaries

2.1.1 Graphs

A graph $G = (V, E)$ consists of a finite set V of vertices and a finite set E of edges. Every edge $e \in E$ joins two vertices u and v (which are called endnodes) and is denoted as uv. Two vertices are said to be adjacent if they are joined by an edge. The degree of a vertex $v \in V$ is the number of edges that have v as an endnode.

A graph G with n vertices is said to be complete if every two vertices in G are adjacent. The complete graph on n vertices is denoted by K_n. A graph G is said to be k-partite if its vertex set can be divided into k partitions such that no two vertices in any partition are adjacent. For a k-partite graph G, if each partition is a complete graph then G is called a k-partite complete graph.

For a graph $G = (V, E)$, a graph $H = (W, F)$ is said to be a subgraph of G if $W \subseteq V$ and $F \subseteq E$. An induced subgraph is a subset of the vertices of the graph together with all the edges whose endnodes are both in this subset. $G[W]$ denotes the subgraph of G induced by the vertex set W. The set W is said to induce a clique in G if $G[W]$ is a complete graph. Given a k-partition $(S_1, ..., S_k)$ of V in G, where $S_i \subset V$ and $1 \leq i \leq k$, the set of edges in G having one endnode in S_i and the other endnode in $V \backslash S_i$ is called the cut determined by S_i and is denoted by $\delta_G(S_i)$. Given a set E and a subset $S \subseteq E$, then the incidence vector of S is the vector $\chi^S \in \mathbb{R}^E$ defined by

$$\chi_e^S = \begin{cases} 1 & \text{if } e \in S, \\ 0 & \text{if } e \in E \backslash S. \end{cases}$$

2.1.2 Polyhedra

Given a set $X \subseteq \mathbb{R}^n$, the *convex hull* of X is defined as

$$\text{Conv}(X) = \left\{ y \in \mathbb{R}^n \mid y = \sum_{x \in X} \lambda_x x, \;\; \lambda_x \geq 0 \text{ for all } x \in X, \text{ and} \sum_{x \in X} \lambda_x = 1 \right\}$$

A set X is said to be *convex* if Conv(X)$= X$ [12].
The polar X^o of $X \subseteq \mathbb{R}^n$ is defined as

$$X^o = \{x \in \mathbb{R}^n \mid x^T y \leq 1, \, \forall \, y \in X\},$$

where

$$x^T y = \sum_{i=1}^{n} x_i y_i \quad \text{for } x, y \in \mathbb{R}^n.$$

Cones and Polytopes

Given a subset $C \subseteq \mathbb{R}^n$, the set C is said to be a *cone* if $\mathbb{R}_+(C) = C$. The polar of a cone C is the cone defined as

$$C^o = \{x \in \mathbb{R}^n \mid x^T y \leq 0, \ \forall y \in C\}.$$

Proposition 1 *If C is a cone, then $C^o = \{x \in \mathbb{R}^n \mid x^T y \leq 1, \forall\, y \in C\}$.*

Proof. Suppose $x \in X^o$, so x satisfies $x^T y \leq 1$, $\forall y \in C$. But $x \notin C^o$, that is $\exists\, y \in C$ such that $x^T y > 0$.

$$\begin{aligned} &\text{So, } 0 < x^T y \leq 1 \quad y \in C, \\ &\text{but } y \in C \Longrightarrow \mathbb{R}_+ y \subseteq C \Longrightarrow \alpha y \in C \quad \forall \alpha \geq 0. \end{aligned}$$

Therefore, $0 \leq x^T(\alpha y) \leq 1$, $\forall \alpha \geq 0$. However, if we choose $\alpha = \frac{1}{x^T y} + 1$, then $\alpha(x^T y) = 1 + x^T y \geq 1$. This leads to a contradiction, so $x \in X^o \backslash C^o$ does not exist $\Longrightarrow X^o = C^o$. $\square$

Let A be an $m \times n$ matrix and $b \in \mathbb{R}^m$, then the set

$$\{x \in \mathbb{R}^n \mid Ax \leq b\}$$

is called a polyhedron. When the vector b is the zero vector then the polyhedron is a cone. Every convex set of the form Conv(X), where X is finite, is called a polytope. Define the cut polytope CUT$_n$ to be

$$\text{CUT}_n := \text{Conv}\left(\{\chi \in \mathbb{R}^E \mid \chi = \chi^{\delta_{K_n}(S)}, \ S \subseteq V_n\}\right).$$

Faces

Let $P \subseteq \mathbb{R}^n$ be a polytope. A set $F \subseteq P$ is called a *face* of P if for every $x \in F$, whenever $x = \alpha y + (1-\alpha)z$ where $0 \leq \alpha \leq 1$ and $y, z \in P$ implies that $y, z \in F$. The only face of dimension dim(P) is the polytope P itself. Every face of dimension dim$(P)-1$ is called a *facet* of P. A face of dimension 0 is of the form $\{x\}$ and x is said to be a vertex of P. A face of dimension 1 is an edge of P. A face F is said to be a simplex face if the vertices of P lying in F are affinely independent. Given a vector $v \in \mathbb{R}^n$ and a scalar $v_0 \in \mathbb{R}$, the inequality $v^T x \leq v_0$ is said to be valid for P if $v^T x \leq v_0$ holds for all $x \in P$. Therefore, the set

$$F = \{x \in P \mid v^T x = v_0\}$$

is a face of P.

2.2 Linear Programming Approach

2.2.1 Definition and Integer Linear Programming Formulation of MkP

Definition 1 *An instance of the minimum k-partition problem consists of an undirected graph $G = (V, E)$ with edge weights w_{ij} of the edges, and a positive integer $k \geq 2$. The objective is to find a partition of V into at most k disjoint partitions $V_1, ..., V_k$ such that $\sum_{l=1}^{k} \sum_{i,j \in V_l} w_{ij}$ is minimized.*

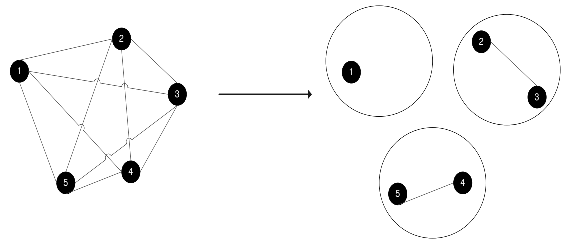

Figure 2.1: A k-partition of a graph with $|V| = 5$ and $k = 3$.

Without loss of generality the graph $G = (V, E)$ can be completed to $K_{|V|}$ and the edge weighting extended by assigning a weight of zero to all the new edges. The edge set is then

$E = \{ij \mid 1 \leq i < j \leq n\}$. Define the variable z_{ij} to be

$$z_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same partition,} \\ 0 & \text{otherwise.} \end{cases}$$

Chopra and Roa in [9] considered the following integer linear programming (ILP) formulation:

$$\textbf{(ILPMKP)} \quad \min \sum_{i,j \in V} w_{ij} z_{ij} \tag{2.1}$$

$$\text{s.t. } z_{ih} + z_{hj} - z_{ij} \leq 1 \qquad \forall\, h, i, j \in V \tag{2.2}$$

$$\sum_{i,j \in Q} z_{ij} \geq 1 \qquad \forall\, Q \subseteq V \text{ where } |Q| = k+1 \tag{2.3}$$

$$z_{ij} \in \{0, 1\} \qquad \forall\, i, j \in V, \tag{2.4}$$

where inequalities (2.2) and (2.3) are the triangle and clique inequalities respectively.

Constraint (2.2) requires the values of the variables to be consistent. For example, if z_{ih} and z_{hj} indicate that i, h, and j are in the same partition, then by transitivity the value of z_{ij} has to reflect that as well. Constraint (2.3) imposes that at least two from every subset of $k+1$ vertices have to be in the same partition. Together with the constraints (2.2), this implies that there are at most k partitions. There are $3\binom{|V|}{3}$ triangle inequalities and $\binom{|V|}{k+1}$ clique inequalities. Note that for (ILPMKP), if we remove the clique inequalities then we will get the trivial zero vector solution in case of all positive weights w_{ij}.

Define the polytope $\mathcal{P}_k$ as the convex hull of all integer points of (ILPMKP):

$$\mathcal{P}_k = \text{ Conv } \{z_{ij} \in \{0,1\} \mid z_{ih} + z_{hj} - z_{ij} \leq 1 \quad \forall\, h, i, j \in V;$$
$$\sum_{i,j \in Q} z_{ij} \geq 1 \quad \forall\, Q \subseteq V \text{ where } |Q| = k+1 \}.$$

The hypermetric inequalities are a class of valid in inequalities to the polytope $\mathcal{P}_k$ and they are defined as follows:

Definition 2 *[12] Given $k \geq 3$, the complete graph K_n and the vertex weights $b_v \in \mathbb{Z}$ with $\sum_{v \in V} b_i = \eta \geq 0$, the hypermetric inequality for the polytope $\mathcal{P}_k$ is defined as:*

$$\sum_{ij \in E} b_i b_j z_{ij} \geq \sum_{ij \in E} b_i b_j - f_{hm}(\eta, k). \tag{2.5}$$

Assume that for the integers k, t and q

$$\sum_{i \in V} b_i = tk + q = \eta,\ t \geq 0,\ 0 \leq q < k.$$

The hypermetric inequalities reflect the fact that if we have η nodes and a k-partition of the graph then to maximize the number of edges between these partitions the nodes should be distributed in such a way that we will have $\binom{\eta \bmod k}{2} \lceil \frac{\eta}{k} \rceil^2$ edges between the q subsets containing $t+1$ nodes, $\binom{k - \eta \bmod k}{2} \lfloor \frac{\eta}{k} \rfloor^2$ edges between the $n-q$ subsets containing t nodes, and $(\eta \bmod k)(k - \eta \bmod k)\lceil \frac{\eta}{k} \rceil \lfloor \frac{\eta}{k} \rfloor$ edges between the subsets containing $t+1$ and t nodes. The hypermetric inequality is a generalization of the clique inequality that was described earlier in this section.

Another IP formulation was proposed by Chopra and Roa in [8]. In [8], the authors define, in addition to the binary z_{ij} edge variables, k binary variables, y_i^l, one for each vertex i, for $l = 1, \ldots, k$, and require that $y_i^l = 1$ if and only if vertex i is in the lth partition and 0 otherwise. This formulation is defined as follows:

$$\textbf{(ILPMKP2)} \quad \min \sum_{ij \in E} w_{ij} z_{ij} \tag{2.6}$$

$$\text{s.t. } y_i^l + y_j^l - z_{ij} \leq 1 \qquad \forall\, ij \in E \quad \forall\, l \in \{1, \ldots, k\} \tag{2.7}$$

$$y_i^l - y_j^l + z_{ij} \leq 1 \qquad \forall\, ij \in E \quad \forall\, l \in \{1, \ldots, k\} \tag{2.8}$$

$$-y_i^l + y_j^l + z_{ij} \leq 1 \qquad \forall\, ij \in E \quad \forall\, l \in \{1, \ldots, k\} \tag{2.9}$$

$$\sum_{l=1}^{k} y_i^l = 1 \qquad \forall\, i \in V \tag{2.10}$$

$$y_i^l, \quad z_{ij} \in \{0, 1\}, \tag{2.11}$$

where inequalities (2.7), (2.8), and (2.9) ensure that if vertices i and j are in the same partition then $z_{ij} = 1$, otherwise $z_{ij} = 0$. Equality (2.10) assigns each vertex to exactly one partition.

(ILPMKP2) doesn't require the graph G to be complete. Hence, for sparse graphs it has fewer variables than (ILPMKP). However, for a given k-partition, (ILPMKP2) has $k!$ different ways of allocating vertices to partitions whereas (ILPMKP) has a unique one. For the rest of the thesis we will consider the (ILPMKP) formulation.

2.2.2 Linear Programming Relaxation

Using the (ILPMKP) formulation with GAMS and the CPLEX solver, it is only possible to solve instances with up to 20 nodes in a reasonable time due to the number of constraints that grow exponentially as n or k are increased. For example, when $k = 3$ and $n = 20$ we have 1140 triangle inequalities and 9690 clique inequalities; and when $k = 4$ and $n = 20$, while we have the same number of triangle inequalities, we have 15504 clique inequalities. Therefore, we relax the binary variable z_{ij} by replacing the feasible variable values $\{0, 1\}$ by their convex hull $[0, 1]$. Hence, we have the following LP relaxation:

$$\textbf{(LPMKP)} \quad \min \sum_{i,j \in V} w_{ij} z_{ij} \tag{2.12}$$

$$\text{s.t. } z_{ih} + z_{hj} - z_{ij} \leq 1 \qquad \forall\, h, i, j \in V \tag{2.13}$$

$$\sum_{ij \in Q} z_{ij} \geq 1 \qquad \forall\, Q \subseteq V \text{ where } |Q| = k + 1 \tag{2.14}$$

$$0 \leq z_{ij} \leq 1 \qquad \forall\, i, j \in V. \tag{2.15}$$

The feasible set of the resulting LP relaxation is the polytope denoted by $\mathcal{P}_k^{LP}$. Solving the LP relaxation still requires a lot of memory and computational time for large k or n values.

Separating the clique inequalities would significantly decrease the number of constraints, hence decreasing computational time and the needed memory space. The motivation for this method is that a complete description of $\mathcal{P}_k^{LP}$ usually involves too many constraints to handle efficiently, and many of them won't be binding at the optimal solution, so instead of adding them all in at the beginning, we add in only violated inequalities in an iterative fashion. Thus we have a sequence of LP relaxations, where we start by solving an initial LP problem with few clique inequalities (LPMKP-C), then check if the solution satisfies all the clique inequalities; if yes then we terminate, else we add the most violated ones (outside a tolerance tol_c), decrease the tolerance by a certain value ε_{LP}, and solve again. Algorithm 1 gives a detailed description of the clique separation algorithm and Figure 2.2 illustrates the idea of separation.

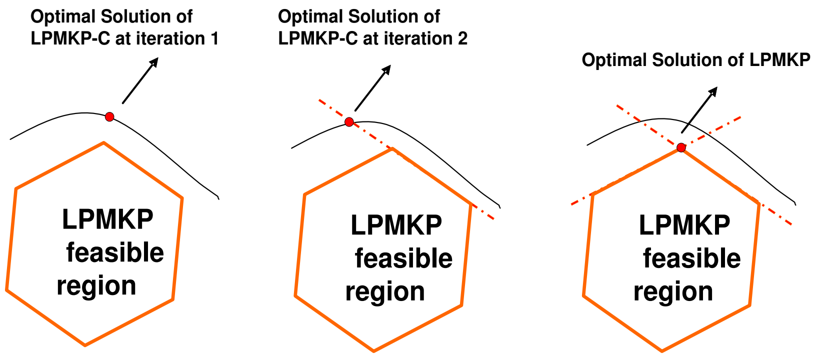
Optimal Solution of LPMKP-C at iteration 1
Optimal Solution of LPMKP-C at iteration 2
Optimal Solution of LPMKP
LPMKP feasible region
LPMKP feasible region
LPMKP feasible region

Figure 2.2: Separation of clique inequalities for (LPMKP).

In the initialization phase, few clique constraints are added because removing all the clique inequalities from (LPMKP) would result in a trivial solution where all $z^*_{ij} = 0$. The number of clique inequalities added at the beginning depends on the size of the problem (i.e., n) and the weight matrix. We sum the edge weights of every clique, take the highest n of them and add the corresponding cliques at the beginning.

We note that, given the complete graph K_n, $n \geq 3$, and a vector $z \in [0, 1]$, checking whether the clique inequality is met or not for all set of cliques $Q \subseteq V$ with $|Q| = k + 1$ is a $\mathcal{NP}$-hard problem (see Proposition 7.31 of [15]). In Algorithm 1, the separation of clique inequalities is done by enumeration. This is a simple way of finding violated clique inequalities however, in future research we want to improve this separation algorithm to reduce the computational time.

Algorithm 1 Separating Clique Inequalities for (LPMKP)

```
1- Initialization
    1.a- Remove all clique inequalities from the relaxed (LPMKP) problem.
    1.b- Add clique inequalities that cover cliques with highest weights.
    1.c- Solve this LP relaxation.
    1.d- Get an initial solution.
    1.e- Check for violated clique inequalities.
2- Adding clique inequalities.
    2.a- Add the violated clique inequalities outside a tolerance $tol_c$.
    2.b- Decrease $tol_c$ by $\varepsilon_{LP}$.
    2.c- Solve this LP relaxation.
    2.d- Get a solution.
3- Termination.
    3.a- If the solution violates clique inequalities
        3.a.1- Return to step 2.
    3.b- Else
        3.b.1- Terminate (none of the clique inequalities are violated).
```

Barahona et al. in [4] designed a cutting plane algorithm for the max-cut problem within a branch-and-bound framework. They solve at the root node the trivial LP

$$\min \sum_{i,j \in V} w_{ij} z_{ij}$$
$$\text{s.t. } 0 \leq z_{ij} \leq 1,$$

and then generate cutting planes using odd cycle inequalities. The cutting planes are added not only at the root node, but also at each node of the branch-and-bound tree. The computational results presented in Barahona et al. [4] show that the optimal solution for graphs of any density with up to $n = 30$ nodes can be computed in reasonable time. But with an increasing number of nodes, the limits on the largest density that can be handled decreases rapidly. Therefore, this algorithm becomes impractical for dense instances.

Results on an improved version of this algorithm are presented in [31] for the max-cut problem. They focus on solving Ising spin glass problems using an LP-based branch-and-cut approach. These types of graphs are sparse, and since linear approaches can exploit the sparsity, they can solve problems of large sizes. However, for dense graphs the LP-based approach isn't suitable for solving these types of graphs.

2.3 Semidefinite Programming Approach

Semidefinite relaxations of combinatorial optimization problems were pioneered by Lovasz [33] in 1979 in order to compute the Shannon capacity of a graph. This problem arises in information theory where the graph represents the channel, the vertices represent single letters in the alphabet, and the edge ij indicates that letters i and j are confusable in transmission. Moreover, in 1995 Goemans and Williamson used a similar SDP relaxation in an approximation algorithm for the max-cut problem ($k = 2$). Frieze and Jerrum extended it to the max k-cut problem. These relaxations are described in Section 2.4.3 and Section 2.4.1 respectively.

2.3.1 Basic Concepts of Semidefinite Programming

A semidefinite programming (SDP) problem consists of minimizing a linear function of a symmetric matrix variable X subject to linear equality constraints on X and X being positive semidefinite. The set of positive semidefinite matrix is a closed convex cone but it is not polyhedral. The duality theory for semidefinite programming isn't as smooth as that of linear programming since a gap between the optimal primal and dual objective function values is possible. The standard primal semidefinite program is formulated as follows:

$$\begin{array}{lll} \textbf{(PSDP)}\ \min & < C, X > & \\ \text{s.t.} & < A_i, X >= b_i & i = 1, \ldots, m \\ & X \succeq 0, & \end{array}$$

where $< C, X >$ is the trace of matrix $C^T X$. The corresponding dual semidefinite program is:

$$\begin{aligned} \textbf{(DSDP)} \ \max \quad & b^T y \\ \text{s.t.} \quad & Z = C - \sum_{i=1}^{m} y_i A_i \\ & Z \succeq 0. \end{aligned}$$

Let $\mathcal{S}_n^+$ denote the space of $n \times n$ positive semidefinite matrices and $\mathcal{S}_n^{++}$ denote the space of $n \times n$ positive definite matrices.

Definition 3 *$X \in \mathcal{S}_n^+$ is said to be feasible for (PSDP) if $< A_i, X >= b_i$.*
Similarly, $y \in \mathbb{R}^m$ and $Z \in \mathcal{S}_n^+$ are said to be feasible for (DSDP) if $C - \sum_{i=1}^m y_i A_i = Z$.

If X is a primal feasible solution and the pair (y, Z) is a dual feasible solution, then the duality gap is defined as the difference between the objective values of (PSDP) and (DSDP).

Definition 4 *If $X \in \mathcal{S}_n^+$, $y \in \mathbb{R}^m$ and $Z \in \mathcal{S}_n^+$ are feasible for (PSDP) and (DSDP), then the duality gap is defined as:*

$$\langle C, X \rangle - \langle b, y \rangle$$

We have $\langle C, X \rangle - \langle b, y \rangle = \langle A^T y + Z, X \rangle - \langle AX, y \rangle = \langle Z, X \rangle$ and because $\langle Z, X \rangle \geq 0$, then $\langle C, X \rangle - \langle b, y \rangle \geq 0$. That is, the objective value of any primal feasible solution is greater or equal to the objective of any dual feasible solution. This property is known as weak duality. When the gap between the primal and the dual objective is zero, we say that strong duality holds and this gives optimal solutions to (PSDP) and (DSDP). However, unlike for linear programming, Vandenberghe and Boyd [44] exemplify that optimality does not imply that we have a zero gap between the primal and the dual objective ($\langle Z, X \rangle = 0$). Slater's constraint qualification provides sufficient (but not necessary) conditions to have problems with zero duality gap at the optimal solution.

Definition 5 *(PSDP) satisfies Slater's condition if there exists a strictly feasible X that is $X \in \mathcal{S}_n^{++}$ with $AX = b$.*
(DSDP) satisfies Slater's condition if there exists a strictly feasible pair (y, Z) that is $Z \in \mathcal{S}_n^{++}$ and $A^T y - Z = C$.

Theorem 1 *[14] Let* $p^* = \inf\{\langle C, X\rangle : \langle A_i, X\rangle = b_i, X \in \mathcal{S}_n^{++}\}$ *and* $d^* = \sup\{\langle b, y\rangle : Z = C - \sum_{i=1}^m y_i A_i, Z \succeq 0\}$*, then we have the following:*

- *If (PSDP) satisfies Slater's condition and* p^* *is finite, then* $p^* = d^*$ *and this value is attained for (DSDP).*
- *If (DSDP) satisfies Slater's condition and* d^* *is finite, then* $p^* = d^*$ *is attained for (PSDP).*
- *If (PSDP) and (DSDP) both satisfy Slater's condition, then* $p^* = d^*$ *is attained for both problems.*

2.3.2 Software for Solving SDP

Interior-point algorithms, as well as the spectral bundle method, low-rank approach, and augmented Lagrangian method [37] have been used to solve semidefinite programs. A list of links to the various packages can be found on the Semidefinite Programming Website [21].
SeDuMi [41] and SDPT3 [42] are interior-point codes written in Matlab. Another interior-point code is CSDP by Borchers [6] however, it is written in C. Helmberg implemented the Spectral Bundle Method, SBMethod. S. Burer and R. Monteiro [7] implemented SDPLR that solves SDPs via low-rank factorization. PENNON software [27] solves SDPs using augmented Lagrangian method.

In this thesis, we used the CSDP solver to solve the SDPs. CSDP is an interior-point primal-dual algorithm. These methods compute both primal and dual feasible solutions. The algorithm stops when an optimal solution is found.

2.3.3 Basic Semidefinite Relaxation of MkP

To obtain an SDP formulation of MkP we make use of the following lemma.

Lemma 1 *[15] For all integers n and k satisfying $2 \le k \le n+1$, the following holds:*

1. *There exists k unit vectors $\bar{u}_1, ..., \bar{u}_k \in \mathbb{R}^n$ such that $\langle \bar{u}_i\ ,\ \bar{u}_j \rangle = \frac{-1}{k-1}$ for all $i \neq j$.*

2. *Any given k unit vectors $u_1, ..., u_k \in \mathbb{R}^n$ satisfy $\sum_{i<j} \langle u_i\ ,\ u_j \rangle \ge \frac{-k}{2}$ and if $\langle u_i\ ,\ u_j \rangle \le \delta$ for all $i \neq j$, then $\delta \ge \frac{-1}{k-1}$.*

To shed some light on the structure of the SDP relaxation that follows, we include the details of the proof of part 1 of the lemma.

Proof. Let $k = n+1$ and let t_i be an $(n+1)$-dimensional vector for $1 \le i \le n+1$ with all its entries equal to $-\sqrt{\frac{1}{n(n+1)}}$ except the ith entry which is equal to $\sqrt{\frac{n}{n+1}}$.
Then

$$\begin{aligned}\langle t_i\ ,\ t_j \rangle &= (n-1) \times \left(-\sqrt{\frac{1}{n(n+1)}}\right)^2 - 2\sqrt{\frac{n}{n+1}}\sqrt{\frac{1}{n(n+1)}} \\ &= \frac{n-1}{n(n+1)} - \frac{2}{n+1} \\ &= -\frac{1}{n} \qquad i \neq j.\end{aligned}$$

If the t_i were vectors in $\mathbb{R}^n$ instead of $\mathbb{R}^{n+1}$ then the first part of the lemma would be proved. However, the subspace spanned by the vectors t_i is at most n-dimensional because

$$\langle t_i\ ,\ [1, 1, ..., 1]^T \rangle = -n\sqrt{\frac{1}{n(n+1)}} + \sqrt{\frac{n}{n+1}} = 0.$$

Therefore, we can rotate the coordinate system so that $[1, 1, ..., 1]^T$ turns into a multiple of the vector $[0, 0, ..., 0, 1]^T$. Since the last coordinate of each t_i is zero in the new coordinate system, we can obtain the desired vectors $\bar{u}_i$ from the t_i by truncation. Therefore, $\langle \bar{u}_i, \bar{u}_j \rangle = -\frac{1}{n} = \frac{-1}{k-1}$ for all $i \neq j$. $\square$

According to Lemma 1, we may fix a set $U = \{u_1, ..., u_k\} \subseteq \mathbb{R}^n$ of unit vectors with

$$\begin{cases} \langle u_i, u_j \rangle = \frac{-1}{k-1} & \text{for } i \neq j \\ \langle u_i, u_i \rangle = 1 & \text{for } i = 1, ..., k \end{cases}$$

The minimum k-partition problem is finding an assignment $x : V \to U$ that minimizes the expression

$$\sum_{i,j \in V} w_{ij} \frac{(k-1) < x_i, x_j > +1}{k}.$$

Assemble the scalar products into a square matrix X such that it is indexed row and column wise by V. The symmetric matrix X is characterized by the following:

1. All entries on the principal diagonal are ones.
2. All off-diagonal elements are either $\frac{-1}{k-1}$ or 1.
3. $X \succeq 0$.

The minimum k-partition problem was formulated in [16] as follows:

$$\begin{aligned}
\min \sum_{i<j \in V} w_{ij} \frac{(k-1)X_{ij}+1}{k} & & (2.16)\\
\text{s.t. } X_{ii} = 1 & \qquad \forall i \in V & (2.17)\\
X_{ij} \in \{\frac{-1}{k-1}\ , 1\} & \qquad \forall i < j \in V & (2.18)\\
X \succeq 0 & & (2.19)
\end{aligned}$$

Relaxations of the MkP can be obtained by relaxing some of the constraints. All possible solutions of this problem are feasible for its relaxation, and the optimal value of the relaxations, are lower bounds on its optimal value.
In particular, replacing constraint (2.18) by $\frac{-1}{k-1} \leq X_{ij} \leq 1$ results in a semidefinite relaxation. However, the constraint $X_{ij} \leq 1$ can be dropped since it is enforced implicitly by the following two constraints:

$$X_{ii} = 1 \text{ and}$$

$$X \succeq 0$$

This is because $X \succeq 0$ is equivalent to all principal minors being non-negative. For $X \in \mathcal{S}_n$, $X_{ii} = 1$ and $i \neq j$, consider the following principal minor:

$$\begin{pmatrix} 1 & X_{ij} \\ X_{ij} & 1 \end{pmatrix}.$$

By positive semidefiniteness,

$$\begin{aligned}\det \begin{pmatrix} 1 & X_{ij} \\ X_{ij} & 1 \end{pmatrix} &= 1 - X_{ij}^2 \geq 0 \\ &\implies X_{ij}^2 \leq 1 \\ &\iff -1 \leq X_{ij} \leq 1.\end{aligned}$$

Hence, the SDP relaxation is as follows:

(SMKP)

$$\begin{aligned} \min \ & \sum_{i<j \in V} w_{ij} \frac{(k-1)X_{ij}+1}{k} && && (2.20) \\ \text{s.t. } & X_{ii} = 1 && \forall i \in V && (2.21) \\ & X_{ij} \geq \frac{-1}{k-1} && \forall i < j \in V && (2.22) \\ & X \succeq 0. && && (2.23) \end{aligned}$$

Denote by $E^{ij}(n)$ the $n \times n$ symmetric matrix with entries equal to 1 at positions (i, j) and (j, i), and zeros elsewhere. For every matrix $W \in \mathcal{S}_n$, the primal semidefinite program is

$$\begin{aligned} \min \ & \sum_{1 \leq i < j \leq n} w_{ij} X_{ij} \qquad (2.24) \\ \text{s.t. } & \langle E^{ii}, X_{ii} \rangle = 1 \\ & \langle E^{ij}, X_{ij} \rangle \geq \frac{-1}{k-1}, \quad \forall\, i, j \in \{1, ..., n\},\ i < j \\ & X \succeq 0. \end{aligned}$$

and the dual semidefinite program is

$$\begin{aligned} \max \ & \sum_{i=1}^{n} y_{ii} - \sum_{1 \leq i < j \leq n} \frac{y_{ij}}{k-1} \qquad (2.25) \\ \text{s.t. } & W - \sum_{1 \leq i \leq j \leq n} y_{ij} E^{ij} \succeq 0, \quad y_{ij} \geq 0. \end{aligned}$$

The dual variable associated to the primal constraint $\langle E^{ii}, X_{ii}\rangle = 1$ is y_{ii} and that associated to the primal constraint $\langle E^{ij}, X_{ij}\rangle \geq \frac{-1}{k-1}$ is y_{ij}.

To investigate the strength of the semidefinite relaxation, we need to relate the solution set of the semidefinite relaxation to the polytope $\mathcal{P}_k$ introduced in Section 2.2.1. Let $\Psi_{k,n} = \{X \in \mathbb{R}^{n\times n} \mid X \succeq 0, X_{ii} = 1, X_{ij} \geq \frac{-1}{k-1}$ where $i,\ j \in \{1, ..., n\}\}$, that is $\Psi_{k,n}$ is the set of feasible solutions of the semidefinite relaxation stated previously.
Define an affine mapping that projects $\Psi_{k,n}$ into $\mathbb{R}^{\binom{n}{2}}$ by letting

$$\begin{aligned}\zeta_{k,n} :& X \to z \\ & \mathcal{S}_n \to \mathbb{R}^{\binom{n}{2}}.\end{aligned}$$

Moreover, define $T_k : \mathbb{R} \to \mathbb{R}$ to be the affine transformation $x \mapsto \frac{k-1}{k}x + \frac{1}{k}$. The affine transformation T_k is extended by letting $T_k : \mathcal{S}_n \to \mathcal{S}_n$, where $S \mapsto \frac{k-1}{k}S + \frac{1}{k}E(n,n)$ and $E(n,n)$ is an $n \times n$ matrix with all entries are equal to 1. Therefore, we will have $X \mapsto \zeta_{k,n}(X) = z$, where $z_{ij} = (T_k(X))_{ij}$ and

$$\Theta_{k,n} = \zeta_{k,n}(\Psi_{k,n}) = \{\zeta_{k,n}(X) \mid X \in \Psi_{k,n}\}$$

The restriction of $\zeta_{k,n}$ to $\Psi_{k,n}$ is one-to-one, and $\zeta_{k,n} \mid_{\Psi_{k,n}} : \Psi_{k,n} \to \Theta_{k,n}$ is an affine bijection.
To prove that T_k is an affine transformation, consider the following:

$$\begin{aligned}T_k(X) &= \frac{k-1}{k}X + \frac{1}{k}E(n,n) \\ &= \underbrace{\frac{k-1}{k}X}_{\text{Linear Transformation}} + \underbrace{\frac{1}{k}E(n,n)}_{\text{Constant}} \\ &\implies T_k(X) \text{ has the form of } AX + b\end{aligned}$$

Therefore, T_k is an affine mapping since it is a linear transformation followed by a translation.

A transformation is a bijection if it is one-to-one and onto. First we show that the transformation is one-to-one:
Given $X \in \mathcal{S}_n$ and $Y \in \mathcal{S}_n$, then the transformation T_k defined on $\mathcal{S}_n$ is one-to-one if $T_k(X) = T_k(Y)$ only if $X = Y$.
Suppose $T_k(X) = T_k(Y)$ then:

$$\begin{aligned} \frac{k-1}{k}X + \frac{1}{k}E(n,n) &= \frac{k-1}{k}Y + \frac{1}{k}E(n,n) \\ \frac{k-1}{k}X &= \frac{k-1}{k}Y \\ X &= Y. \end{aligned}$$

Therefore, T_k is one-to-one.

Finally, we need to show that T_k is an onto transformation:
The transformation T_k defined on $\mathcal{S}_n$ is onto if there is an $X \in \mathcal{S}_n$ such that $T_k(X) = Y$ for all $Y \in \mathcal{S}_n$.
Given $Y \in \mathcal{S}_n$, consider there exists an X such that $T_k(X) = Y$ for all Y. Then $X \in \mathcal{S}_n$ and

$$\begin{aligned} T_k(X) &= \frac{k-1}{k}X + \frac{1}{k}E(n,n) \\ &= Y \\ \implies \frac{k-1}{k}X &= -\frac{1}{k}E(n,n) + Y. \end{aligned}$$

$X = \frac{k}{k-1}Y - \frac{1}{k-1}E(n,n)$ and hence T_k is onto. Since T_k is one-to-one and onto, then it is a bijection. Consequently, T_k is an affine bijection.

For any given $X \in \Psi_{k,n}$, any given $w \in \mathbb{R}^{\binom{n}{2}}$, and the symmetric matrix W with $W_{ii} = 0$ and $W_{ij} = w_{ij}$ for all $1 \leq i < j \leq n$, the following result is obtained:

$$\frac{1}{2}\langle W, T_k(X)\rangle = \langle w, \zeta_{k,n}(X)\rangle \implies \tag{2.26}$$

$$\min \frac{1}{2}\langle W, T_k(X)\rangle \quad \text{s.t. } X \in \Psi_{k,n} \qquad \text{and} \qquad \min \langle w, z\rangle \quad \text{s.t. } z \in \Theta_{k,n}$$

are equivalent.
This can be shown as follows:

$$\begin{aligned}\langle W, T_k(X)\rangle &= \text{trace } (T_k^T(X)\, W)\\ &= \text{trace } (W\, T_k(X)) \qquad \text{since } W \text{ and } T_k(X) \in \mathcal{S}_n\\ &= \sum_{ij} w_{ij}(T_k(X))_{ij}\\ &= \sum_{ij} w_{ij} z_{ij}\\ &= 2\sum_{i<j} w_{ij} z_{ij}\\ &= 2\langle w, z\rangle\\ &= 2\langle w, \zeta_{k,n}(X)\rangle\end{aligned}$$

Therefore, $\frac{1}{2}\langle W, T_k(X)\rangle = \langle w, \zeta_{k,n}(X)\rangle$.

The affine image $\Theta_{k,n}$ of the truncated elliptope $\Psi_{k,n}$ contains the polytope $\mathcal{P}_k$.

Lemma 2 *[15] $\Theta_{k,n}$ is the semidefinite relaxation of $\mathcal{P}_k$ and they both contain the same integral points.*

Proof. Let $\bar{z}$ be an integral vector in $\Theta_{k,n}$. Under the mapping $\zeta_{k,n}$, $\bar{X}$ is the pre-image of $\bar{z}$. So $\bar{X} \succeq 0$ and all the entries of $\bar{X}$ are either $\frac{-1}{k-1}$ or $+1$ since $z_{ij} = (T_k(X))_{ij}$ for $i < j$. If inequality (2.2) is not satisfied then for some i, j, and k, we have

$$z_{ih} + z_{hj} - z_{ij} \geq 2$$

and hence, the matrix $\bar{z}$ has one of the following as a principal submatrix:

$$\begin{pmatrix}1 & 0 & 1\\ 0 & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \quad \begin{pmatrix}1 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 1\end{pmatrix} \quad \begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 0\\ 1 & 0 & 1\end{pmatrix}.$$

Therefore, the pre-image $\bar{X}$ will have one of the following principal submatrices:

$$\begin{pmatrix}1 & \frac{-1}{k-1} & 1\\ \frac{-1}{k-1} & 1 & 1\\ 1 & 1 & 1\end{pmatrix} \quad \begin{pmatrix}1 & 1 & \frac{-1}{k-1}\\ 1 & 1 & 1\\ \frac{-1}{k-1} & 1 & 1\end{pmatrix} \quad \begin{pmatrix}1 & 1 & 1\\ 1 & 1 & \frac{-1}{k-1}\\ 1 & \frac{-1}{k-1} & 1\end{pmatrix}.$$

However, the determinant of all these matrices is $-(\frac{k}{k-1})^2 < 0$, which contradicts the fact that $\bar{X} \succeq 0$. Therefore, none of these matrices can appear as a principal submatrix of $\bar{X}$ $\Longrightarrow$ inequality (2.2) is not violated.
In addition, if inequality (2.3) is not satisfied then

$$\sum_{ij \in Q} z_{ij} \leq 0,$$

where $|Q| = k+1$. Let $\bar{z}_{QQ}$ be a submatrix of $\bar{z}$ of size $k+1 \times k+1$. If inequality (2.3) is violated, then all of the off-diagonal elements of $\bar{z}_{QQ}$ are equal to 0. Therefore, the pre-image $\bar{X}$ will have the following submatrix of size $(k+1) \times (k+1)$:

$$\bar{X}_{QQ} = \begin{pmatrix} 1 & -\frac{1}{k-1} & \cdots & -\frac{1}{k-1} \\ -\frac{1}{k-1} & 1 & \cdots & -\frac{1}{k-1} \\ \vdots & \vdots & \ddots & \vdots \\ -\frac{1}{k-1} & -\frac{1}{k-1} & \cdots & 1 \end{pmatrix}.$$

However, consider the following:

$$\begin{aligned} e^T(\bar{X}_{QQ})e &= (k+1) + \left[(k+1)^2 - (k+1)\right] \frac{-1}{k-1} \qquad \text{(where } e \text{ is the vector of all ones)} \\ &= (k+1)\left[1 - \frac{(k+1)-1}{k-1}\right] \\ &= (k+1)\left[\frac{(k-1)-k}{k-1}\right] \\ &= -1\left(\frac{k+1}{k-1}\right) < 0 \end{aligned}$$

This implies that $\bar{X}_{QQ} \not\succeq 0$, and hence $\bar{X} \not\succeq 0$. Therefore, no subset Q of size $k+1$ can induce a submatrix $\bar{X}_{QQ}$ with all its off-diagonal elements equal to $-\frac{1}{k-1}$. Consequently, the clique inequality (2.3) is not violated. □

The next two lemmas show that the triangle inequalities in equation (2.2) and clique inequalities in equation (2.3) are more than half satisfied by every point in $\Theta_{k,n}$.

Lemma 3 *[16] Given the complete graph K_n, then for $4 \leq k \leq n$ and $z \in \Theta_{k,n}$*

$$z_{ij} + z_{jh} - z_{ih} \leq 1 + \frac{\sqrt{2(k-2)(k-1)} - (k-2)}{k} \qquad \left[< \sqrt{2}\right] \tag{2.27}$$

holds for every triangle. Moreover, for every clique Q of size $k+1$ in K_n,

$$\sum_{ij \in E(Q)} z_{ij} \geq 1 - \frac{k-1}{2k} \qquad \left[> \frac{1}{2}\right]. \tag{2.28}$$

In the next section, we will describe how the separation of the triangle and clique inequalities is done to strengthen the SDP relaxation.

2.3.4 Strengthening the SDP Relaxation with Valid Inequalities

The SDP relaxation can be further tightened by adding valid inequalities. The two types of valid inequalities to be added are the triangle and the clique inequalities. The resulting relaxation is (SMKP-C).

The first type of valid inequalities to be added are the triangle inequalities. We add violated triangle inequalities at each iteration. The triangle inequalities have the following form:

$$X_{ij} + X_{jh} - X_{ih} \leq 1,$$

where i, j, and $h \in V$. Let us take the following example to illustrate this inequality:

Example:	If i and j are in the same partition $\Rightarrow$	$X_{ij} = 1$
	If j and h are in the same partition $\Rightarrow$	$X_{jh} = 1$
	Now suppose i and h are not in the same partition $\Rightarrow$	$X_{ih} = \frac{-1}{k-1}$.
	Then we have $X_{ij} + X_{jh} - X_{ih} =$	$1 + 1 + \frac{1}{k-1} \nleq 1$
	Therefore, i and h should be in the same partition to have	$X_{ij} + X_{jh} - X_{ih} \leq 1$

The second type of valid inequalities added to the SDP relaxation are the clique inequalities. Recall that for the SDP relaxation, at the optimal solution of the integer problem, each entry X_{ij} should be either 1 or $\frac{-1}{k-1}$. The clique inequalities for (SMKP) have the following form:

$$\sum_{i,j \in Q} X_{ij} \geq -\frac{k}{2} \quad \forall\, Q \subseteq V \text{ where } |Q| = k+1.$$

To show this, recall that the clique inequalities ensure that if we have k partitions and n vertices then each set $Q \subseteq V$ that satisfies $|Q| = k+1$ at leat 2 vertices should be in the same partition. This means that at least one of the $X_{ij} = 1$. Therefore,

$$\begin{aligned}
&\sum_{i<j \in Q} X_{ij} \geq 1 + \sum_{i=1}^{\binom{k+1}{2}-1} \frac{-1}{k-1} \quad \forall\, Q \subseteq V \text{ where } |Q| = k+1. \\
&\sum_{i<j \in Q} X_{ij} \geq 1 + \left[\frac{(k+1)k}{2} - 1\right] \frac{-1}{k-1} \\
\Leftrightarrow &\sum_{i<j \in Q} X_{ij} \geq 1 + \frac{2 - k^2 - k}{2k-2} \\
\Leftrightarrow &\sum_{i<j \in Q} X_{ij} \geq 1 - \frac{(k+2)(k-1)}{2(k-1)} \\
\Leftrightarrow &\sum_{i<j \in Q} X_{ij} \geq 1 - \frac{k+2}{2} \\
\Leftrightarrow &\sum_{i<j \in Q} X_{ij} \geq \frac{-k}{2}
\end{aligned}$$

For example, when we have a 3-partition problem then we consider the 4-clique inequality. For every vertex i, j, k, and h in the vertex set, we have the following clique inequality:

$$X_{ij} + X_{ik} + X_{ih} + X_{jk} + X_{jh} + X_{kh} + \geq -\frac{3}{2}$$

Once the (SMKP) is solved, one can check for violated clique inequalities and add them to the SDP problem, hence getting a better lower bound. These inequalities will strengthen the relaxation by cutting off some feasible solution for (SMKP) that are not feasible for the original problem (ILPMKP).

The SDP relaxation provides a non-trivial solution, hence no inequalities need to be added at the start (in contrast to the LP relaxation). The number of clique inequalities added at each iteration depends on the size of the problem. Moreover, at first we try to satisfy the most violated clique inequalities so we give a tolerance of t_c then this tolerance decreases by a value of ε_{SDP} at each iteration until finally it is the desired $-\frac{k}{2}$ value.

The algorithm for adding clique and triangle inequalities is described in Algorithm 2.

Algorithm 2 Separating Clique and Triangle Inequalities for (SMKP-C)

```
1- Initialization
    1.a- Solve (SMKP)
2- Adding clique and triangle inequalities.
    2.a- Add the most violated clique and triangle inequalities within tolerances $t_c$ and $t_t$
respectively.
    2.b- Decrease $t_c$ and $t_t$ by $\varepsilon_{SDP}$.
    2.c- Solve the SDP problem.
    2.d- Get a solution.
3- Termination.
    3.a- If the solution violates any inequalities
        3.a.1- Return to step 2.
    3.b- Else
        3.b.1- Terminate when none of the inequalities is violated.
```

2.3.5 Properties of the SDP Relaxation

Take any three vertices in different partitions from the set of vertices in the graph $G = (V, E)$. Any fourth vertex should be in the same partition of one of these three vertices. We show that a feasible solution to the SDP automatically satisfies this. Consider four vertices i, j, k, and l with $X_{ij} = -\frac{1}{2}, X_{ik} = -\frac{1}{2}$, and $X_{jk} = -\frac{1}{2}$. Then we have the following principal submatrix:

$$X = \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} & x \\ -\frac{1}{2} & 1 & -\frac{1}{2} & y \\ -\frac{1}{2} & -\frac{1}{2} & 1 & z \\ x & y & z & 1 \end{pmatrix}$$

By positive semidefiniteness,

$$\begin{pmatrix} 1 & -\frac{1}{2} & y \\ -\frac{1}{2} & 1 & z \\ y & z & 1 \end{pmatrix} - \begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ x \end{pmatrix} \begin{pmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ x \end{pmatrix}^T \succeq 0$$

$$\Leftrightarrow \begin{pmatrix} 1 & -\frac{1}{2} & y \\ -\frac{1}{2} & 1 & z \\ y & z & 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & -\frac{x}{2} \\ \frac{1}{4} & \frac{1}{4} & -\frac{x}{2} \\ -\frac{x}{2} & -\frac{x}{2} & x^2 \end{pmatrix} \succeq 0$$

$$\Leftrightarrow \begin{pmatrix} \frac{3}{4} & -\frac{3}{4} & y + \frac{x}{2} \\ -\frac{3}{4} & \frac{3}{4} & z + \frac{x}{2} \\ y + \frac{x}{2} & z + \frac{x}{2} & 1 - x^2 \end{pmatrix} \succeq 0.$$

Taking the determinant:

$\frac{3}{4}(\frac{3}{4} - \frac{3}{4}x^2 - (z + \frac{x}{2})^2) + \frac{3}{4}(-\frac{3}{4} + \frac{3}{4}x^2 - (z + \frac{x}{2})(y + \frac{x}{2})) + (y + \frac{x}{2})(-\frac{3}{4}z - \frac{3}{8}x - \frac{3}{4}y - \frac{3}{8}x) \geq 0$

$\Rightarrow -\frac{3}{4}z^2 - \frac{3}{4}x^2 - \frac{3}{4}y^2 - \frac{3}{2}xy - \frac{3}{2}xz - \frac{3}{2}yz \geq 0$

$\Rightarrow \frac{1}{2}(z^2 + x^2 + y^2) + xy + xz + yz \leq 0$

$\Rightarrow (x + y + z)^2 \leq 0$

Therefore, $x + y + z = 0$. In the discrete case, this means that either x or y or z should be one and the other two entries should be $-\frac{1}{2}$.

In general for $k \geq 3$, we consider $k+1$ vertices with the following principal matrix:

$$X = \begin{pmatrix} 1 & -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & x_1 \\ -\frac{1}{k-1} & 1 & -\frac{1}{k-1} & \dots & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & 1 & x_k \\ x_1 & x_2 & \dots & x_k & 1 \end{pmatrix}$$

By positive semidefiniteness,

$$\begin{pmatrix} 1 & -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & x_2 \\ -\frac{1}{k-1} & 1 & -\frac{1}{k-1} & \dots & x_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & 1 & x_k \\ x_2 & x_3 & \dots & x_k & 1 \end{pmatrix} - \begin{pmatrix} -\frac{1}{k-1} \\ -\frac{1}{k-1} \\ \vdots \\ x_1 \end{pmatrix} \begin{pmatrix} -\frac{1}{k-1} \\ -\frac{1}{k-1} \\ \vdots \\ x_1 \end{pmatrix}^T \succeq 0$$

$$\Leftrightarrow \begin{pmatrix} 1 & -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & x_2 \\ -\frac{1}{k-1} & 1 & -\frac{1}{k-1} & \dots & x_3 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{1}{k-1} & -\frac{1}{k-1} & \dots & 1 & x_k \\ x_2 & x_3 & \dots & x_k & 1 \end{pmatrix} - \begin{pmatrix} \frac{1}{(k-1)^2} & \frac{1}{(k-1)^2} & \cdots & \frac{1}{(k-1)^2} & -\frac{x_1}{k-1} \\ \frac{1}{(k-1)^2} & \frac{1}{(k-1)^2} & \cdots & \frac{1}{(k-1)^2} & -\frac{x_1}{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{1}{(k-1)^2} & \frac{1}{(k-1)^2} & \cdots & \frac{1}{(k-1)^2} & -\frac{x_1}{k-1} \\ -\frac{x_1}{k-1} & -\frac{x_1}{k-1} & \vdots & -\frac{x_1}{k-1} & x_1^2 \end{pmatrix} \succeq 0$$

$$\Leftrightarrow \begin{pmatrix} \frac{(k-1)^2-1}{(k-1)^2} & -\frac{k}{(k-1)^2} & -\frac{k}{(k-1)^2} & \dots & x_2 + \frac{x_1}{k-1} \\ -\frac{k}{(k-1)^2} & \frac{(k-1)^2-1}{(k-1)^2} & -\frac{k}{(k-1)^2} & \dots & x_3 + \frac{x_1}{k-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ -\frac{k}{(k-1)^2} & -\frac{k}{(k-1)^2} & \dots & \frac{(k-1)^2-1}{(k-1)^2} & x_k + \frac{x_1}{k-1} \\ x_2 + \frac{x_1}{k-1} & x_3 + \frac{x_1}{k-1} & \dots & x_k + \frac{x_1}{k-1} & 1 - x_1^2 \end{pmatrix} \succeq 0.$$

Taking the determinant and performing linear algebra operations we get the following:

$$\left(\sum_{i=1}^{k} x_i\right)^2 \leq 0$$

$$\Rightarrow \sum_{i=1}^{k} x_i = 0.$$

In the discrete case, this means that one x_i should be one and the other $k-1$ entries should be $-\frac{1}{k-1}$ since in that case $\sum_{i=1}^{k} x_i = 1 - (\sum_{i=1}^{k-1} \frac{1}{k-1}) = 1 - \frac{k-1}{k-1} = 0$.

2.3.6 Comparison of the LP and SDP Relaxations

The objective function value of a relaxation of the original problem provides a lower bound (in case we are minimizing). A relaxation is strong if it provides a lower bound that is close to the optimal solution of the original problem. The SDP relaxation is stronger than the LP relaxation hence, it provides a tighter lower bound. Moreover, if we tighten the SDP relaxation by adding violated clique inequalities, then the lower bound will improve.

In this section we will compare the LP relaxation plus separation of clique inequalities (LPMKP-C) with the SDP relaxation (SMKP) and with the SDP relaxation plus separation of clique inequalities (SMKP-C) in terms of bounds and computational efficiency. In order to compare the discussed approaches, several instances are used. These instances were randomly generated using a C code. The clique inequality separation algorithms for both SDP and LP were implemented in C. Furthermore, CPLEX through GAMS was used to solve (LPMKP-C). The CSDP solver [6] was used to solve the SDP problems with and without clique inequalities. All instances discussed were solved using a Sun Sparc.

The values of tol_c and ε_{LP} for the LP separation algorithm are given by the user. Different values of ε_{LP} are allowed. When comparing the results of different approaches in this section, $\varepsilon_{LP} = 0.1$ was used. Preliminary testing showed that $\varepsilon_{LP} = 0.1$ seems to be an effective choice. Moreover, for the SDP separation algorithm ε_{SDP} was chosen to be 0.1.

The three different approaches were applied to most of these instances and the results are provided in Table 2.1. As the problem gets larger it takes too long to get the optimal solution for the LP problem, so for 100 nodes only (SMKP) and (SMKP-C) were considered. Finally, for 200 nodes only (SMKP) was used.

From Table 2.1, (SMKP) provides a higher objective function than (LPMKP-C) and hence an improvement of the lower bound. In addition, (SMKP-C) provides the best

Instance	LPMKP-C	SMKP	SMKP-C
25_1	244.833	309.745	318.252
25_2	244.833	304.9715	316.1462
25_3	302.333	379.594	396.722
25_4	281.711	338.227	343.889
25_5	240.333	298.72	310.741
50_1	1032.5	1530.218	1555.591
50_2	962.32	1426.61	1439.26
50_3	1021.39	1514.45	1528.09
100	4107.394	6816.74	6860.581
200	-	28643.3	-

Table 2.1: Lower bounds given by (LPMKP-C), (SMKP), and (SMKP-C) for 25, 50, 100 and 200 vertices (k=3 for all instances).

Instance	LPMKP-C	SMKP	SMKP-C
25_1	0:0:5	0	0:0:9
25_2	0:0:5	0	0:0:10
25_3	0:0:6	0	0:0:16
25_4	0:0:5	0	0:0:11
25_5	0:0:5	0	0:0:9
50_1	0:1:49	0:0:20	0:546
50_2	0:1:42	0:0:20	0:8:29
50_3	0:1:46	0:0:20	0:8:31
100	1:28:01	0:17:50	4:37:36
200	-	2:22:06	-

Table 2.2: Computational time (in hr:min:sec) of (LPMKP-C), (SMKP), and (SMKP-C) for 25, 50, 100 and 200 vertices (k=3 for all instances).

bounds. Increasing the number of nodes beyond 25, we notice the significant difference in the bounds between (LPMKP-C) and (SMKP). Moreover, the improvement in the bound between (SMKP) and (SMKP-C) is worthwhile if one is willing to spend extra computational time, see Table 2.2.

We note that we didn't compare these approaches with (LPMKP) since it has clique constraints that reach up to 3 million for 100 nodes. In addition, (LPMKP) was found to be inefficient since it provides weak bounds which are the same as the bounds of (LPMKP-C). However, for (LPMKP-C) the number of constraints is significantly less since only the violated clique inequalities need to be added.

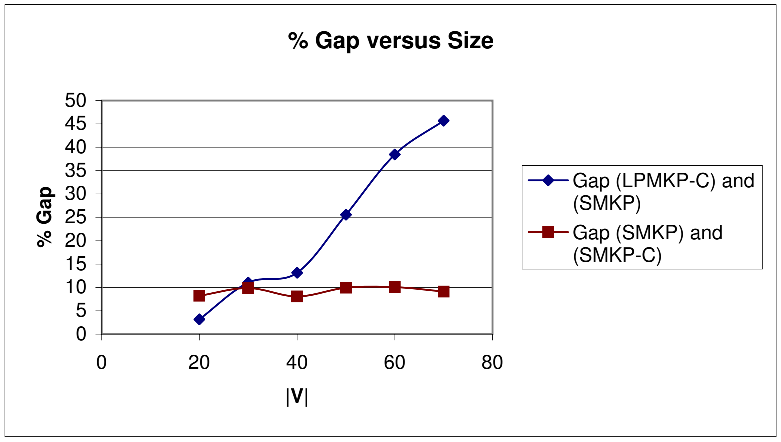

Figure 2.3: Behavior of the gaps of the different bounds.

For the SDP approach, (SMKP) provides better bounds than the LP relaxation and it is time efficient which enables it to be applied in a branch-and-bound algorithm. For (SMKP-C) only few constraints are added before none of the clique inequalities are violated. Finally, (SMKP-C) provides the best bound but it needs extra computational time; it can be applied in a branch-and-cut algorithm.

Our results suggest that the approach to adopt actually depends on the purpose. However, from Figure 2.3, we note that adding the cutting planes always improves the bound by only a fixed percentage (8 % to 10 %), whereas a greater improvement can be seen when comparing SDP bounds to LP bounds. In this thesis, we will use (SMKP) and (SMKP-C) in the branch-and-cut algorithm to solve the MkP problem.

2.4 Approximation Algorithms

An α-approximation algorithm is a polynomial-time algorithm for a combinatorial optimization problem that provides a solution of objective value that is guaranteed to be at most $(1-\alpha)$ away from the optimal value.

2.4.1 Goemans-Williamson Approximation Algorithm for Max-Cut

Goemans and Williamson [19] used semidefinite programming in the design of a randomized approximation algorithm for the max-cut problem which always produces solutions of expected value at least 0.87856 times the optimal value. This was the first time that semidefinite programming was used in an approximation algorithm.

Given an undirected graph $G = (V, E)$ and non-negative weights w_{ij} for each edge ij we can formulate the max-cut problem as in Section 1.2.1:

$$\textbf{(MC)} \qquad \max \quad \sum_{i<j} \frac{w_{ij}}{2}(1 - z_i z_j) \tag{2.29}$$

$$\text{s.t.} \quad z_i \in \{-1, 1\}. \tag{2.30}$$

Once $z_i z_j$ is replaced by $v_i \cdot v_j$, then the resulting relaxation is as follows:

$$\begin{aligned} \max \quad & \sum_{i<j} \frac{w_{ij}}{2}(1 - v_i \cdot v_j) \\ \text{s.t.} \quad & v_i \in \mathcal{S}_n. \end{aligned}$$

where $\mathcal{S}_n$ is the n-dimensional unit sphere. To solve this relaxation, [19] used semidefinite programming to formulate the relaxation in the form:

$$\textbf{(MCSDP)} \qquad \begin{array}{rl} \max & \sum_{i<j} \frac{w_{ij}}{2}(1 - X_{ij}) \\ \text{s.t.} & X_{ii} = 1 \\ & X \succeq 0. \end{array}$$

The simple randomized algorithm for the max-cut problem is as follows:

1. Solve (MCSDP) and use the Cholesky factorization $V^T V$ of the optimal X to obtain vectors v_i.
2. Generate a vector r uniformly distributed on the unit sphere $\mathcal{S}_n$.
3. Partition the vertices into 2 partitions. The first partition is where the vectors v_i have a nonnegative inner product with r and the second is those with a negative inner product with r.

Let $w(\mathcal{V})$ denote the value of the cut produced using this algorithm, and $E[w(\mathcal{V})]$ its expected value. Analysis in [19] showed that the expected weight of the cut defined by a random hyperplane rounding is:

$$E[w(\mathcal{V})] = \sum_{i<j} w_{ij} \frac{\arccos(v_i \cdot v_j)}{\pi},$$

and that

$$E[w(\mathcal{V})] \geq \alpha \frac{1}{2} \sum_{i<j} w_{ij}(1 - v_i \cdot v_j),$$

where $\alpha = \min\limits_{0 \leq \theta \leq \pi} \frac{2}{\pi} \frac{\theta}{1 - \cos\theta} > 0.878$ and θ the angle between vector v_i and the random vector r.

The results in [19] showed that the cut generated using the randomized algorithm was in the range of 4% to 9% away from the semidefinite bound. Hence, it is an effective heuristic technique for generating cuts.
In addition to the max-cut, approximation algorithms were presented in [19] for the MAX 2SAT and MAX DICUT problems.

2.4.2 Approximating Max 3-Cut Using Complex Semidefinite Programming

Toh and Trefethen showed that semidefinite programming over the complex domain can be solved in polynomial-time [43]. Goemans and Williamson in [20] proposed to take advantage of this using complex semidefinite programming. They used the 3 complex roots of unity (1, $\exp^{\frac{2\pi}{3}i}$, $\exp^{\frac{4\pi}{3}i}$) to formulate the max 3-cut, and extended the rounding technique in [19] by letting the v_i vectors be in $\mathbb{C}_n$ and r be a random vector from the n-dimensional complex distribution.

The analysis in [20] shows that their relaxation for the max 3-cut problem using complex SDP is identical to the relaxation for the max 3-cut problem in [18] and that the performance guarantee of the algorithm presented in [18] and of their algorithm is 0.836008 for the max 3-cut problem. However, an extension of this approach isn't straightforward i.e., the k roots of unity can't be used in the same way to approximate max k-cut.

2.4.3 Improved Approximation Algorithms for Max k-Cut

Frieze and Jerrum in [18] presented an extension of [19] to obtain a polynomial-time approximation algorithm for the max k-cut problem. They first start by relaxing the max k-cut using the SDP presented in Section 2.3 to obtain the following problem:

$$\textbf{(MkCSDP)} \qquad \max \frac{k-1}{k} \sum_{i<j} w_{ij}(1 - X_{ij}) \tag{2.31}$$

$$\text{s.t. } X_{ij} \geq \frac{-1}{k-1} \qquad \forall i < j \tag{2.32}$$

$$X_{ij} \succeq 0. \tag{2.33}$$

Frieze and Jerrum described a rounding heuristic based on the SDP relaxation that can be used to obtain a feasible solution of the max k-cut problem. This method works as follows:

1. Solve (MkCSDP) to get the optimal solution, $X = (X_{ij})$. Find unit vectors $v_1, \ldots, v_n \in \mathbb{R}^n$ satisfying $v_i^T v_j = X_{ij}$ where $(i, j \in V)$. This can be done by computing the Cholesky factorization $V^T V$ of X.

2. Choose k independent random vectors $r_1, \ldots, r_k \in \mathbb{R}^n$. Requiring $\| r \|= 1$ complicates the analysis. Hence, choose their kn components as independent random variables from a standard normal distribution with mean 0 and variance 1.

3. Partition V into $\mathcal{V}_k = \{V_1, \ldots, V_k\}$ according to which $r_1, \ldots r_k$ is closest to each v_i. Hence, $V_j = \{i : v_i \cdot r_j \geq v_i \cdot r_{j'},$ for $j \neq j'\}$ for $1 \leq j \leq k$

The authors in [18] proved the existence of a sequence of constants $\alpha_{(k \geq 2)}$ such that:

$$\mathbf{E}(w(\mathcal{V}_k)) \geq \alpha_k w(\mathcal{V}_k^*)$$

where $w(\mathcal{V}_k) = \sum_{1 \leq r < s \leq k} \sum_{i \in V_r, j \in V_s} w_{ij}$, $\mathcal{V}_k^*$ is the optimal cut, and $\mathbf{E}$ denotes the expected value.

Theorem 2 *α_k satisfies*

1. $\alpha_k > \frac{k-1}{k}$

2. $\alpha_k - \frac{k-1}{k} \sim \frac{2 \ln k}{k^2}$

3. $\alpha_2 \geq 0.878567 \qquad \alpha_3 \geq 0.800217 \qquad \alpha_4 \geq 0.850304 \qquad \alpha_5 \geq 0.874243$

This process can be iterated by varying the random vector r and taking the best solution (i.e., minimum upper bound). The cut obtained by this hyperplane rounding technique may be further improved by moving single vertices from one partition to the other.
Notice that α has the lowest value when $k = 3$, which is $\alpha_3 \geq 0.800217$. This suggests that the case of $k = 3$ is the most interesting, since for the hyperplane rounding has the weakest guarantee for this case.

2.5 METIS

METIS is a software used for k-way partitioning of large graphs. It is based on a multilevel partitioning algorithm that consists of three phases. In phase one the algorithm successively reduces the size of the graph, then in phase two it finds a high-quality partition of the smaller graph. Finally, in phase three it refines the partition. METIS provides *pmetis* and *kmetis* programs to partition unstructured graphs into k equal size partitions. The k-way partitioning problem has applications in many areas including parallel scientific computing, task scheduling, and VLSI design. The problem is defined as follows:

Definition 6 *Given a graph $G = (V, E)$ with $|V| = n$, partition V into k subsets such that $|V_i| = \frac{n}{k}$ and the number of edges of E whose endnodes are in different partitions is minimized.*

METIS tries to minimize the number of edges whose endnodes belong to different subsets i.e., the edge-cut of the partition. *Kmetis* is a k-way multilevel algorithm where the graph $G = (V, E)$ is first coarsened down to a small number of vertices, a k-way of this much smaller graph is computed, and then the partitioning is projected back to the original graph by periodically refining the partitioning at each intermediate level.

METIS can partition large number of graphs arising in various applications such as circuit design. The partitions produced by METIS are 10% to 50% better than those produced by spectral partitioning algorithms. Moreover, METIS is efficient in terms of speed as it can partition graphs with over 1,000,000 vertices in 256 parts in a few seconds [26]. Further information about METIS is available on the website [1].

Unlike out algorithm, METIS only considers positive edge-weights. It provides a feasible solution and hence a bound for the k-way partitioning problem where partitions are of the same size. This equilibrium-type constraint is not part of the MkP problem and hence a comparison with our algorithm is not suitable for this case. However, we conducted several test results on METIS and found it very efficient in terms of computational time. In future work we might extend our algorithm to the k-way partitioning problem and test it using benchmarks for circuit design partitioning applications.

2.6 k-way Equipartition Problem

The k-way equipartition problem is to divide the graph into k sets of vertices, each of the same size, so as to minimize the total weight of the edges that have both endnodes in the same partition. Mitchell [35] applied a branch-and-cut algorithm to the k-way equipartition problem. Mitchell applied this problem to the National Football League (NFL) alignment of teams to divisions so as to minimize the total intradivisional travel distance. Mitchell used a branch-and-cut LP-based algorithm to solve this problem. The separation algorithm for the branch-and-cut approach is given in [35]. Computational results found the optimal solution for the NFL realignment problem where $k = 8$ and $n = 32$, whereas a percentage gap of less than 2.5% was given for graphs of sizes 100 to 500.

Moreover, Lisser and Rendl [32] described a telecommunication application for the k-way equipartition problem. The edge weights represent the communication between each pair of vertices, the objective is to partition the vertices into equal size partitions so as to maximize the sum of communication within partitions. They investigated both semidefinite and linear relaxations of the problem with iterative cutting plane algorithms. They used real-world data from France Telecom to test their approach where the size of the graph ranged from 100 to 900 vertices with a density of 80%. Their test results provide lower bounds and feasible solutions and hence a measure of the quality of the bounds. These results indicate that for k=5, 10, the SDP approach produces a gap between 4%-6% from the optimal solution and is better than the LP approach. As k gets larger, the LP approach is better than the SDP approach, however in both cases the gap isn't very sensitive to the graph size.

2.7 Biq Mac

The Biq Mac Solver [39] is a binary quadratic and max-cut ($k = 2$) solver. It is similar to the SDP-based branch-and-cut framework of Helmberg and Rendl [22] that uses SDP with cutting planes as a bounding procedure. In [39], the SDP bound is tightened by including triangle inequalities. The major improvement as compared to Helmberg and

Rendl [22] is that the Biq Mac combines an interior-point method to solve the basic SDP relaxation with the bundle method to handle the triangle inequalities. At each node of the branch-and-bound a feasible solution is generated using the following algorithm:

1. Apply the Goemans and Williamson hyperplane rounding technique (described in Section 2.4.1) to the primal matrix X obtained from solving the SDP during the bundle iterations. This would give a feasible solution.

2. Apply a local search technique to improve the feasible solution $\tilde{X}$.

3. Form a convex combination of X and $\tilde{X}$ and return to Step 1.

4. Repeat as long as better feasible solutions are found.

The solver Biq Mac, available via a web interface [38], solves max-cut problems of any structure up to size of $n = 100$. It can also solve unconstrained quadratic 0-1 problems and graph bisection problems. A detailed description of Biq Mac is given in [45].

2.8 Conclusion

In this section, we presented approximation algorithms, exact solution algorithms, and other techniques for the minimum k-partition problem and related problems. In this thesis, we propose a branch-and-cut approach to obtain bounds and global optimal solutions for the minimum k-partition problem. Most methods in the literature are not direct comparators, either because they provide no guarantee of global optimality, because they set constraints on the size of the partitions, or because they are not applicable to general k. Therefore, the only approach we can compare head-to-head with is the ILP approach. However, using CPLEX for the ILP approach and comparing it with our algorithm is unfair due to the exponential number of inequalities required for the ILP formulation. CPLEX doesn't recognize the structure of the problem whereas our branch-and-cut approach exploits the structure using valid cuts.

Chapter 3

Lifting the Basic SDP Relaxation

In 2002, Lasserre [28] introduced SDP relaxations corresponding to liftings of polynomial 0-1 problems into higher dimensions. He applied the SDP liftings to quadratic 0-1 programs and max-cut problems. Lasserre presented necessary and sufficient conditions under which after a finite number of such liftings, the optimal objective value of max-cut problem is reached. Using Lagrangian duality, strengthened SDP relaxations in a higher dimension were derived in [2] for the max-cut problem. This motivates the use of liftings to obtain a tighter relaxation for the minimum k-partition problem.

3.1 Formulation

Recall the following ILP formulation of MkP discussed in Section 2.2.1:

$$\begin{aligned} \min \ & \sum_{i<j\in V} w_{ij} z_{ij} & & \\ \text{s.t. } & z_{ih} + z_{hj} - z_{ij} \leq 1 & \forall\, i < h < j \in V & \quad (3.1) \\ & \sum_{i<j\in Q} z_{ij} \geq 1 & \forall\, Q \subseteq V \text{ where } |Q| = k+1 & \quad (3.2) \\ & z_{ij} \in \{0,1\}, & & \end{aligned}$$

assuming an undirected graph and symmetric edge weights. Define the variable:

$$y_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are in the same partition,} \\ -1 & \text{if } i \text{ and } j \text{ are in different partitions,} \end{cases}$$

and observe that the transformation

$$y_{ij} = 2z_{ij} - 1$$

maps the two sets of variables in a way that preserves their meaning.

We now consider what happens to the constraints. The inequality (3.1) stays the same:

$$\begin{aligned} z_{ih} + z_{hj} - z_{ij} \leq 1 \quad & \Leftrightarrow \tfrac{y_{ih}+1}{2} + \tfrac{y_{hj}+1}{2} - \tfrac{y_{ij}+1}{2} \leq 1 \\ & \Leftrightarrow y_{ih} + y_{hj} - y_{ij} + 1 \leq 2 \\ & \Leftrightarrow y_{ih} + y_{hj} - y_{ij} \leq 1 \end{aligned}$$

However, the inequality (3.2) becomes:

$$\begin{aligned} \textstyle\sum_{i<j\in Q} z_{ij} \geq 1 \quad & \Leftrightarrow \textstyle\sum_{i<j\in Q} \left(\frac{y_{ij}+1}{2}\right) \geq 1 \\ & \Leftrightarrow \textstyle\sum_{i<j\in Q} (y_{ij}+1) \geq 2 \\ & \Leftrightarrow \textstyle\sum_{i<j\in Q} y_{ij} \geq 2 - \binom{|Q|}{2}. \end{aligned}$$

For the case $k = 3$, $|Q| = 4$ and the inequality is $\sum_{i<j\in Q} y_{ij} \geq -4$.

The formulation is as follows:

$$\begin{aligned} \min \quad & \sum_{i<j\in V} w_{ij} y_{ij} & & \\ \text{s.t.} \quad & y_{ih} + y_{hj} - y_{ij} \leq 1 & \forall\, i < h < j \in V & \qquad (3.3) \\ & \sum_{i<j\in Q} y_{ij} \geq 2 - \binom{|Q|}{2} & \forall\, Q \subseteq V \text{ where } |Q| = k+1 & \qquad (3.4) \\ & y_{ij} \in \{-1, 1\}, & & \end{aligned}$$

To simplify notation, let $y_{ij} = a, y_{ih} = b$, and $y_{hj} = c$. If $c = 1$ then a and b should have the same value. That is, if vertices h and j are in the same partition ($c = 1$) then either

i will be in the same partition of h and j ($b = a = 1$) or it will not ($b = a = -1$). There is $3\binom{n}{3}$ of these transitivity conditions for any value of i,j, and h, where n is the number of vertices of a graph. For each three vertices we get the three transitivity conditions as follows:

$$(b-a)(c+1) = 0 \Rightarrow bc - ac + b - a = 0 \tag{3.5}$$

$$(a-c)(b+1) = 0 \Rightarrow ab - bc + a - c = 0 \tag{3.6}$$

$$(b-c)(a+1) = 0 \Rightarrow ab - ac + b - c = 0 \tag{3.7}$$

Moreover, for each three vertices out of the total eight combinations of $\{1, -1\}$ only five are valid. Next we elaborate these five valid combinations that result from the following four cases:

1. If vertices i and j are in the same partition (a=1) and vertices i and h are in the same partition ($b = 1$) then vertices j and h should be in the same partition (c=1).

2. If vertices i and j are in the same partition (a=1) and vertices i and h are in different partitions ($b = -1$) then vertices j and h should be in different partitions (c=-1).

3. If vertices i and j are in different partitions(a=-1) and vertices i and h are in different partitions ($b = -1$) then vertices j and h could be in the same partition or in different partitions (c=1 or $c = -1$).

4. If vertices i and j are in different partitions (a=-1) and vertices i and h are in the same partition ($b = 1$) then vertices j and h should be in different partitions (c=-1).

The above cases are summarized in the table below:

a	b	c	abc
1	1	1	1
1	-1	-1	1
-1	-1	± 1	± 1
-1	1	-1	1

From the table above, we notice that $abc = -1$ only if $a = b = c = -1$.
Therefore,

$$\begin{aligned} &(abc-1)(a+b+c+3) = 0 \\ &\Rightarrow 3abc + ab + ac + bc - a - b - c - 3 = 0 \end{aligned} \tag{3.8}$$

Substituting $abc = 1 - ab + c$ in equation (3.8) we get:
$bc - ac - 2ab - a - b + 2c = 0$ which is equivalent to $-2*$ equation (3.6)+ equation (3.7).
Moreover, we can get equation (3.8) by multiplying equation (3.5) by c, equation (3.6) by c, equation (3.7) by b and then adding them as follows:
$abc - a + bc - 1 = 0$
$abc - b + ac - 1 = 0$
$abc - c + ab - 1 = 0$
Adding the above three equations we get $3abc+ab+ac+bc-a-b-c-3 = 0$ which is equation (3.8). So using equations (3.5)-(3.7) is enough to ensure that equation (3.8) is satisfied.

In the case of 3-partition, we have a four-vertices clique. Consider four vertices i, j, k, and h, if any three of these vertices are in different partitions, then the fourth vertex should be in the same partition of one of these three vertices. This can be written as follows:
If $y_{ik} = -1, y_{ij} = -1$ and $y_{kj} = -1$
$\Rightarrow y_{ih}$ or y_{kh} or $y_{jh} = 1$
$\Rightarrow y_{ih} + y_{jh} + y_{kh} = -1$
Therefore, $(y_{ij} - 1)(y_{ik} - 1)(y_{jk} - 1)(y_{ih} + y_{jh} + y_{kh} + 1) = 0$
$\Rightarrow (y_{ij}y_{ik}y_{jk} - y_{ij}y_{jk} - y_{ij}y_{ik} - y_{jk}y_{ik} + y_{ik} + y_{jk} + y_{ij} - 1)(y_{ih} + y_{jh} + y_{kh} + 1) = 0$
$\Rightarrow y_{ij}y_{ik}y_{jk}y_{ih} - y_{ij}y_{jk}y_{ih} - y_{ij}y_{ik}y_{ih} - y_{jk}y_{ik}y_{ih} + y_{ik}y_{ih} + y_{jk}y_{ih} + y_{ij}y_{ih} - y_{ih}$
$+y_{ij}y_{ik}y_{jk}y_{jh} - y_{ij}y_{jk}y_{jh} - y_{ij}y_{ik}y_{jh} - y_{jk}y_{ik}y_{jh} + y_{ik}y_{jh} + y_{jk}y_{jh} + y_{ij}y_{jh} - y_{jh}$
$+y_{ij}y_{ik}y_{jk}y_{kh} - y_{ij}y_{jk}y_{kh} - y_{ij}y_{ik}y_{kh} - y_{jk}y_{ik}y_{kh} + y_{ik}y_{kh} + y_{jk}y_{kh} + y_{ij}y_{kh} - y_{kh}$
$+y_{ij}y_{ik}y_{jk} - y_{ij}y_{jk} - y_{ij}y_{ik} - y_{jk}y_{ik} + y_{ik} + y_{jk} + y_{ij} - 1 = 0$

We get a similar analysis when we do the permutation:
$(y_{ij} - 1)(y_{ih} - 1)(y_{jh} - 1)(y_{ik} + y_{jk} + y_{kh} + 1) = 0$
$(y_{ik} - 1)(y_{ih} - 1)(y_{kh} - 1)(y_{ij} + y_{jk} + y_{jh} + 1) = 0$

$(y_{jk}-1)(y_{jh}-1)(y_{kh}-1)(y_{ij}+y_{ik}+y_{ih}+1)=0$
Therefore, we have a total of four clique equalities. However, using these equalities is hard due to the presence of the monomials with four variables. We need to have a bigger SDP matrix to include such monomials and this will increase the computational time significantly. Instead, we use clique inequalities of the form (3.4).

In the next section, we will use these equalities (3.5)-(3.7) to form the SDP relaxation. However, we don't go to higher degrees, that is, we don't use monomials of degree three or higher.

3.2 Strengthened SDP relaxation via Lifting

We will show that the original triangle inequality is weaker than the ones provided in equation (3.5)-(3.7). Recall the triangle inequality and its permutations

$$y_{ih}+y_{hj}-y_{ij}\le 1 \qquad \Rightarrow b+c-a\le 1 \tag{3.9}$$
$$y_{ij}+y_{hj}-y_{ih}\le 1 \qquad \Rightarrow a+c-b\le 1 \tag{3.10}$$
$$y_{ij}+y_{ih}-y_{hj}\le 1 \qquad \Rightarrow a+b-c\le 1 \tag{3.11}$$

where $y_{ij}=a, y_{ih}=b$, and $y_{hj}=c$.

Lemma 4 *If y_{ij}, y_{ih}, and y_{hj} satisfy equations* (3.5)-(3.7)*, then they satisfy equations* (3.9)-(3.11)*.*

Proof.
Equation (3.5), can be written as $(b-a)(c+1)=0$. In case $c\neq -1$, then $b=a$. Substituting $b=a$ in equation (3.9), we get $c\le 1$ which is satisfied since $-1\le c\le 1$.
In case $c=-1$, then $b+c-a=b-a-1\le 1$ since $-1\le b\le 1$ and $-1\le a\le 1$.
Similarly, we can show that if a point satisfies equation (3.6) and equation (3.7) then it satisfies equation (3.10) and (3.11) respectively.
Moreover, we can show by counter example that there exists a point satisfying equation (3.9) but not equation (3.5). Take $a=-\frac{1}{2}, b=1$, and $c=\frac{1}{2}$. Substituting in equation (3.9), we have $1+\frac{1}{2}-\frac{1}{2}\le 1$ and hence the equation is satisfied. However, if we substitute

$a = -\frac{1}{2}, b = 1$, and $c = \frac{1}{2}$ in equation (3.5), we have $\frac{1}{2} - \frac{1}{4} + 1 - \frac{1}{2} = \frac{3}{4} \neq 0$.
Hence, every discrete point satisfying equation (3.5) satisfies equation (3.9) but not vice versa and similarly for the other constraints. $\square$

Let the matrix $X = x_{ij}$ containing the pairs and the triplets. In case we have 3 vertices then X will have the following form:

$$X = \begin{pmatrix} 1 & y_{12} & y_{13} & y_{23} & y_{12}y_{13} & y_{12}y_{23} & y_{13}y_{23} & y_{12}y_{13}y_{23} \\ & 1 & y_{12}y_{13} & y_{12}y_{23} & \vdots & \vdots & \vdots & y_{13}y_{23} \\ & & 1 & y_{13}y_{23} & \vdots & \vdots & \vdots & y_{12}y_{23} \\ & & & \ddots & \vdots & \vdots & \vdots & \vdots \\ & & & & \dots & \dots & \dots & 1 \end{pmatrix}$$

Notice that $X_{15} = X_{23}$ is one example of the many equalities that arise from the structure of this matrix. For n vertices, X has dimension $\binom{\binom{n}{2}}{3} + \binom{\binom{n}{2}}{2} + \binom{n}{2} + 1$ and if we remove the columns of triple products from the matrix, X will have a dimension of $\binom{\binom{n}{2}}{2} + \binom{n}{2} + 1$. Moreover, if we consider the equalities in the structure of the matrix we can reduce the size to be of dimension $\binom{n}{2} + 1$ as follows:

$$X = \begin{pmatrix} 1 & y_{12} & y_{13} & \dots & y_{n-1,n} \\ & 1 & \vdots & \vdots & \vdots \\ & & \ddots & \vdots & y_{ij}y_{kh} \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix}_{(\binom{n}{2}+1)\times(\binom{n}{2}+1)}$$

The X_{ij} entries can be obtained using the following transformation:

$$X_{1,T(i,j)} = y_{ij},$$
$$X_{T(i,j),T(k,h)} = y_{ij}y_{kh},$$

where $T(i,j) = \frac{(j-1)(j-2)}{2} + i + 1$.

Relaxing the $\{-1, 1\}$ constraints, the SDP formulation becomes as follows:

$$\textbf{(LSDP)} \qquad \begin{aligned} \min \ & \sum_{i,j} w_{ij} \frac{(1+X_{ij})}{2} \\ \text{s.t. } & X_{ij} - X_{jh} + X_{1i} - X_{1h} = 0 && \forall\, h,i,j && (3.12) \\ & X_{ij} - X_{ih} + X_{1j} - X_{1h} = 0 && \forall\, h,i,j && (3.13) \\ & X_{jh} - X_{ih} + X_{1j} - X_{1i} = 0 && \forall\, h,i,j && (3.14) \\ & X_{ii} = 1 \\ & X \succeq 0. \end{aligned}$$

The triangle equalities are from equation (3.5)-(3.7). However, adding equations (3.12) and (3.14) will result in equation (3.13). Taking any two of the equalities we can get the third one. Hence, we can remove any one of them without loss of information.

$$\begin{aligned} \min \ & \sum_{i,j} w_{ij} \frac{(1+X_{ij})}{2} \\ \text{s.t. } & X_{ij} - X_{jh} + X_{1i} - X_{1h} = 0 && \forall\, h,i,j \\ & X_{jh} - X_{ih} + X_{1j} - X_{1i} = 0 && \forall\, h,i,j \\ & X_{ii} = 1 \\ & X \succeq 0. \end{aligned}$$

Moreover, using equations (3.9)-(3.11) we have a set of valid inequalities for (LSDP). These inequalities have the form:

$$\begin{aligned} & X_{1i} + X_{1j} - X_{1h} \leq 1 && \forall\, h,i,j && (3.15) \\ & X_{1i} + X_{1h} - X_{1j} \leq 1 && \forall\, h,i,j && (3.16) \\ & X_{1h} + X_{1j} - X_{1i} \leq 1 && \forall\, h,i,j && (3.17) \end{aligned}$$

When $X_{ij} \in \{-1, 1\}$, then if vertices i, j, and h satisfy equations (3.12)-(3.14), then they satisfy equations (3.15)-(3.17). However, when X_{ij} entries are fractional this might not be the case.

Since solving (LSDP) without the clique inequalities will give the trivial identity matrix solution, we choose to add clique inequalities. The clique inequalities can be added by including clique inequalities of the form $\sum X_{1i} \geq -4$ for the case of $k = 3$. The separation algorithm for the clique inequalities is similar to the LP separation algorithm presented in Algorithm 1.

In the next section, we compare the performance of the basic SDP relaxation and the (LSDP) in terms of quality of bounds and the computational efficiency.

3.3 Comparison between Basic SDP Relaxation and the Lifted SDP

In order to compare the previously discussed SDP relaxations, different instances were randomly generated using a C code. The CSDP solver [6] was used to solve the SDP problems. All instances discussed were solved on the same Sun workstation with 16 processors and 32 GB of RAM.

From Tables 3.1-3.3, we notice that the lower bound of (LSDP) is in most cases better than the lower bound of (SMKP) especially as the number of vertices of the graph increases. The % gap in most cases is between 1% and 4%. However, the computational time required by (LSDP) is significantly larger than that of (SMKP) and this is due to the size of the SDP matrix. The SDP matrix in (SMKP) case has a size of $n \times n$ whereas for (LSDP) the SDP matrix has a size of $[\binom{n}{2} + 1] \times [\binom{n}{2} + 1]$. Due to the inefficiency of (LSDP) in terms of time, we will not be able to use it in a branch-and-cut algorithm since it will significantly degrade the algorithm's performance. Although it is a promising relaxation in terms of the improved bounds, the computational time required by (LSDP) is too large for a branch-and-cut algorithm. It would be necessary to solve the SDP much more efficiently to be able to exploit this relaxation.

Instance	(LSDP)		(SMKP)		%Gap
	LB	Time	LB	Time	
10_1	33.95407	0	34.02139	0	**-0.19787**
10_2	43.57347	0	42.15544	0	**3.363829**
20_1	189.3691	0:8:10	191.6418	0	**-1.1859**
20_2	188.3403	0:7:17	186.366	0	**1.059356**
20_3	172.4323	0:7:54	176.0377	0	**-2.04808**
20_4	196.1249	0:8:10	209.0209	0	**-6.1697**
25_1	346.9252	0:44:48	339.5227	0	**2.180272**
25_2	336.0117	0:47:17	328.4746	0	**2.294595**
25_3	326.5639	0:44:43	320.514	0	**1.887537**
30_1	490.1492	3:40:33	477.9304	0:0:1	**2.556621**
30_2	506.5425	3:42:03	494.6722	0:0:1	**2.399627**
30_3	489.5041	3:42:02	473.7856	0:0:1	**3.317644**

Table 3.1: Lower bound comparison between the lifted and the basic SDP relaxation for 10, 20, 25 and 30 vertices. k=3 and time is in hr:min:sec.

Instance	(LSDP)		(SMKP)		%Gap
	LB	Time	LB	Time	
20_1	112.3795	0:10:08	109.2322	0	**2.881256**
20_2	106	0:9:04	103.6107	0	**2.306036**
20_3	116.8102	0:9:50	114.6116	0	**1.918226**
20_4	120.6657	0:10:30	118.6468	0	**1.701623**
25_1	199.0892	0:48:01	192.4265	0:0:1	**3.462454**
25_2	205.0574	0:50:3	197.9614	0:0:1	**3.584557**
25_3	191.806	0:45:12	188.8182	0:0:1	**1.582385**

Table 3.2: Lower bound comparison between the lifted and the basic SDP relaxation for 20 and 25 vertices. k=4 and time is in hr:min:sec.

Instance	(LSDP)		(SMKP)		%Gap
	LB	Time	LB	Time	
20_1	68.26464	0:9:12	65.95797	0	**3.497183**
20_2	69.63217	0:9:19	69.83121	0	**-0.28503**
20_3	71.97329	0:10:41	72.51428	0	**-0.74605**
25_1	134.2568	0:41:24	128.9805	0:0:1	**4.090726**
25_2	136.919	0:44:51	131.9623	0:0:1	**3.756165**
25_3	122.9496	0:41:02	120.9309	0:0:1	**1.669301**

Table 3.3: Lower bound comparison between the lifted and the basic SDP relaxation for 20 and 25 vertices. k=5 and time is in hr:min:sec.

Chapter 4

A Semidefinite Programming-Based Branch-and-Cut Framework (SBC)

4.1 Brief Overview of Branch-and-Cut

The branch-and-cut method consists of the combination of the cutting-plane method and the branch-and-bound algorithm. Traditionally, branch-and-cut solves a sequence of LP relaxations of the ILP problem at each node of the branch-and-bound tree. Cutting-plane methods improve the LP relaxation at the nodes, tightening the bounds. The branch-and-bound algorithm guarantees that a global optimal solution is obtained.

As explained in Section 2.3.6, the SDP relaxation provides a significantly tighter bound than the LP relaxation for the (ILPMKP) problem. Hence, instead of solving a sequence of LPs at each node of the tree, we will solve a sequence of SDP relaxations. In addition, we add valid cuts to improve the SDP relaxations. This can be done either at the root node only or at every node of the tree, as desired.

The branch-and-cut tree starts with a root node corresponding to the original SDP relaxation of the problem with none of the variables fixed. After one or more rounds of adding valid inequalities at the root node and resolving, if a discrete solution is found then we terminate. Otherwise, the branch-and-cut search algorithm branches by creating two subproblems derived by fixing a fractional variable (i.e., a variable that is not 1 nor $\frac{-1}{k-1}$). The subproblems are identical to the parent subproblem except for a variable being fixed to 1 or $\frac{-1}{k-1}$. The branch-and-cut tree stops when all subproblems have a discrete optimal solution or have been fathomed. A subproblem is fathomed if it is either infeasible, has a discrete solution, or if we can conclude that it does not contain an optimum solution. The incumbent solution is the best discrete solution (giving an upper bound, since we are minimizing) found so far in the tree.

An algorithmic description of branch-and-cut using SDP to solve the MkP problem is:

Step 1: Initialization Form the root node by using the (SMKP) problem without fixing any variables.

Step 2: Terminating If none of the nodes of the tree are active (i.e., all nodes are fathomed) then terminate with the incumbent solution, $X_{incumbent}$, as the optimal solution and the corresponding objective value ν^* as the optimal objective value. Else proceed to Step 3.

Step 3: Solving Choose a node t not yet solved. Solve the SDP relaxation (SMKP) to get a solution X_t^* and a lower bound ω_t.

Step 4: Adding Valid Inequalities Separate violated clique and triangle inequalities as discussed in Algorithm 2. If none are violated go to Step 5, otherwise go to Step 4.

Step 5: Obtaining a Feasible Solution Get a feasible solution $X_{feasible_t}$ as discussed in Section 4.2 and an upper bound ν_t. Update the incumbent if $\nu_t < \nu^*$.

Step 6: Fathoming

1. By Solving: If the solution X_t^* has all entries integer, i.e., X_t^* and $X_{feasible_t}$ are identical. Update the incumbent if $\nu_t < \nu^*$. Increment t and go to Step 2.
2. By Bound: If the SDP relaxation gives $\omega_t \geq \nu^*$, then branching on this node will not improve the incumbent. Increment t and go to Step 2.

3. By Infeasibility: If the SDP relaxation doesn't have a feasible solution. Increment t and go to Step 2.

Step 7: Branching Choose a variable that is fractional (i.e., not 1 or $\frac{-1}{k-1}$) and create two new nodes by fixing the variable to 1 for one node and $\frac{-1}{k-1}$ for the other node. Increment t and go to Step 2.

In the following sections, we describe in detail a branch-and-cut technique using SDP as the bounding procedure. The addition of triangle and clique inequalities at each node markedly improves the SDP lower bound, and hence pays off in spite of the computational costs. Typically, branching is done on an edge. Hence, either both endnodes are put in the same partition or in separate partitions. Moreover, at each node a feasible solution is computed to get an upper bound.

4.2 ICH: An Iterative Clustering SDP-based Heuristic

The novel ICH heuristic is designed to find a feasible solution at each node of the tree. Given a graph $G(V, E)$ with n vertices, weights w_{ij} between edges, and partition size k the heuristic proceeds as follows:

1. Initialize a parameter r, the current number of partitions, to zero.

2. Initialize a parameter m, the current number of nodes, to n.

3. Solve the SDP relaxation (SMKP) with m nodes and get the optimal solution X^*.

4. Take each triplet of vertices i, j, and h and sum their edges: $T_{ijh} = X_{ij} + X_{ih} + X_{jh}$.

5. Sort the values of T_{ijh}.

6. (a) Choose vertices i, j, and h with $T_{ijh} > tol$ to be in the same partition.

   (b) If any vertices remain unassigned to a partition, choose vertices with $T_{ijh} < tol$ to be in separate partitions.

   (c) Modify the value of r as the number of partitions change.

7. If $r > k$,
   (a) Aggregate the vertices that are in the same partition to form one new vertex i'.
   (b) Update the value of m and the aggregate weight matrix $\bar{W}$.
   (c) Return to step 3.
8. End.

The intuition behind this approach is the use of aggregate information which is more reliable than single elements of data. When we sum the edges of the three vertices, we have a better idea of whether or not these three vertices should be in the same partition than by looking at each edge separately. The sorting of the data is done to take advantage of the best information first and use the less certain information only if necessary.

Note that sorting is not done for all the $\binom{n}{3}$ combinations since that is quite time consuming. Instead, sorting is done first for a range of high values then for smaller values as needed. This speeds up the algorithm significantly.

We use an example to elaborate on the algorithm, see Figure 4.1. Consider a complete graph with 20 vertices and suppose we want to divide these vertices into 3 partitions. The ICH heuristic proceeds as follows. First initialize r to be zero and m to be 20. We solve the (SMKP) problem and get the solution matrix

$$X^* = \begin{pmatrix} 1 & X_{12} & X_{13} & \dots & X_{1n} \\ & 1 & \vdots & \vdots & \vdots \\ & & \ddots & \vdots & X_{ij} \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix}.$$

Next use the $X_{ij} \to \frac{(k-1)X_{ij}+1}{k}$ mapping to convert the entries of X^* from $[\frac{-1}{k-1}, 1]$ to $[0, 1]$. Take each triplet of vertices i, j, and h and sum their edges to get T_{ijh}. Since we mapped each X_{ij} to [0,1], T_{ijh} will have a value between 0 and 3. After getting the T_{ijh} values, we sort them. In this example, *tol* was set to 1.6. Table 4.1 provides the highest T_{ijh} values for our example.

i	j	h	T_{ijh}
1	12	15	2.989602
2	10	11	2.982662
2	9	11	2.969226
12	15	18	2.966918
4	12	15	2.966367
4	18	20	2.961048
9	10	11	2.960813
1	12	18	2.959404
⋮	⋮	⋮	⋮

Table 4.1: Iteration 1 of the ICH heuristic example with $n = 20$ and $k = 3$.

Using the data in Table 4.1, we put vertices 1, 12, and 15 in one partition and increment r by one. Then we put vertices 2, 10, and 11 in another partition and again increment r. In the next row of the table we have vertices 2, 9, and 11 however vertex 2 and 11 are already assigned to a partition so we put vertex 9 in the same partition without incrementing r. By the time we reach row 7 where $i = 9$, $j = 10$, and $h = 11$, we don't have any change in the partitions. This column will only confirm the previous allocation of vertices 9, 10, and 11. We continue in this way until all vertices are assigned to a partition. If $r > k$ when we are done, then we perform another iteration of the algorithm, otherwise we stop with a feasible solution.

In our example at the first iteration we get $r = 5$ as follows:

$$\begin{cases} \text{Partition} & \text{Vertices} \\ 1: & 1, 4, 12, 15, 18, 20 \\ 2: & 2, 9, 10, 11 \\ 3: & 3, 7, 19 \\ 4: & 5, 13, 14 \\ 5: & 6, 8, 16, 17 \end{cases}$$

Since $r = 5 > 3$, we perform another iteration. We aggregate the vertices that are in the same partition to form one vertex and we set $m = 5$. Therefore, we now have five vertices, and the edge weights between them are adjusted. To illustrate how the edge weights are adjusted, we show how to calculate the edge weight between the aggregated vertices 3 and 4, i.e., $\bar{w}_{34}$. Vertex 3 is the aggregation of vertices 3, 7, and 19, and vertex 4 is the aggregation of vertices 5, 13, and 14. Therefore,

$$\begin{aligned} \bar{w}_{3,4} = {} & w_{3,7} + w_{3,19} + w_{7,19} + w_{5,13} + w_{5,14} \\ & + w_{13,14} + w_{3,5} + w_{3,13} + w_{3,14} + w_{5,7} \\ & + w_{7,13} + w_{7,14} + w_{5,19} + w_{13,19} + w_{14,19}. \end{aligned}$$

Similarly, we calculate the edge weights for the other vertices.

Next we set $r = 0$ and we solve the SDP relaxation to get X^*. After performing the same steps as before, we obtain the T_{ijh} shown in Table 4.2.

From Table 4.2, we can deduce that vertices 1, 2, and 3 are in separate partitions so we update $r = 3$. Moreover, vertices 2 and 5 are put in the same partition, and vertices 3 and 4 are also in the same partition. Since $r = k$, we stop the algorithm with the following feasible solution:

$$\begin{cases} \text{Partition} & \text{Vertices} \\ 1: & 1, 4, 12, 15, 18, 20 \\ 2: & 2, 9, 10, 11, 6, 8, 16, 17 \\ 3: & 3, 7, 19, 5, 13, 14 \end{cases}$$

i	j	h	T_{ijh}
1	2	5	1.00
1	3	4	1.00
2	3	5	1.00
2	4	5	1.00
3	4	5	1.00
2	3	4	1.00
⋮	⋮	⋮	⋮
1	2	3	0
1	3	5	0
1	2	4	0
1	4	5	0

Table 4.2: Iteration 2 of the ICH heuristic example with $n = 20$ and $k = 3$.

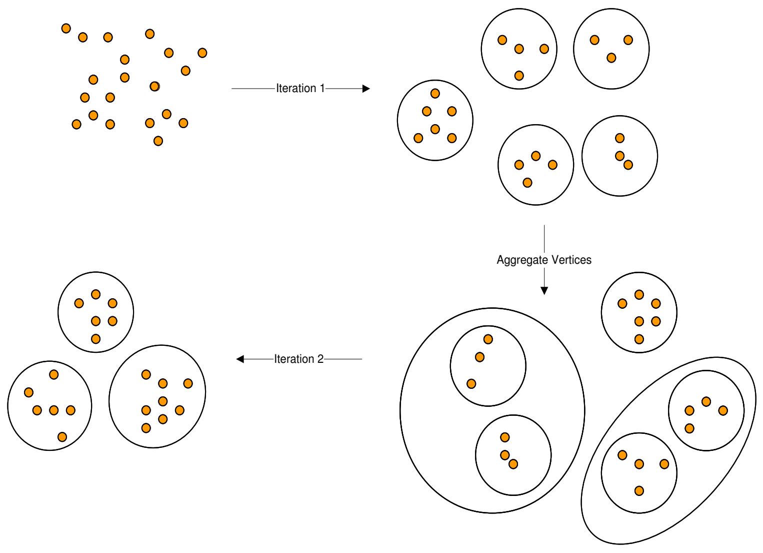

Figure 4.1: The ICH heuristic example with $n = 20$ and $k = 3$.

4.2.1 The ICH Heuristic with Convex Combination

The convex combination technique to improve on the Goemans-Williamson hyperplane rounding was proposed and implemented for $k = 2$ in [45]. Using this convex combination technique results in a better solution than using only hyperplane rounding. This motivated us to apply the convex combination idea to the Frieze and Jerrum [18] algorithm.

Given the SDP solution matrix X_1^* and the hyperplane rounding feasible solution matrix $X_1^{feasible}$, we take their convex combination to obtain the following matrix:

$$X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}$$

Next we take matrix X_2 and we can perform the hyperplane rounding technique on this matrix to get a new feasible solution.

Similarly, we applied the convex combination technique to the ICH heuristic. Taking the feasible solution matrix $X_1^{feasible}$ obtained from the ICH heuristic and the SDP solution matrix, X_1^*, we consider a convex combination of the following form:

$$X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}.$$

Then we can apply the ICH heuristic to the X_2 matrix to get a new feasible solution, $X_2^{feasible}$. However, we found that the new feasible solution $X_2^{feasible}$ is identical to $X_1^{feasible}$. This result is not too surprising since multiplying X_1^* by α only scales the values of X_{ij} and will not change their sorted order. In addition, since we got $X_1^{feasible}$ from X_1^*, they most likely have vertices i, j, and h with the same sorted order. Once we multiply $X_1^{feasible}$ by $(1-\alpha)$ then this will only scale the values but will not change their sorted order. We have, $X_2 = \alpha X_1^* + (1-\alpha) X_1^{feasible}$ so adding the edges values, X_{ij}, of the three vertices using the matrix X_2 will give the same result as when we add the edges of the three vertices using the matrix X_1 since the order of T_{ijh} values in the sorting will likely remain the same (with a difference in the value since it is scaled and shifted). This was the case in all our computational experiments.

Hence, the convex combination technique does not seem to improve the solution for the ICH heuristic. This gives evidence that the heuristic is strong enough that it does not benefit from performing the convex combination improvement technique.

4.2.2 Comparison of Hyperplane Rounding and ICH

In this section, the two techniques to find a feasible solution for the MkP problem are compared. We implemented both algorithms, ICH and the hyperplane rounding presented in [18] using C and MATLAB respectively.

Since the rounding presented by [18] is randomized, each time we run the algorithm we get a different feasible solution. As a result, this algorithm was run 30 times and the minimum and the average of the upper bound (UB) were computed. The average value can be interpreted as an estimate of the expected value of the UB that this algorithm would give. On the other hand, the minimum value is the best solution found over the 30 runs.

The hyperplane rounding with the convex combination technique provides different upper bounds for different values of α as seen in Figure 4.2-4.4. The plots shown in Figure 4.2 are for different values of $|V|$ where we have a complete graph and the edge weights are random with values between 0 and 9. For small values of $|V|$, the upper bound remains almost constant as α varies. As we increase $|V|$ beyond 50 vertices, then we notice that the average upper bound provided by the hyperplane rounding decreases as α increases. However, this can't be generalized for all instances since the upper bound variation with α is unclear for certain groups of instances in particular for spinglass2g instances where we have positive and negative edge-weights, see Figure 4.3.

The two techniques were tested on the following three types of graphs for $k = 2$ and for $k = 3$:

- Random Instances: These instances consist of complete graphs where the edge weights are randomly generated between 0 and 9 using a C code.
- Spinglass2g Instances: These instances consist of graphs that were generated using

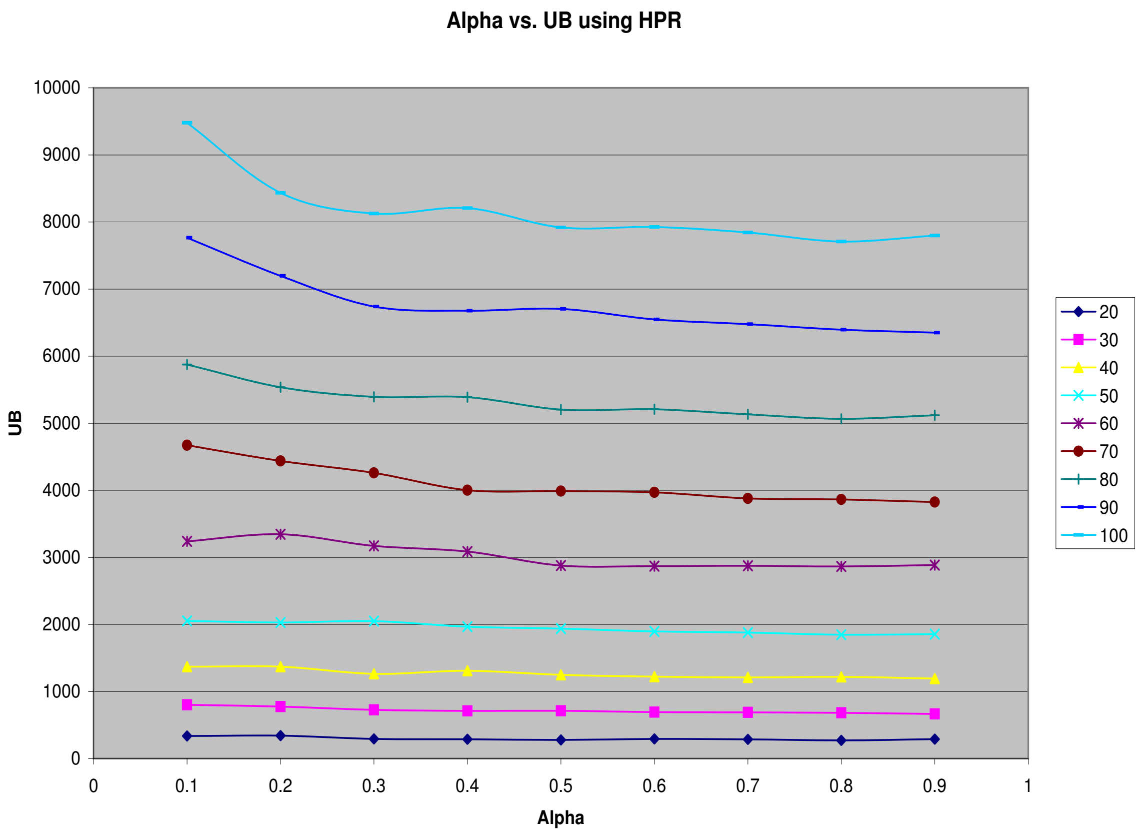

Figure 4.2: The average value of the UB provided by the hyperplane rounding [18] with convex combination versus α for different values of $|V|$ for random instances.

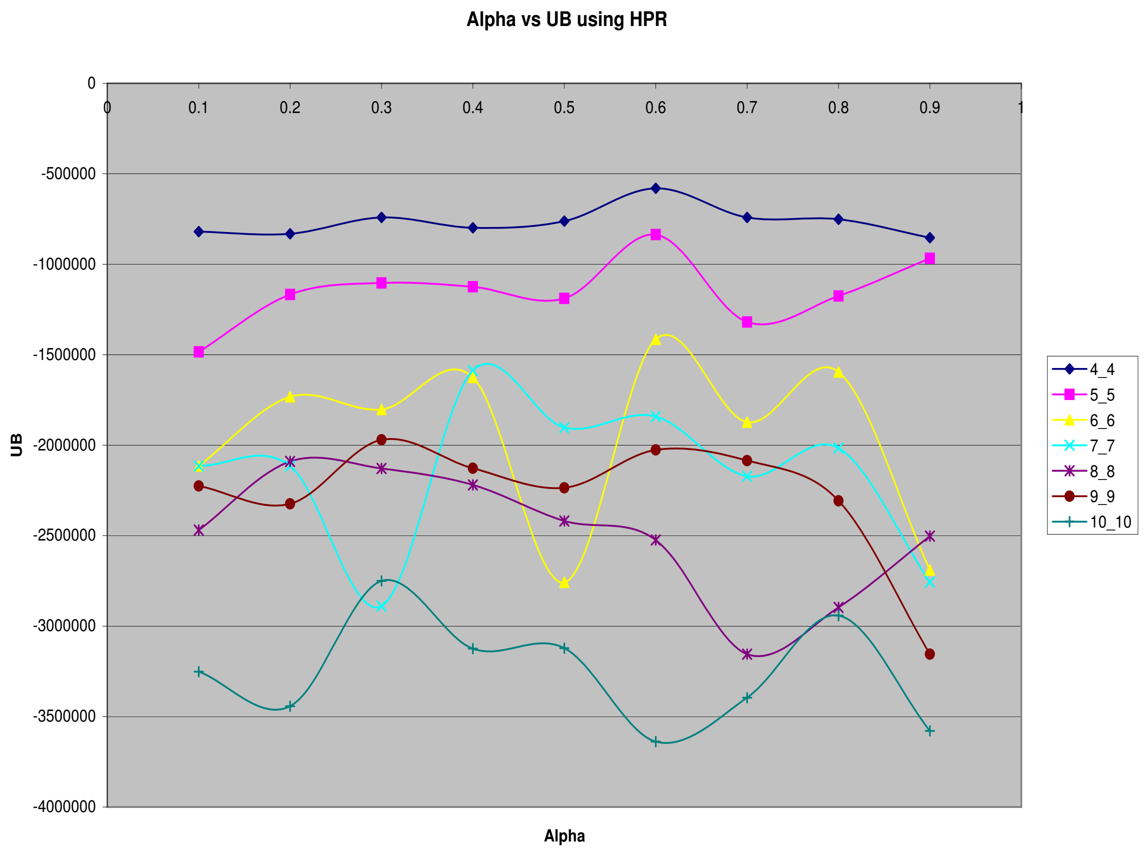

Figure 4.3: The average value of the UB provided by the hyperplane rounding [18] with convex combination versus α for different values of $|V|$ for spinglass2g instances.

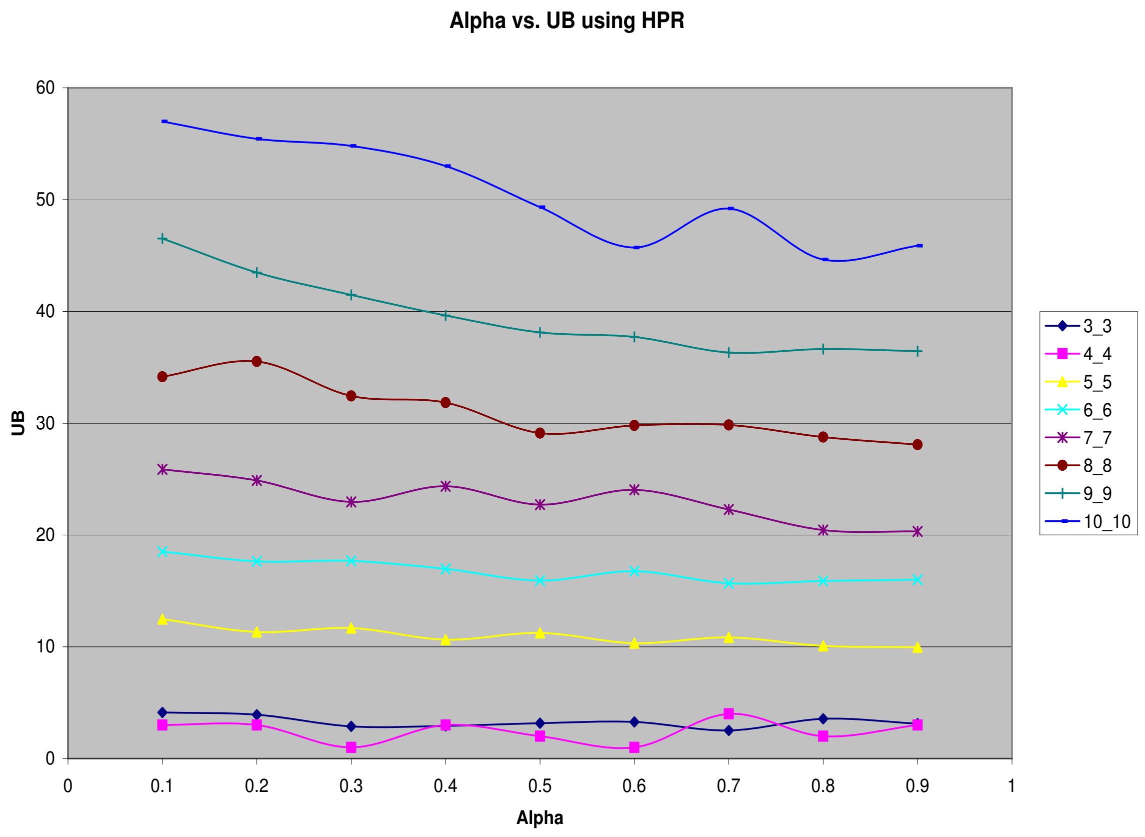

Figure 4.4: The average value of the UB provided by the hyperplane rounding [18] with convex combination versus α for different values of $|V|$ for grid_2D instances.

rudy graph generator [40]. Spinglass2g generates a toroidal 2D grid for a spin glass model with gaussian interactions.

- Grid_2D Instances: These instances consist of graphs that were generated using rudy graph generator [40]. Grid_2D generates a planar bidimensional grid with edge weights all equal to 1.

$\lvert V \rvert$	ICH	[18] (min)	[18] (avg.)	LB	UB (min) for [18] with convex combination for a given α								
					0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
20	**188**	217	283.62	187.3	215	258	**188**	241	216	207	227	225	231
30	557	589	670.41	493.7	614	588	611	623	598	605	580	592	558
40	992	1088	1170.9	925.5	1117	1135	1108	1094	1130	1077	1101	1096	1089
50	1656	1694	1837.6	1497.1	1752	1735	1725	1704	1766	1761	1757	1706	1737
60	2548	2724	2856.3	2351.9	2749	2809	2739	2774	2716	2722	2649	2692	2705
70	3477	3679	3858.3	3223.4	3815	3708	3774	3706	3789	3676	3685	3615	3664
80	4508	4848	5098.4	4293.5	4892	4888	4790	4809	4909	4857	4877	4892	4815
90	5774	6132	6325.1	5420.1	6249	6134	6084	6054	6263	6117	6151	6139	6098
100	6973	7491	7770.8	6634.2	7566	7661	7529	7534	7602	7561	7549	7496	7433

Table 4.3: Computational Results for random instances with $k = 3$.

$\lvert V \rvert$	ICH	[18] (min)	[18] (avg.)	UB (min) for [18] with convex combination for a given α								
				0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
4×4	-954077	-954077	-295100	-819312	-831392	-741231	-798278	-761526	-580161	-741298	-751103	-852946
5×5	-1367840	-1185097	-302620	-1484348	-1166946	-1103329	-1124676	-1188754	-836275	-1319361	-1175218	-966957
6×6	**-2758520**	-2147425	-740870	-2115524	-1732624	-1802507	-1625819	-2758520	-1414849	-1872757	-1595561	-2690359
7×7	**-3282586**	-2115560	-670710	-2115560	-2115560	-2889403	-1587528	-1902404	-1841520	-2171880	-2016232	-2756529
8×8	**-4063059**	-2705506	-1308600	-2469005	-2090016	-2128793	-2219785	-2419073	-2523733	-3154943	-2896465	-2502696
9×9	**-4758332**	-2247374	-1265700	-2225260	-2324296	-1970217	-2127385	-2235664	-2026423	-2085498	-2307414	-3155256
10×10	-6570984	-3150645	-1903600	-3251798	-3442696	-2750327	-3124199	-3122204	-3638336	-3395681	-2941674	-3579241

Table 4.4: Computational Results for spinglass2g with $k = 3$.

				UB (min) for [18] with convex combination for a given α								
$\|V\|$	ICH	[18] (min)	[18] (avg.)	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
3×3	**0**	1	2.88	2	**0**	1	**0**	**0**	1	**0**	1	**0**
4×4	**0**	2	6.04	3	3	1	3	2	1	4	2	3
5×5	**0**	4	9.32	7	7	4	4	1	5	6	4	4
6×6	**0**	7	15.44	12	8	12	8	12	7	8	7	9
7×7	**0**	14	20.72	15	16	18	14	13	17	13	13	13
8×8	**0**	17	26.68	16	20	20	24	20	17	19	17	20
9×9	**0**	22	34.24	32	29	23	24	25	25	24	26	21
10×10	**0**	36	45.88	39	34	33	39	37	35	39	35	33

Table 4.5: Computational Results for grid_2D with $k = 3$. Numbers in bold indicate that the heuristic solution is the optimal solution.

					UB (min) for [18] with convex combination for a given α								
$\|V\|$	ICH	[18] (min)	[18] (avg.)	LB	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
30	**912**	**912**	1005.8	912	**912**	**912**	**912**	921	927	941	925	**912**	**912**
40	**1691**	1728	1852.3	1690.463	1728	1742	1773	**1691**	**1691**	1716	1719	**1691**	**1691**
50	2729	2858	2974.9	2716.069	2848	2767	2823	2857	2815	2787	2810	2802	2855
60	4001	4151	4326.3	3985.364	4054	4151	4128	4106	4172	4196	4112	4095	4159
70	5401	5608	5864.6	5384.104	5667	5625	5452	5581	5654	5520	5501	5584	5557
80	7098	7389	7583.2	7032.764	7389	7382	7211	7321	7363	7381	7281	7341	7230
90	9292	9830.2	9551	9190.776	9551	9572	9595	9480	9599	9508	9450	9592	9568
100	11496	11747	12094	11382.92	11784	11747	11881	11854	11878	11871	11878	11936	11860

Table 4.6: Computational Results for random instances of max-cut (k=2). Numbers in bold indicate that the heuristic solution is the optimal solution.

$\|V\|$	ICH	[18] (min)	[18] (avg.)	LB	UB (min) for [18] with convex combination for a given α								
					0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
3 × 3	**-795504**	**-795504**	-308690	-795504	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**	**-795504**
4 × 4	-592202	-592202	193780	-592433.75	-592202	-592202	-592202	-592202	-592202	-592202	-592202	-592202	-536551
5 × 5	**-1543707**	-1451470	-215890	**-1543707**	**-1543707**	-1182988	**-1543707**	**-1543707**	-1461391	**-1543707**	-1278662	**-1543707**	**-1543707**
6 × 6	**-2846691**	**-2846691**	-1125300	-2846691	**-2846691**	-2554116	**-2846691**	**-2846691**	-2680268	**-2846691**	**-2846691**	-2718202	**-2846691**
7 × 7	**-3020297**	-2818053	-1085400	-3020297	-2787849	-2787849	-2818053	**-3020297**	**-3020297**	**-3020297**	-2807370	**-3020297**	**-3020297**
8 × 8	**-4489989**	**-4489989**	-1344700	-4489989	-4396002	**-4489989**	-4252115	**-4489989**	-4271661	-3702040	**-4489989**	**-4489989**	-4005560
9 × 9	**-6230102**	-5522456	-2538400	-6230102	**-6230102**	-5950570	-5858896	-5522456	-5644327	-5953242	-5213005	**-6230102**	-6153144
10 × 10	**-7872968**	**-7872968**	-2764400	-7872968	**-7872968**	-7760364	**-7872968**	-6869239	**-7872968**	-6480148	-7042388	**-7872968**	-7540716

Table 4.7: Computational Results of max-cut (k=2) for spinglass2g instances. Numbers in bold indicate that the heuristic solution is the optimal solution.

$\|V\|$	ICH	[18] (min)	[18] (avg.)	UB (min) for [18] with convex combination for a given α								
				0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
3	**0**	**0**	3.8	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
4	**0**	**0**	8.52	**0**	**0**	**0**	**0**	**0**	**0**	**0**	3	**0**
5	**0**	**0**	15.44	2	**0**	2	**0**	**0**	**0**	6	6	**0**
6	**0**	**0**	21.56	**0**	**0**	**0**	**0**	3	**0**	2	**0**	2
7	**0**	4	28.32	4	4	3	3	**0**	**0**	**0**	**0**	3
8	**0**	**0**	40.8	**0**	4	20	**0**	4	**0**	**0**	13	**0**
9	**0**	**0**	41.4	3	**0**	10	2	**0**	8	**0**	**0**	12
10	**0**	**0**	49.6	**0**	**0**	10	**0**	3	7	**0**	4	**0**

Table 4.8: Computational Results of max-cut (k=2) for grid_2D instances. Numbers in bold indicate that the heuristic solution is the optimal solution.

$\|V\|$	ICH	LB	%Gap
4 × 4	-954077	-954107.5	0.003
5 × 5	-1367840	-1484348	8.517
6 × 6	-2758520	-2758520	0
7 × 7	-3282586	-3282586	0
8 × 8	-4063059	-4063059	0
9 × 9	-4758332	-5236178	10.042
10 × 10	-6570984	-7230203	10.032

Table 4.9: Comparing the UB provided by ICH and the LB at the root node of the branch-and-cut tree using spinglass2g instances and $k = 3$.

From Tables 4.3-4.8, we can notice that ICH is always better than the expected value of the hyperplane rounding, and is in most cases at least as good as its minimum. Moreover, even using different values of α for the hyperplane rounding with convex combination, the results are still not as good as those of the ICH heuristic. Therefore, the UBs provided by ICH are generally tighter and using it at each node of the branch-and-cut algorithm will help reduce the size of the tree.

From Table 4.4 we can see that for spinglass2g instances where we have positive and negative weights, the ICH heuristic provides a better solution than minimum value of hyperplane rounding for all values of α. This shows that ICH is efficient in the presence of negative weights unlike the hyperplane rounding.

Moreover, from Table 4.9 we can see that the ICH heuristic provides a tight bound at the root node and sometimes can directly find the optimal solution. This would help the branch-and-cut tree not to grow exponentially and likely speed the computational time in many cases.

We note that for the grid_2D instances we can find a solution by inspection. For the case $k = 2$ there is a unique solution, while for $k = 3$ we have multiple solutions. Moreover, for $k = 2$ the SDP matrix X^* satisfies $X_{ij} \in \{-1, 1\}$ while for $k = 3$ the SDP matrix X^* doesn't have its entries $X_{ij} \in \{-\frac{1}{2}, 1\}$ but the matrix is a convex combination of several multiple solutions. We included the results for grid_2D instances to show that even if we don't have the SDP matrix with discrete entries, the ICH heuristic can still extract a feasible solution that was found to be optimal for all test cases tried. So, unlike the hyperplane rounding heuristic, the ICH heuristic can find the optimal solution for both cases, even though the matrix entries aren't of the form $\{\frac{-1}{k-1}, 1\}$.

In this section, we experimentally showed the strength of the ICH heuristic. This justifies its use in the branch-and-cut algorithm to provide a feasible solution at each node.

4.3 Branching Rules

The success of a branch-and-bound algorithm depends on the choice of the edge to branch on. Helmberg and Rendl in [22] presented five different branching rules referred to R1-R5.

R1 and R2 follow the easiest first strategy, that is, to branch first on a pair of vertices $\{i, j\}$, where the decision whether nodes i and j are in the same partition at the optimum or not seems to be obvious. In the case of $k = 2$, R1 selects edge ij with maximum $|X_{ij}|$ value while R2 chooses i and j so that they minimize $\sum_{l=1}^{n}(1 - |X_{il}|)^2$.

Rules R3 and R4 go for the opposite policy, namely branching first where the decision seems to be hard. When $k = 2$, R3 selects edge ij with minimum $|X_{ij}|$ value. R4 chooses edge ij with vertex i that minimizes $\sum_{l=1}^{n}(1-|X_{il}|)^2$ and vertex j that minimizes $\sum_{l=1}^{n} X_{jl}^2$. Finally, in rule R5, information given by the triangle inequalities is used.

Based in the results of the analysis done in [22], we decided to use a version of R3 as a branching rule in our branch-and-cut implementation. Our rule R3 works as follows:

Select the edge ij with X_{ij} farthest from 1 and $\frac{-1}{k-1}$, i.e., branch on the edge ij that minimizes $|\frac{2X_{ij}(k-1)-k+2}{k}|$.

For example, when $k = 3$ we will branch on the edge that has a value farthest from 1 and $-\frac{1}{2}$. So if we were given the following values of X_{ij}:

$$X^* = \begin{pmatrix} 1 & 0.524 & -0.56 & 0.1996 \\ & 1 & 0.911 & -0.34 \\ & & 1 & 0.257 \\ & & & 1 \end{pmatrix},$$

then we choose to branch on X_{34} since it has the value farthest from 1 and $-\frac{1}{2}$. So on one branch we set X_{34} to have a value of 1 and on the other branch to a value of $-\frac{1}{2}$.
By branching on the most difficult decision X_{ij}, we hope that the bound will improve faster. If this is the case then the branch-and-cut tree will be more balanced, as seen for example in Figure 4.5, where $|V| = 30$ and $k = 3$.

4.4 Conclusion

In this chapter we described our branch-and-cut SDP-based algorithm. To obtain a feasible solution, we use the ICH heuristic without the convex combination technique, since it had a better performance than the hyperplane rounding and the convex combination did not show any improvement of the solution. The branching strategy used is discussed in Section 4.3. Furthermore, we added the option of tightening the SDP by triangle and clique inequalities at the root node or at each node of the tree. The flow chart of the SDP-based branch-and-cut algorithm (SBC) is shown in Figure 4.6.

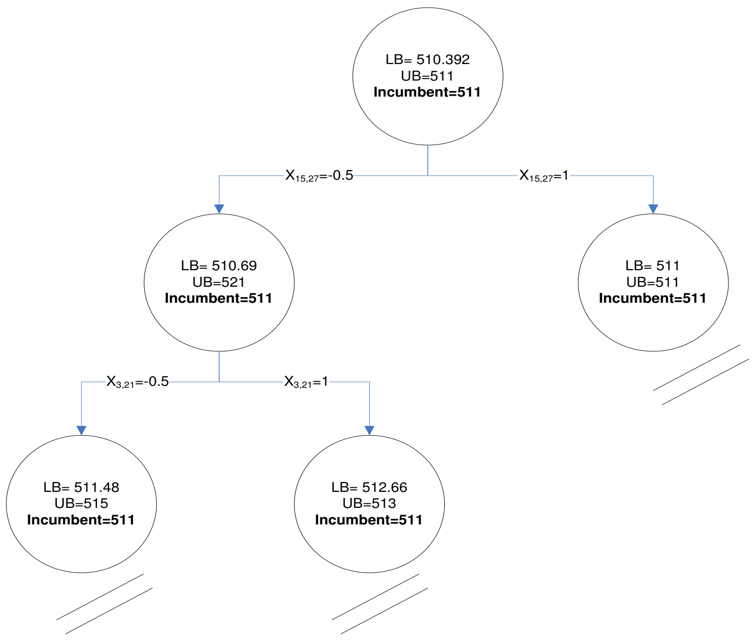

Figure 4.5: A branch-and-cut example for $|V| = 30$ and k=3.

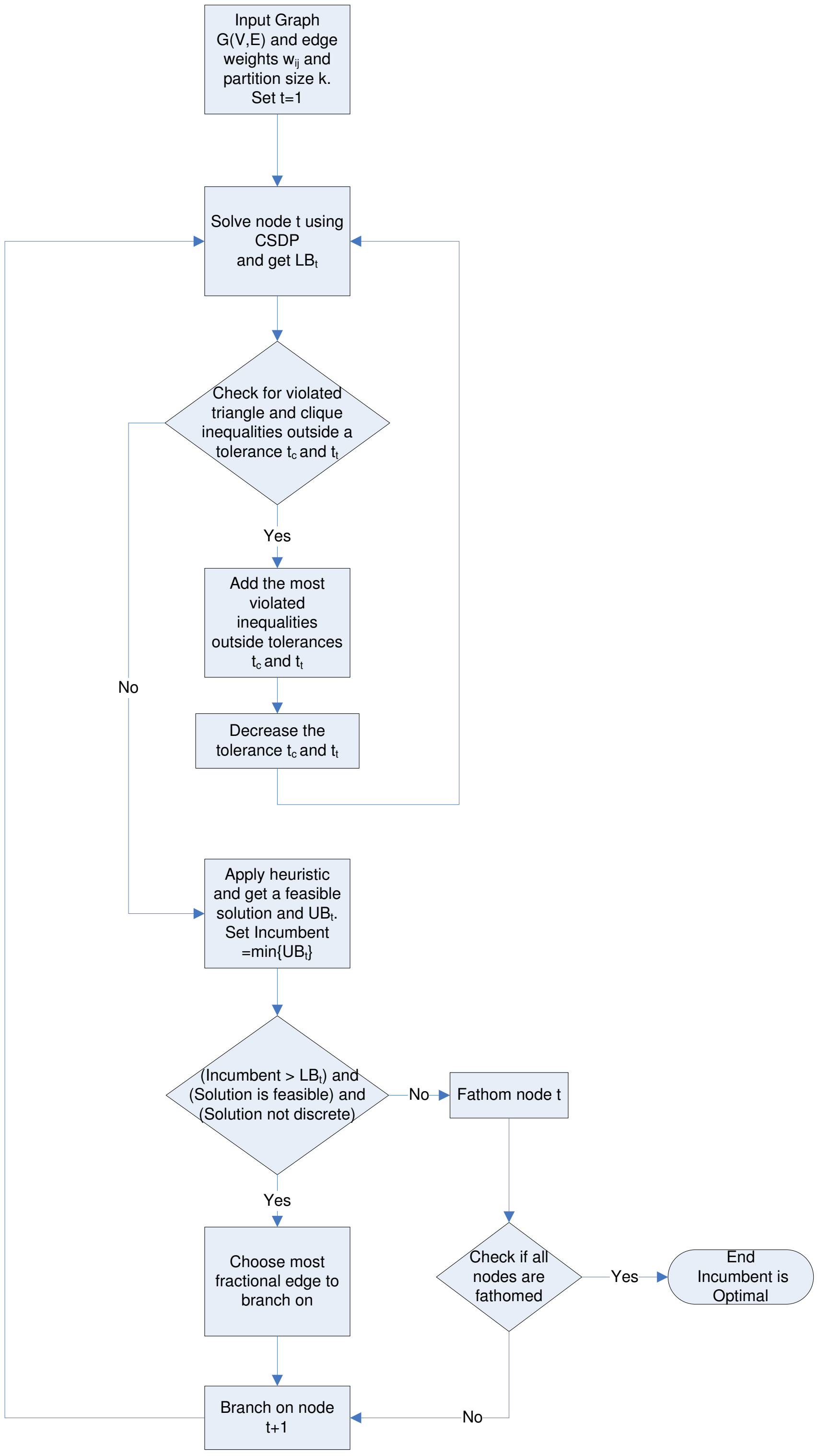
Input Graph G(V,E) and edge weights w_{ij} and partition size k. Set t=1
Solve node t using CSDP and get LB_t
Check for violated triangle and clique inequalities outside a tolerance t_c and t_t
Yes
No
Add the most violated inequalities outside tolerances t_c and t_t
Decrease the tolerance t_c and t_t
Apply heuristic and get a feasible solution and UB_t. Set Incumbent $=\min\{UB_t\}$
(Incumbent > LB_t) and (Solution is feasible) and (Solution not discrete)
No
Fathom node t
Yes
Choose most fractional edge to branch on
Check if all nodes are fathomed
Yes
End Incumbent is Optimal
No
Branch on node t+1

Figure 4.6: Flow chart of the SBC algorithm.

Chapter 5

Computational Experiments with the New Framework

5.1 Computational Results

We implemented in C the branch-and-cut SDP-based Algorithm (SBC) described in the previous chapter. To solve the SDP, which has to be done at each node of the tree, we used the CSDP solver [6]. The SDP relaxations were solved from scratch at each node of the tree since it is not straight forward to do warm start with SDP by exploiting information generated at parent nodes. The SBC algorithm was tested using several instances on a Sun workstation with 16 processors and 32 GB of RAM.

5.1.1 Test Instances

The first group of instances consists of complete graphs (100% edge density) with random edge weights. All graphs in this group were generated by using a C code. We generated several instances each of the following type of graph:

- Random: Complete graphs with integer edge weights chosen randomly from [0, 9].

The second group of instances consists of graphs generated using the graph generator of Rinaldi [40], rudy. We generated several instances for each of the following types of graphs:

$\|V\|$	Time	Optimal Solution	$\|V\|$	Time	Optimal Solution
15	0:0:08	86	30	1:27:01	531
15	0:0:17	104	30	0:44:10	530
15	0:0:21	109	30	3:27:29	523
20	0:0:33	207	35	7:16:41	721
20	0:1:35	236	35	8:14:38	720
20	0:7:51	201	35	8:38:49	685
25	0:1:11	348	40	7:10:14	920
25	0:4:25	338	40	4:36:01	903
25	0:2:29	322	40	7:50:48	951

Table 5.1: SBC results for random instances where k=3. The time is given in hr:min:sec.

- Grid_2D: described in Section 4.2.2.
- Toroidal_grid_2D: generates toroidal bidimensional grid with edge weights equal to 1. The grid has size n=(height×width).
- Spinglass2G: described in Section 4.2.2.
- Spinglass3G: generates a toroidal 3D grid for a spin glass model with Gaussian distributed $\pm J$ interactions. The grid has size n =(rows×columns×layers).
- Spinglass2pm: generates a toroidal 2D-grid for a spin glass model with ± 1 interactions. The grid has size n =(rows×columns). The percentage of negative interactions in this case is taken 50%.
- Spinglass3pm: generates a toroidal 3D-grid for a spin glass model with ± 1 interactions. The grid has size n =(rows×columns×layers). The percentage of negative interactions in this case is taken 50%.

We note that for instances that have integer edge weights the optimal objective value will always have an integer value. Therefore, if upper and lower bounds in a node are such that $|$ LB-UB $|< 1$, we can fathom the node. If the global bounds differ by at most 1, then we stop the algorithm with an optimal solution. Tables 5.1-5.8 show the computational results for the SBC algorithm for k=3. Table 5.1 presents the time and the optimal solution for random type instances with $|V|$ being the number of vertices of the graph. Table 5.2 presents results for grid_2D and toroidal_grid_2D type of instances where the number of vertices is given by height$\times$width. Tables 5.3-5.8 give results for two-dimensional and three-dimensional spin glass instances. In addition to the optimal solution, the lower bound and the upper bound at the root node as well as the time at the root node is presented. Moreover, the number of nodes of the branch-and-cut tree as well as the time to reach a certain percentage gap are given in the tables. The symbol $\curvearrowleft$ denotes that a lower % gap was achieved directly. For Table 5.3, we give optimal solution for sizes up to 100 vertices (10$\times$10) and provide a feasible solution for larger sizes (up to 169 vertices) with a percentage gap less than 6%.

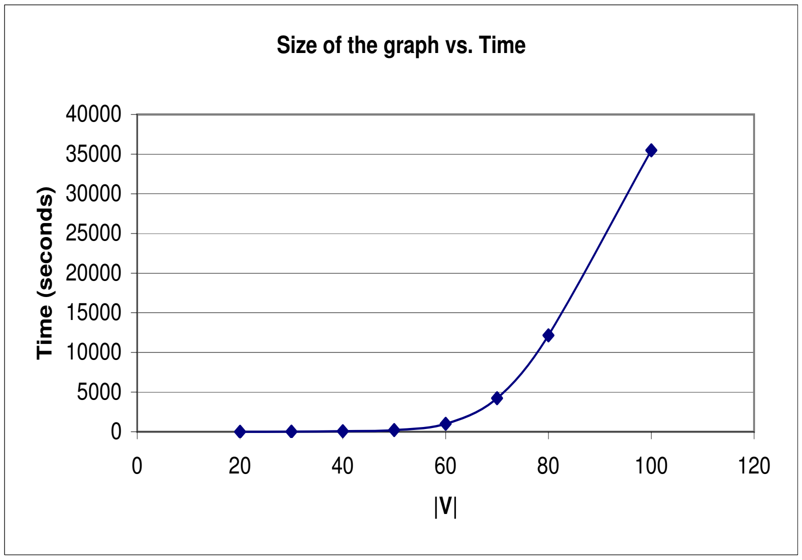

Figure 5.1: Size of the graph versus computational time.

In order to compare Biq Mac and SBC, we need to choose $k = 2$ since Biq Mac is designed specifically for max-cut only. The performance is compared in terms of time since both algorithms provide the global optimal solution. The results are given in Table 5.7.

	$\lvert V\rvert$	Optimal solution	Time	Number of Nodes
Grid_2D	3 × 3	0	0:0:01	1
	4 × 4	0	0:0:2	1
	5 × 5	0	0:0:11	1
	6 × 6	0	0:0:49	1
	7 × 7	0	0:02:37	1
	8 × 8	0	0:12:54	1
	9 × 9	0	0:43:18	1
	10 × 10	0	1:58:22	1
Toroidal_grid_2D	4 × 4	0:0:0	7	1
	5 × 5	0	0:0:16	1
	6 × 6	0	0:0:43	1
	7 × 7	0	0:04:09	1
	8 × 8	0	0:12:35	1

Table 5.2: SBC results for grid_2D and toroidal_grid_2D instances where k=3. The time is given in hr:min:sec and the number of branched nodes of the branch-and-cut tree is given in the last column.

$\lvert V \rvert$	Best Solution Value	Root Node			Number of Nodes - Time to achieve % Gap				
		LB	UB	Time	0%	1%	2%	5%	10%
3×3	-639287	-639287	-639287	0:0:19	1 - 0:0:19	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
3×3	-367196	-367196	-367196	0:0:7	1 - 0:0:7	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
3×3	-449795	-449795	-449795	0:0:5	1 - 0:0:5	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
4×4	-1162476	-1162476	-1162476	0:0:22	1 - 0:0:22	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
4×4	-954107	-955381	-954107	0:00:18	3 - 0:0:29	1 - 0:0:18	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
4×4	-954077	-954077	-954077	0:0:16	1 - 0:0:16	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
5×5	-1367840	-1367840	-1367840	0:0:24	1 - 0:0:24	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
5×5	-2321578	-2321578	-2321578	0:0:20	1 - 0:0:20	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
5×5	-1484348	-1484722	-1484348	0:00:18	2 - 0:0:23	1 - 0:0:18	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
6×6	-2634775	-2634775	-2634775	1:32:34	1 - 1:32:34	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
6×6	-2758520	-2758520	-2758520	1:10:01	1 - 1:10:01	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
6×6	-2432958	-2432958	-2432958	1:14:19	1 - 1:14:19	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
7×7	-3282586	-3282586	-3282586	2:52:12	1 - 2:52:12	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
7×7	-3885568	-3885568	-3885568	3:45:09	1 - 3:45:09	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
7×7	-3282435	-3282435	-3282435	2:55:31	1 - 2:55:31	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
8×8	-4011207	-4011207	-4011207	9:18:43	1 - 9:18:43	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
8×8	-4951974	-4951974	-4951974	14:01:55	1 - 14:01:55	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
9×9	-4758332	-4806178	-4758332	3:35:49	4 - 13:41:17	1 - 3:35:49	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
10×10	-6570984	-6570984	-6630202.5	10:36:23	15 - 22:50:46	4 - 13:42:58	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
11×11	-8586382	-8586382	-9015701.1	5:48:50	-	-	-	5:48:50	$\curvearrowleft$
12×12	-10646782	-10646782	-11189768	9:31:00	-	-	-	9:31:00	$\curvearrowleft$
13×13	-11618406	-11618406	-12292274	29:33:27	-	-	-	-	29:33:27
14×14	-13780370	-13780370	-14607192	47:16:57	-	-	-	-	47:16:57

Table 5.3: SBC results for spinglass2g instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

$\lvert V \rvert$	Optimal Solution Value	Root Node			Number of Nodes - Time to achieve % Gap				
		LB	UB	Time	0%	1%	2%	5%	10%
$2 \times 3 \times 4$	-2197030	-2197030	-2197030	0:03:59	1 - 0:03:59	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 3 \times 4$	-1952753	-1952753	-1952753	0:27:29	1 - 0:27:29	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 3 \times 5$	-2026448	-2035907	-1996572	2:08:40	3 - 2:15:29	1 - 2:08:40	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 3 \times 5$	-2273897	-2273897	-2273897	0:23:06	1 - 0:23:06	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 4 \times 5$	-3396942	-3418285	-3396942	2:05:51	6 - 3:08:42	1 - 2:05:51	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 4 \times 5$	-2653512	-2658471	-2653512	2:01:43	2 - 2:35:45	1 - 2:01:43	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 3$	-2145971	-2145971	-2145971	0:14:04	1 - 0:14:04	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 3$	-2194622	-2194622	-2194622	0:12:03	1 - 0:12:03	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 4$	-3410696	-3415728	-3410696	2:07:21	5 - 7:31:14	1 - 2:07:21	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 4$	-3192317	-3192317	-3192317	0:26:52	1 - 0:26:52	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 5$	-4204246	-4209348	-4204246	2:52:31	5 - 3:38:37	2 - 3:03:52	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 5$	-4560275	-4576778	-4556179	3:08:35	4 - 5:01:04	1 - 3:08:35	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 4 \times 4$	-4293925	-4296250	-4250811	5:54:56	3 - 9:56:09	1 - 5:54:56	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 4 \times 4$	-4262046	-4262046	-4262046	5:55:59	1 - 5:55:59	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 4 \times 5$	-5053641	-5323788	-5049424	6:02:52	13 - 27:03:07	10 - 25:29:03	7 - 16:02:13	4 - 10:32:10	$\curvearrowleft$
$3 \times 4 \times 5$	-5278612	-5683601	-5323788	12:54:36	22 - 30:13:51	6 - 20:43:03	$\curvearrowleft$	5 - 14:19:53	$\curvearrowleft$
$4 \times 4 \times 4$	-6809386	-6917561	-6682223	13:10:04	15 - 17:56:21	3 - 15:24:38	$\curvearrowleft$	1 - 13:10:04	$\curvearrowleft$
$4 \times 4 \times 4$	-7474525	-7529318	-7474525	9:03:37	3 - 10:12:11	1 - 9:03:37	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$

Table 5.4: SBC results for spinglass3g instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

	Optimal Solution	Root Node			Number of Nodes - Time to achieve % Gap				
$\lvert V\rvert$	Value	LB	UB	Time	0%	1%	2%	5%	10%
5×5	-21	-21	-21	0:22:39	1 - 0:22:39	↶	↶	↶	↶
5×5	-25	-25	-25	0:26:29	1 - 0:26:29	↶	↶	↶	↶
5×5	-25	-25	-25	0:21:41	1 - 0:21:41	↶	↶	↶	↶
6×6	-29	-29	-29	1:46:26	1 - 1:46:26	↶	↶	↶	↶
6×6	-30	-30	-30	0:35:19	1 - 0:53:51	↶	↶	↶	↶
6×6	-28	-28	-27	0:31:13	3 - 0:48:36	↶	↶	1 - 0:31:13	↶
7×7	-42	-42	-42	8:42:45	4 - 11:22:23	1 - 8:42:45	↶	↶	↶
7×7	-40	-40	-40	8:16:01	3 - 10:51:50	2 - 10 :11:23	↶	↶	↶
7×7	-42	-42	-42	11:20:22	1 - 11:20:22	↶	↶	↶	↶
8×8	-55	-55	-55	13:57:16	1 - 13:57:16	↶	↶	↶	↶
8×8	-53	-53	-52	8:03:38	3 - 14:45:09	↶	1- 8:03:38	↶	↶
9×9	-63	-64	-59	7:36:17	122 - 20:28:17	57 - 15:03:37	51- 13:25:51	32 - 10:29:53	1- 7:36:17
9×9	-65	-66	-63	7:18:03	37 - 14:50:14	31 - 13:29:16	23 - 11:41:32	4 - 5:31:19	↶

Table 5.5: SBC results for spinglass2pm instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

	Optimal Solution	Root Node			Number of Nodes - Time to achieve % Gap				
$\|V\|$	Value	LB	UB	Time	0%	1%	2%	5%	10%
$2 \times 3 \times 4$	-20	-20	-20	0:19:42	1 - 0:19:42	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 3 \times 4$	-22	-22	-22	0:14:03	1 - 0:14:03	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$2 \times 4 \times 4$	-27	-27	-26	0:20:14	3 - 0:32:08	$\curvearrowleft$	$\curvearrowleft$	1 - 0:20:14	$\curvearrowleft$
$2 \times 4 \times 4$	-31	-31	-31	0:50:21	1 - 0:50:21	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 3$	-27	-27	-27	0:21:01	1 - 0:21:01	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 3$	-26	-26	-26	0:05:43	1 - 0:05:43	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 4$	-37	-37	-37	1:23:15	1 - 1:23:15	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 3 \times 4$	-35	-35	-35	1:23:34	1 - 1:23:34	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 4 \times 4$	-47	-49	-47	2:40:22	3 - 4:45:35	$\curvearrowleft$	$\curvearrowleft$	1 - 2:40:22	$\curvearrowleft$
$3 \times 4 \times 4$	-48	-48	-48	2:58:15	1 - 2:58:15	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$3 \times 4 \times 5$	-65	-66	-62	4:38:12	16 - 8:55:33	10 - 7:09:22	7 - 6:04:19	1 - 4:38:12	$\curvearrowleft$
$3 \times 4 \times 5$	-63	-63	-61	5:51:31	22 - 35:37:47	13 - 15:30:26	10 - 13:41:07	1 - 5:51:31	$\curvearrowleft$
$4 \times 4 \times 4$	-65	-66	-63	4:09:38	114 - 14:19:40	87 - 10:56:11	83- 10:15:54	10 - 5:02:52	$\curvearrowleft$
$4 \times 4 \times 4$	-65	-65	-64	4:32:18	69 - 8:36:15	52 - 7:38:33	1 - 4:32:18	$\curvearrowleft$	$\curvearrowleft$

Table 5.6: SBC results for spinglass3pm instances where $k = 3$. The time is given in hr:min:sec. The last five columns are the number of nodes of the tree and the time required to reach the given gap.

$\lvert V\rvert$	SBC	Biq Mac
10	3	0.01
15	9	0.04
20	32	0.1
25	41	0.12
30	417	2
35	213	3
40	814	6.4

Table 5.7: Computational time (in seconds) of SBC versus Biq Mac.

We next show the computational results when comparing SBC with the ILP model (ILPMKP) implemented in GAMS and using CPLEX solver. We set a time limit of 1000 seconds for both approaches. CPLEX can solve up to 20 vertices within a reasonable time. For 20 vertices it stops with a percentage gap between 15%-18% since the time limit was exceeded. Using CPLEX, the percentage gap varied between 25% and 36% for $\lvert V\rvert$ =25. On the other hand, SBC can solve these graphs to optimality within few minutes. For $\lvert V\rvert > 25$ it would take CPLEX more than a day to find a solution. This is expected since we have an exponential number of inequalities for ILP formulation and CPLEX doesn't know the structure of the problem.

		$k = 5$		$k = 7$	
	$\|V\|$	Objective Value	Time	Objective Value	Time
spinglass2pm	5 × 5	-21	0:07:48	-21	430
	6 × 6	-28	0:25:48	-28	1212
	7 × 7	-42	0:44:04	-42	10382
	8 × 8	-55	5:32:14	-55	21410
	9 × 9	-69	12:53:57	-69	47550
spinglass3pm	2 × 3 × 4	-21	0:07:27	-21	221
	2 × 3 × 5	-28	0:12:10	-29	3144
	2 × 4 × 4	-28	0:19:36	-28	975
	3 × 3 × 3	-26	0:37:53	-26	1645
	3 × 3 × 4	-37	0:49:23	-37	2676
	3 × 4 × 4	-48	4:10:31	-50	10256
	3 × 4 × 5	-65	5:30:21	-66	17412
	4 × 4 × 4	-68	13:49:23	-70	42670
spinglass2g	6 × 6	-2738870	0:23:41	-2738870	21
	7 × 7	-3843979	0:42:31	-3864156	2363
	8 × 8	-5485579	4:03:17	-5541348	7985
	9 × 9	-5745419	11:22:31	-6026024	31169
	10 × 10	-6860706	19:14:02	-7644016	63149
spinglass3g	2 × 3 × 4	-2212707	0:05:21	-2212707	462
	2 × 3 × 5	-2081357	1:26:47	-2081358	2375
	2 × 4 × 5	-3578762	2:30	-3578762	5281
	3 × 3 × 3	-2932403	0:24:47	-2932403	3003
	3 × 3 × 4	-3552295	5:41:58	-3559337	14475
	3 × 3 × 5	-4561622	2:04:49	-4648539	3729
	3 × 4 × 4	-5371414	7:14:12	-5466518	9536
	3 × 4 × 5	-5474952	24:49:15	-5530625	14963
	4 × 4 × 4	-7619675	9:30:19	-7646881	17825

Table 5.8: SBC results for $k = 5$ and 7. The time is given in hr:min:sec.

5.2 Comparison and Analysis

The computational results which we have presented for graphs with random edge weights and for spin-glass problems using Gaussian and ± 1 distributions lead to the following observations:

1. SBC found the optimal objective for problems with Gaussian distributed and ± 1 interactions of dimension 2 and 3 and with sizes up to $n = 60$ and k=3 in a reasonable amount of time. For $60 < n \leq 100$ we can find a gap of 1% with a reasonable amount of time, however reaching a 0% gap converges slowly.

2. SBC found the optimal objective for grid_2D and toroidal_grid_2D instances with k=3 of sizes up to 80 in less than an hour and at the root node.

3. Comparing SBC on random instances with 100% density with spinglass2g and spinglass2pm instances, we notice for the same n the time required for random type instances is larger. This is because the spinglass instances are sparse while random type instances are complete graphs and hence density might have an effect on the computational time.

4. For $k = 5$ and 7, our empirical analysis shows that for a given $|V|$ as k increases, the computational time decreases. Moreover, for some test cases the objective function values of the same test instance with different k values are the same, see Table 5.8. This is because the solution partitioned the vertices into partitions less than k due to the presence of positive and negative edge weights.

5. The remarkable tightness of the bounds obtained at the root node (in most cases $<$ 5% gap) make it worthwhile to conduct a branch-and-cut algorithm since this might reduce the number of nodes of the tree.

6. Furthermore, the ICH heuristic algorithm applied to the solution of (SMKP-C) seems to often provide an optimal solution at the root node or after only a few branches. Most of the times the lower bound is the bottleneck where we often have the optimal solution from the heuristic but cannot prove optimality.

Biq Mac is a software specialized for $k = 2$, hence, as shown in Table 5.7, it is expected to perform better than SBC which is applicable for any value of k. Moreover, Biq Mac uses the bundle method that is specialized for max-cut and is much faster than the general-purpose CSDP solver.

Chapter 6

Conclusion and Future Work

In this thesis we considered the minimum k-partition problem. We developed and implemented the novel ICH heuristic which appears to be a promising method for generating a good feasible solution. The proposed model improves the upper bound and gives a good feasible solution. ICH can be applied to the MkP problem for different values of k. When compared with other approaches in the literature such as the hyperplane rounding technique by Frieze and Jerrum, it provides a better solution. Moreover, the ICH heuristic was used in a SDP-based branch-and-cut approach to provide optimal solution for MkP. The SBC algorithm was implemented and tested using several instances and different values of k. Finally, computational results show the potentials of SBC in handling the MkP problem.

Future research should investigate the algorithm used to solve the SDP at each node of the tree since this is the major bottleneck in SBC algorithm. In particular, exploiting the structure of the graph and its sparsity may lead to an efficient way for solving the SDP relaxations. In addition, improvements can be made on the separation of clique and triangle inequalities in order to be used efficiently in SBC. Furthermore, adding more valid inequalities might tighten the bound and reduce the computational time. Also we might investigate warm starting our branch-and-cut algorithm so that we can use information generated at parent nodes and especially at the root node when branching, instead of solving the SDP relaxation at each node from scratch. Future work also includes modifying the SBC algorithm and the ICH heuristic so that they can be applied to closely related partitioning problems such as the k-way equipartition problem. Finally, we also need to improve the computational time for solving the strengthened SDP relaxation to be able to utilize it in the branch-and-cut algorithm.

Bibliography

[1] *http://glaros.dtc.umn.edu/gkhome/metis/metis/overview.*

[2] M. Anjos. *New Convex Relaxations for the Maximum Cut and VLSI Layout Problems.* PhD thesis, University of Waterloo, 2001.

[3] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operation Research*, 36:493–513, 1988.

[4] F. Barahona, M. Jünger, and G. Reinelt. Experiments in quadratic 0-1 programming. *Mathematical Programming*, 44:127–137, 1989.

[5] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36:157–173, 1986.

[6] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods Software*, 11/12(1-4):613–623, 1999.

[7] S. Burer and R. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming*, 95(38, Series B):329–357, 2003.

[8] S. Chopra and M. R. Rao. The partition problem. *Mathematical Programming*, 59:87–115, 1993.

[9] S. Chopra and M. R. Rao. Facets of the k-partition problem. *Discrete Applied Mathematics*, 61:27–48, 1995.

[10] M. Deza, M. Grotschel, and M. Laurent. Complete descriptions of small multicut polytopes. *Applied Geometry and Discrete Mathematics - The Victor Klee Festschrift*, 4:205–220, 1991.

[11] M. Deza, M. Grotschel, and M. Laurent. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, 17:981–1000, 1992.

[12] M. Deza and M. Laurent. *Geometry of Cuts and Metrics*, volume 15. Algorithms and Combinatorics, 1997.

[13] J. Domingo-Ferrer and J. M. Mateo-Sanz. Practical data-oriented microaggregation for statistical disclosure control. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):189–201, 2002.

[14] R. Duffin. Infinite programs. *In Linear inequalities and related systems, Annals of Mathematics Studies*, 38:157–170, 1956.

[15] A. Eisenblätter. *Frequency Assignment in GSM Networks: Models, Heuristics, and Lower Bounds.* PhD thesis, Technische Universität, 2001.

[16] A. Eisenblätter. The semidefinite relaxation of the k-partition polytope is strong. *Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, 2337:273–290, 2002.

[17] K. H. Fischer and J. A. Hertz. *Spin glasses.* Cambridge University Press, 1991.

[18] A. Frieze and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18:67–81, 1997.

[19] M. Goemans and D. Williamson. New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem. *SIAM Journal of Discrete Mathematics*, 7(4):656–666, 1994.

[20] M. Goemans and D. Williamson. Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming. *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 443–452, 2001.

[21] C. Helmberg. *Semidefinite programming website. http://www-user.tu-chemnitz.de/ helmberg/semidef.html.*

[22] C. Helmberg and F. Rendl. Solving quadratic (0, 1)-problems by semidefinite programs and cutting planes. *Mathematical Programming*, 82(3, Series A):291–315, 1998.

[23] C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior point method for semidefinite programming. *SIAM Journal on Optimization*, pages 342–361, 1996.

[24] Michael Jünger. *Spin-glass server. http://www.informatik.uni-koeln.de/ls_juenger/research/sgs/index.html.*

[25] D. Karger, R. Motwani, and M. Sudan. Approximate graph coloring by semidefinite programming. *Journal of the ACM*, 45(2):246–265, 1998.

[26] G. Karypis and V. Kumar. Multilevel graph partitioning schemes. Technical report, Department of Computer Science, University of Minnesota, 1995.

[27] M. Kocvara and M. Stingl. Pennon - a code for convex nonlinear and semidefinite programming. *Optimization Methods and Software*, 18(3):317–333, 2003.

[28] J. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization*, 12(3):756–769, 2002.

[29] L. W. Lee, Helmut G. Katzgraber, and A. P. Young. Critical behavior of the three- and ten-state short-range Potts glass: A Monte Carlo study. *Physical Review B*, 74:104–116, 2006.

[30] F. Liers. *Contributions to determining exact ground-states of Ising spin-glasses and to their physics*. PhD thesis, Universität zu Köln, 2004.

[31] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard ising spin glass problems by branch-and-cut. *Wiley*, pages 47–68, 2004.

[32] A. Lisser and F. Rendl. Telecommunication clustering using linear and semidefinite programming. Technical report, Institüt für Mathematik, Universität Klagenfurt, A-9020 Klagenfurt, Austria, 2000.

[33] L. Lovász. On the Shannon capacity of a graph. *IEEE Transactions Information Theory*, IT-25:1–7, 1979.

[34] M. Mezard, G. Parisi, and M. A. Virasoro. *Spin Glass Theory and Beyond*. World Scientific, 1987.

[35] J. Mitchell. Branch-and-cut for the k-way equipartition problem. Technical report, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 2001.

[36] J. E. Mitchell. Realignment in the National Football League: Did they do it right? *Naval Research Logistics*, 50(7):683–701, 2003.

[37] J. Povh, F. Rendl, and A. Wiegele. A boundary point method to solve semidefinite programs. *Computing*, 78(3):277–286, 2006.

[38] F. Rendl, G. Rinaldi, and A. Wiegele. *http://biqmac.uni-klu.ac.at/*. Department of Mathematics Alpen-Adria-Universität Klagenfurt.

[39] F. Rendl, G. Rinaldi, and A. Wiegele. SDP based branch and bound for max-cut. Technical report, Alpen-Adria-Universität Klagenfurt, 9020 Klagenfurt, Austria, 2006.

[40] G. Rinaldi. *Rudy. http://www-user.tu-chemnitz.de/ helmberg/rudy.tar.gz.*

[41] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653, 1999.

[42] K. Toh, M. Todd, and R. Tütüncü. SDPT3 A MATLAB software package for semidefinite programming version 1.3. *Optimization Methods and Software*, 11-12(1-4), 1999.

[43] K. Toh and L. Trefethen. The Chebyshev polynomials of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 20(2):400–419, 1999.

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

[45] A. Wiegele. *Nonlinear Optimization Techniques Applied to Combinatorial Optimization Problems*. PhD thesis, Alpen-Adria-Universität Klagenfurt, 2006.