ABSTRACT

NIE, TIANTIAN. Quadratic Programming with Discrete Variables. (Under the direction of Dr. Shu-Cherng Fang.)

In this dissertation, we study the quadratic programming problem with discrete variables (DQP). DQP is important in theory and practice, but the combination of the quadratic feature of the objective function and the discrete nature of the feasible domain makes it hard to solve. In this thesis, we utilize state-of-the-art continuous optimization techniques to study DQP. A new relaxation of DQP based on the linear conic approach is proposed. Numerical results support the high-quality of lower bounds obtained from the proposed relaxation. We then develop a new linearization method to provide effective 0-1 mixed-integer linear programming (MILP) reformulations of DQP. The new method uses a minimum number of linear inequalities to realize the binary representation of the product of discrete variables. It outperforms other known state-of-the-art linearization methods as supported by intensive computational experiments. We also extend the research to study a special linearly constrained DQP problem, called l^1-norm constrained convex quadratic programming problem with integer variables (l^1-DQP). By exploiting the embedded first-order cone structure, we develop a new type of nonlinear conic cuts induced by the discrete nature of the problem. The proposed conic cuts are shown to work more effectively than other known conic cuts in solving l^1-DQP using a branch-and-bound scheme. The research in this dissertation has revealed the potential of utilizing continuous optimization techniques and exploring special discrete structures in developing efficient algorithms for solving DQP. Future research directions are included at the end of this document.

Quadratic Programming with Discrete Variables

by
Tiantian Nie

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Industrial Engineering

Raleigh, North Carolina

2016

APPROVED BY:

Dr. James R. Wilson	Dr. Yunan Liu
Dr. Osman Ozaltin	Dr. Shu-Cherng Fang Chair of Advisory Committee

DEDICATION

Dedicated to

My parents Yongjian Nie and Yanming Wei
And my boyfriend Weihang Liu

BIOGRAPHY

Tiantian Nie, the only daughter of Yongjian Nie and Yanming Wei, was born on December 10th, 1991 in Anlu, Hubei, China. Attending primary school at a young age, she grew up in love and encouragement. In 2005, she was admitted to No. 1 Middle School affiliated to Central China Normal University in Wuhan, Hubei and started her boarding life there. In 2008, she attended Tsinghua University in Beijing, where she received her B.S. degree in Mathematics Science in 2012. After that, she joined North Carolina State University to pursue her Ph.D. degree in the Edward P. Fitts Department of Industrial and Systems Engineering under the supervision of Dr. Shu-Cherng Fang.

ACKNOWLEDGEMENTS

I would like to express my deepest and sincerest gratitude to my advisor Dr. Shu-Cherng Fang for his great guidance, support and encouragement through my Ph.D. study. He not only supervises me on conducting good research, but also sets a perfect example for me to be a rigorous scholar, kindly teacher and wise person. More importantly, he is like a father encouraging me to enjoy life and live happily.

I am also very grateful to my committee members - Dr. James R. Wilson, Dr. Yunan Liu and Dr. Osman Ozaltin for their valuable comments and suggestions on my research and presentation. Their encouragement inspired me to excel in my work and be more confident. Thanks also go to the graduate school representative Dr. Min Liu for her generous support and help. I would also like to thank Dr. John E. Lavery for his kindly help in my Ph.D. study and his strong mind that inspired me a lot.

I want to thank Mr. Edward Fitts for the Edward P. Fitts Fellowship that supports my graduate study. Also I am grateful to the staff of the Industrial and Systems Engineering Department, especially Ms. Cecilia Chen, Mr. William Irwin, Mr. Justin Lancaster and Mr. Robert Lasson, for their support and help during my graduate study.

I am really lucky to receive the support and encouragement from all my fellow friends in the FANGroup and the ISE Department: Ye Tian, Zhibin Deng, Jian Luo, Ziteng Wang, Chien-Chia Huang, Jiahua Zhang, Giyoung Kim, Yao Yu, Sha Luo, Qi An, Ling Zhang, Yu-Liang Lin, Chi-Yi Chen, Shan Jiang and so many others that I cannot list them all. In particular, I would like to thank Zhibin Deng for his guidance on my graduate study. Sincere thanks also go to my dear friends Yicong Yong, Eileen Gu and all roommates in apartment 101 for their friendship that made my life sweet in Raleigh.

Lastly but most importantly, I am deeply grateful to my dad, mom, and grandparents for their endless love and support. They always stand beside me whenever I face difficulties in life. I would also like to express special thanks to my boyfriend, Weihang Liu, for his companionship, understanding and sincere love that support me throughout my Ph.D. study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Quadratic programming with discrete variables (DQP) is an important nonlinear discrete optimization problem. Motivated by the progress made in quadratic programming and integer programming, DQP has recently moved into the focus of optimization research. The aim of this dissertation is to investigate more effective solution methods for DQP by exploiting its special structure and utilizing the advanced continuous optimization techniques.

1.1 Problem Statement and Motivation

A quadratic programming problem with discrete variables is a mathematical optimization problem in which each variable is restricted to a finite set of discrete values and the objective function is quadratic. DQP can be written in the following form:

$$\begin{aligned} \min \quad & \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} \quad & x_i \in \{d_1^i, \ldots, d_{m_i}^i\}, \quad i = 1, \ldots, n, \end{aligned} \tag{1.1}$$

where $\boldsymbol{A} \in \mathcal{S}^n$ is a symmetric matrix, $\boldsymbol{c} \in \mathbb{R}^n$ is a cost vector and $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of decision variables in which x_i may take m_i possible values of $\{d_1^i, d_2^i, \ldots, d_{m_i}^i\}$ with $d_1^i < d_2^i < \ldots < d_{m_i}^i$, for $i = 1, \ldots, n$. Without loss of generality, we may assume that $d_1^i \geqslant 0$ and $m_1 = m_2 = \ldots = m_n$. But our work is also applicable to the general case with some $d_1^i < 0$ and $m_i \neq m_j$, $i, j \in \{1, \ldots, n\}$ and $i \neq j$.

DQP has many applications in engineering and system sciences. For example, the problem of filtered approximation (FA) in signal-processing [23] is to find discrete-time discrete-valued signals to approximate the given continuous-valued signals such that difference of these two types of signals, once passed through a filter, exhibits the minimum energy. Since the energy of a filtered signal can be quadratically characterized, the FA problem can be transformed into an

equivalent DQP problem. Other applications include the closest vector problem in communication systems [57], optimal design in process systems engineering [11] and resource allocation in distributed computing systems [28]. DQP also has great importance for theoretical research in discrete optimization. The well-known max-cut problem [65], binary quadratic programming problem [84] and ternary quadratic programming problem [63] are special cases of DQP, with $x_i \in \{-1, 1\}$, $x_i \in \{0, 1\}$ and $x_i \in \{-1, 0, 1\}$, for $i = 1, \ldots, n$, respectively. So is the integer quadratic programming problem [56], with $x_i \in \mathbb{Z} \cap [l_i, u_i]$ and $l_i \leqslant u_i$ for $i = 1, \ldots, n$, which has recently attracted lots of attention.

It is known that DQP is nondeterministic polynomial-time hard (NP-hard) in general [58, 62], even for the simple version of binary quadratic programming. With integer variables, linear programming problems have been extensively studied for many decades [70], while practical methods for the nonlinear case, even when the objective is quadratic, are still rare and ineffective [36]. Moreover, since DQP requires variables to be of general discrete values, known methods of integer quadratic programming may not be directly applicable for solving DQP.

Since it is hard to find exact solutions to DQP (due to the NP-hardness), one may compromise to consider high-quality approximate solutions that can be obtained in polynomial time. In practice, heuristic techniques search for good candidate solutions, however, they are restricted to specific subclasses of DQP with special structures [23]. From a global optimization point of view, the task is to explore tight and polynomial-time solvable approximations for DQP. Along this direction, both the discreteness and non-convexity of the problem need to be treated. In the literature, approximations of DQP have been studied but mainly for the subclass of 0-1 binary quadratic programs (BQP) [30, 55, 56, 84]. Note that the 0-1 binary requirement of variables is equivalent to a set of quadratic equality constraints, hence BQP also belongs to the class of continuous quadratic programming problems (QP). The difficulty in developing approximations for BQP is then reduced to handling the non-convexity only. The embedded continuous nature of BQP also enables researchers to investigate high-quality approximations of BQP following those of QP, which have been extensively studied using advanced continuous optimization techniques in recent decades. When it comes to DQP with general discrete variables, effective approximate solutions methods that handle both the discreteness and non-convexity of DQP are still missing.

The approximate solution of DQP obtained could be of high-quality, but is difficult to be verified as an exact solution due to the NP-hardness of the problem. For general NP-hard problems, the widely adopted approach of finding exact solutions is a branch-and-bound scheme that systematically explores the branching nodes of a rooted tree consisting of all feasible solutions [43]. A branching node is checked against some known upper and lower bounds, and is discarded if it cannot produce a better solution to replace the current incumbent. Otherwise, the node is further explored. In this way, all candidate solutions are enumerated in a system-

atic manner. For finding exact solutions, the efficiency of a branch-and-bound scheme is then impacted by both the complexity of the branching tree and the computational cost at each branching node. A tighter bounding routine may reduce the number of nodes explored, but at the same time imply heavier computational burden of node checking. Hence, A trade-off exist between the computational efficiency and quality of the lower bound obtained for branching. In the literature, we have seen extensive studies on the branch-and-bound schemes of classical mixed-integer linear programming problems (MILP). The scheme branches according to the integer requirement of variables and solves continuous relaxation problems to provide lower bounds for branching. In the meanwhile, an advanced technique that incorporates valid linear inequalities induced by the integer requirement (called as "mixed-integer linear cut") is usually applied for improving the overall computational efficiency of the branch-and-bound scheme.

For our problem DQP, a sublass of nonlinear discrete optimization, branch-and-bound based exact solution methods can be explored in two main directions. On one hand, since efficient branch-and-bound based commercial MILP solvers are available, it remains to develop effective MILP representations for DQP. In fact, for a continuous nonlinear program, the equivalent linear representation does not exist in general. It is the discrete nature of DQP that enables us to reconstruct the nonlinear feature of DQP in a linear manner. Therefore, exploiting the nonlinear feature's discrete property may help us design effective MILP reformulations for DQP. The other exact solution method is to develop efficient branch-and-bound algorithms for DQP with specific branching rules and bounding routines. Intuitively, one may branch according to the discrete requirement of variables and then solve convex continuous relaxation problems for providing lower bounds at branching nodes. Some convex-relaxation-based branch-and-bound algorithms of DQP can be found in [18, 19, 20]. In the meanwhile, the conventional linear cut generation method does not apply here since DQP is a nonlinear problem. But in the special case when DQP has a convex objective function, we have seen some preliminary theoretical research on the mixed-integer nonlinear cuts in the recent literature [7, 8].

From the above mentioned, we may see that the study of DQP is an emerging research area yet effective solution methods of DQP are less studied. In particular, the discrete, nonlinear and non-convex properties have not been fully exploited in current solution methods of DQP. In this dissertation, we intend to explore more effective solution methods for DQP by exploiting its special structure and utilizing state-of-the-art continuous optimization techniques.

1.2 Contributions

In this dissertation, several effective solution methods of DQP are obtained.

(1) A linear conic relaxation of DQP has been proposed.

A linear conic programming (LCoP) problem minimizes a linear function over the intersection of an affine subspace and a convex cone, which may involve special structures such that the LCoP problem can be solved in polynomial time. The advanced linear conic relaxation approach was initiated for continuous quadratic programming problems (QP). It packs all difficulties of QP into a convex cone in a higher dimensional matrix-space and then relaxes the convex cone to a well-structured one. In this way, a polynomial-time solvable LCoP relaxation problem is constructed. Several linear conic relaxations for QP have been shown to be tight for providing high-quality lower bounds [16, 22, 54, 56].

We adopt the linear conic relaxation approach to propose a special linear conic relaxation for DQP, based on a linearly constrained 0-1 binary quadratic programming reformulation of DQP. Numerical results verify that the proposed linear conic relaxation is capable of providing high-quality and robust lower bounds for DQP. Moreover, we find that the discrete nature of DQP brings a special property to the proposed linear conic relaxation. The optimal solution of the proposed relaxation problem is in the form of a matrix, whose rank is defined as the size of the largest collection of linearly independent columns. We have shown that when the proposed relaxation problem has an optimal solution with rank one or two, optimal solutions to the original problem DQP can be explicitly generated. This special property can be further extended to the linear conic relaxations of DQP with linear constraints. The results of this part of work have been published in [60].

(2) A new linearization method has been developed.

For the purpose of finding exact solutions to DQP, we fully exploit the discrete structure embedded in the discrete quadratic terms to develop effective 0-1 MILP reformulations for DQP. Note that the quadratic feature can be generalized to the product of multiple variables. We first investigate the linearization of the following generalized signomial programming problem with discrete variables (DSP):

$$\begin{array}{ll} \min & \sum_{j=1}^{J} \left(c_j \prod_{i=1}^{n} (x_i)^{a_{i,j}} \right) \\ \text{s.t.} & x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, \end{array}$$

where $c_j, a_{i,j} \in \mathbb{R}$ for $i = 1, \ldots, n$, $j = 1, \ldots, J$ and the objective is a signomial function [27]. Linearization methods for DSP have been well studied for many decades [2, 41, 46].

A state-of-the-art method, called Logarithmic Method, that uses a minimum number of 0-1 binary variables in the MILP reformulation was motivated from the binary representation of discrete variables. We develop a Super Logarithmic Method to further realize the binary representation of the product of discrete variables by using a minimum number of

inequality constraints and binary variables. Both the theoretical analysis and numerical results strongly support its superior performance when compared with known state-of-the-art linearization methods. Later, we customize the proposed method to provide effective 0-1 MILP reformulations for DQP. In particular, for DQP with special network-structured coefficients, the reformulation of DQP can be further enhanced for better efficiency. The results of this work can be found in a published paper [45].

(3) A variation of DQP with l^1-norm constraints has been explored.

We further extend our study to a special linearly constrained DQP problem, called l^1-norm constrained convex quadratic programming with integer variables (l^1-DQP), which is commonly seen in real-life applications due to the favorable sparse and outlier-robust properties of the l^1-norm. While the solution concepts of DQP are naturally applicable to l^1-DQP, here we explore the special conic structure of the l^1-norm to investigate advanced solution methods for l^1-DQP.

Due to the convexity of the quadratic objective function, second-order conic (SOC) cuts induced from the integer requirement are usually adopted in solving l^1-DQP by a branch-and-bound scheme [7, 8, 24]. By exploiting the first-order conic structure embedded in the l^1-norm constraint, we develop new ways to generate and manage conic cuts of l^1-DQP. Numerical results show that the proposed conic cuts work more effectively than the SOC conic cuts in reducing the complexity of branching trees and improving the computational efficiency of the branch-and-bound scheme. The results of this work have been documented in a working paper [61].

The research in this dissertation reveals that the combination of the quadratic feature and the discrete nature, which makes DQP a difficult problem, also provides opportunities to develop effective solution methods for DQP. We propose a high-quality linear conic relaxation and a more effective MILP reformulation of DQP for finding approximate solutions and exact solutions, respectively. These results can be directly extended to more complicated DQP problems with additional linear constraints and quadratic constraints. In addition, we explore an advanced solution method that generates conic cuts for solving a linearly constrained DQP problem with special conic structure embedded. The linear conic relaxation approach and conic cut generation method both take the conic structure of DQP into consideration, verifying the potential of using conic optimization techniques to handle nonlinear and non-convex features to which conventional approaches do not apply. In the meanwhile, we have shown that it is important to utilize the special discrete structure of DQP for developing effective exact solutions methods for DQP. The approaches and results in the dissertation can be further extended and may provide insights for future studies on DQP-related nonlinear discrete optimization problems.

1.3 Outline of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, we provide a literature review of DQP in detail. Background information and knowledge about linear conic optimization and quadratic programming with continuous variables are included. In Chapter 3, a linear conic relaxation for DQP is proposed for generating high-quality lower bounds for DQP. In Chapter 4, a Super Logarithmic Method is developed to linearize DQP as a 0-1 mixed integer linear programming problem for finding an exact optimal solution. In Chapter 5, mixed-integer conic cuts of l^1-DQP are explored. In Chapter 6, we summarize our findings and provide some future research directions.

Chapter 2

Literature Review and Basic Knowledge

In this chapter, we review the related studies of DQP and some preliminary knowledge. Some useful notations are first introduced in Section 2.1. In Section 2.2, a literature review of DQP is presented, while some basic knowledge about the relation between quadratic programming and linear conic optimization is given in Section 2.3.

2.1 Notation

We denote the set of real numbers and nonnegative real numbers by $\mathbb{R}$ and $\mathbb{R}_+$, respectively. In the n-dimensional real space $\mathbb{R}^n$, let the set of nonnegative real vectors be $\mathbb{R}^n_+$. Denote the set of nonnegative integer numbers and the set of nonnegative integer vectors in $\mathbb{R}^n$ by $\mathbb{Z}_+$ and $\mathbb{Z}^n_+$, respectively. The l^p norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is given by $\|\boldsymbol{x}\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$, for $p \in \mathbb{Z}_+$ and the corresponding n-dimensional p^{th}-order cone is defined as $\left\{\boldsymbol{x} \in \mathbb{R}^n |\ (|x_1|^p + \ldots + |x_{n-1}|^p)^{1/p} \leq x_n\right\}$. In particular, the n-dimensional second-order cone is denoted by $\mathcal{L}^n$.

Let $\mathbb{R}^{m\times n}$ be the set of $m \times n$ real matrices, $\mathcal{S}^n$ be the set of $n \times n$ symmetric real matrices, $\mathcal{S}^n_+ = \{\boldsymbol{A} \in \mathcal{S}^n : \boldsymbol{A} \succcurlyeq 0\}$ be the set of $n \times n$ positive semidefinite matrices, $\mathcal{N}^n = \{\boldsymbol{A} \in \mathcal{S}^n : \boldsymbol{A} \geq 0\}$ be the set of $n \times n$ entry-wise nonnegative matrices. The set $\mathcal{S}^n_+ \cap \mathcal{N}^n$ is called a doubly nonnegative cone. For a matrix $\boldsymbol{A} \in \mathcal{S}^n$, A_{ij} stands for the element in the i-th row and j-th column of $\boldsymbol{A}$. Let $\mathbf{1}_{m\times n}$ and $\mathbf{0}_{m\times n}$ be $m \times n$ matrices with all entries equal to 1 and 0, respectively, and $\boldsymbol{I}_n$ be a diagonal $n \times n$ matrix with all diagonal entries equal to 1. The dot product operation in $\mathbb{R}^{m\times n}$ is given by $\boldsymbol{A} \bullet \mathbf{B} = \text{trace}(\mathbf{A}\mathbf{B}^{\mathbf{T}})$.

Given a set $\mathcal{F} \subseteq \mathbb{R}^{m\times n}$, the smallest closed set containing $\mathcal{F}$ is defined to be the closure of $\mathcal{F}$, denoted by $\texttt{cl}\,\{\mathcal{F}\}$. The *convex hull* of $\mathcal{F}$ is denoted by $\texttt{conv}\,\{\mathcal{F}\}$, as the smallest convex set

that contains $\mathcal{F}$. Whereas, the *conic hull* of $\mathcal{F}$ defined as the smallest convex cone containing $\mathcal{F}$ is denoted by $\texttt{cone}\,\{\mathcal{F}\}$. Given two sets A and B, let $A\backslash B$ denote the relative complement of A with respect to B, i.e., the set of elements in A but not in B.

2.2 Quadratic Programming with Discrete Variables

As a commonly seen practical model in engineering design and system sciences, DQP was primarily studied using heuristic techniques such as simulated annealing [6] and genetic algorithms [50]. These heuristic techniques find some candidate solutions that may be close to an optimal one for DQP. In addition, some effective relaxation and reformulation techniques and branch-and-bound methods have been developed for finding solutions to DQP.

2.2.1 Relaxation Approach

Relaxations of DQP construct polynomial-time solvable problems that provide approximate solutions and high-quality lower bounds for DQP. In the literature, most relaxations of DQP considered the special case when the discrete variables are binary, in particular, a 0-1 binary quadratic programming problem (BQP). Linear conic relaxation was originally applied to quadratic programming with continuous variables (QP) which includes BQP as a special case. We will review the relationship between linear conic programming (LCoP) and QP in Section 2.3.

On the linear conic relaxation of BQP, two well-known methods are based on the semidefinite programming (SDP) [55, 81] and reformulation-linearization technique (RLT) [75, 77], respectively. SDP relaxation is to drop the rank-one constraint in an SDP reformulation. Specifically, a matrix $\boldsymbol{X}$ is introduced to relax the desired equation $\boldsymbol{X} = \boldsymbol{x}\boldsymbol{x}^T$, which is nonconvex, to a convex constraint $\boldsymbol{X} \succcurlyeq \boldsymbol{x}\boldsymbol{x}^T$ by deleting the restriction of $\text{rank}(\boldsymbol{X}) = 1$. The gap between a BQP problem and its SDP relaxation has been analyzed in [56, 84]. The RLT method is to provide an equivalent reformulation by introducing new variables to represent product terms and a convex envelope to cover the lifted feasible domain. A brief review of the RLT method can be found in Section 2.3. Based on the RLT reformulation, it is applicable to develop effective conic relaxations for a specific BQP by choosing proper convex tractable cones. The RLT method also provides a way for constructing valid cuts for a linear conic relaxation to achieve improved lower bounds [4]. Besides the SDP and RLT relaxations, motivated by [21], BQP can also be represented as a copositive program. Although the copositive cone is not polynomial-time detectable, it is indeed a well-structured convex cone and its tractable relaxations have been explored in [16]. These linear conic relaxations have been shown to be tight for providing high-quality lower bounds and approximate solutions to BQP [16, 56, 65, 84, 93].

However, research on the relaxations of the general DQP problem can hardly be found except [20], which was motivated by the observation that DQP can be modeled as an SDP plus a rank-one constraint. Embedded in a branch-and-bound framework of DQP, this SDP relaxation was not designed for providing high-quality approximate solutions and lower bounds for DQP.

2.2.2 Reformulation Techniques

Although good commercial solvers can directly handle nonconvex mixed integer programming problems, the computation may still be inefficient. In order to improve the computational efficiency, an effective representation of DQP is desired for fast computation.

An initial step of reformulating DQP linearly expresses the discrete variables. For a discrete variable $x_i \in \{d_1^i, \ldots, d_{m_i}^i\}$, a naive linearization method is to introduce new binary variables $u_l \in \{0,1\}, l = 1, \ldots, m_i$, satisfying $u_1 + u_2 + \ldots + u_{m_i} = 1$ such that x_i can be linearly represented as $x_i = d_1^i u_1 + d_2^i u_2 + \ldots + d_{m_i}^i u_{m_i}$. In practice, however, this linearization method would require a huge number of binary variables, which significantly limits the computational efficiency of a branch-and-bound-based solver. Recently, a logarithmic method is developed that requires a half order fewer binary variables to linearly represent a discrete variable, at the cost of introducing more continuous variables and linear constraints [12, 46, 89]. Numerical results showed that the logarithmic method is superior to other linearization methods in terms of the computational efficiency. With discrete variables linearly expressed, reformulation techniques of DQP fall into two main categories.

One is linearizing DQP as a 0-1 mixed integer linear programming problem. The main issue arises from the linearization of quadratic product terms with discrete variables [12, 72]. Expressing the discrete variables by 0-1 binary variables, it suffices to linearize the corresponding 0-1 quadratic programming problem. A most widely used technique of linearizing a quadratic term xy with $x, y \in \{0,1\}$ is the RLT method [73, 74, 77, 78], which introduces a binary variable $z \in \{0,1\}$ satisfying $z \leqslant x, z \leqslant y$ and $z \geqslant x + y - 1$ to represent the nonlinear term xy.

The other linearization technique is reformulating DQP as a 0-1 mixed integer convex programming problem. Different from the linearization technique, the quadratic convex reformulation (QCR) method is simply realized by adding a parameterized perturbation. For instance, consider the nonconvex quadratic function $f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T \boldsymbol{x}$ with variables $\boldsymbol{x} \in \mathcal{F} \subseteq \mathbb{R}^n$ and coefficients $\boldsymbol{A} \in \mathcal{S}^n$ and $\boldsymbol{c} \in \mathbb{R}^n$. As long as the parameter λ satisfies $\boldsymbol{A} - \lambda \boldsymbol{I}_n \succcurlyeq 0$, the perturbed function $\bar{f}(\boldsymbol{x}; \lambda) \triangleq f(\boldsymbol{x}) + \lambda \sum_{i=1}^n (v_i - x_i^2)$ becomes a convex quadratic function with $\bar{f}(\boldsymbol{x}; \lambda) = f(\boldsymbol{x})$ under the condition of $v_i = x_i^2, i = 1, \ldots, n$. However, to find a good parameter λ that provides the tightest continuous relaxation bound requires solving a semidefinite programming program. For the QCR methods of DQP, there exist different choices for the perturbation

term. This leads to various 0-1 mixed integer convex quadratic reformulations [13, 14].

2.2.3 Branch-and-Bound Algorithms

Given a representation of DQP, branch-and-bound algorithms develop specific routines of branching and bounding that may significantly improve the computational efficiency of finding exact solutions to DQP. Most branch-and-bound algorithms face a trade-off between the quality of solution and computational time of bounding.

In [91], the canonical duality conditions of DQP were analytically derived for designing an effective branching routine for branch-and-bound. Recently, Buchheim et al. [18, 19, 20] proposed three different relaxation-based branch-and-bound algorithms for DQP. In [18], a special case of DQP, when the quadratic objective function is convex, was studied. They showed in this case the continuous relaxation of DQP can be easily solved during the branching process and valid cuts can be effectively generated to tighten the bounds. However, this bounding method is inapplicable to nonconvex DQP since the corresponding continuous relaxation can be NP-hard. Subsequently, they proposed a new nonconvex relaxation method [19]. Their nonconvex relaxation is in the form of a trust region problem, which is an important nonlinear optimization problem with many efficient algorithms available [66]. In these two branch-and-bound algorithms [18, 19], lower bounds are provided by continuous relaxations of DQP. Differently, an SDP relaxation of DQP was given by [20], based on which a branch-and-bound algorithm was subsequently designed. The computational results in [18, 19, 20] showed that the SDP-based branch-and-bound algorithm is more efficient for solving DQP with many discrete values for each variable, while less efficient than the other two algorithms when the discrete variables become binary or ternary.

Under the branch-and-bound scheme of DQP, an advanced technique that has recently attracted a lot attention is developing effective nonlinear cuts induced from the discrete requirement, which can then be incorporated into branch-and-bound algorithms to tighten the continuous relaxations at branching nodes. This direction can be viewed as an extension of utilizing mixed-integer linear cuts in solving mixed-integer linear programming problems (MILP). However, due to the nonlinear property of DQP, the conventional linear cuts do not apply here. Alternatively, the progress was made in developing nonlinear conic cuts of mixed-integer second-order cone programming problems (MISOCP), an extension of MILP with second-order cone constraints involved. Note that an n-dimensional second-order cone $\mathcal{L}^n$ is given by $\mathcal{L}^n = \{\boldsymbol{x} \in \mathbb{R}^n | \sqrt{x_1^2 + \ldots + x_{n-1}^2} \leq x_n\}$. When DQP has a convex quadratic objective function and integer variables, an equivalent MISOCP reformulation of DQP exists. Therefore, we may obtain corresponding mixed-integer nonlinear cuts of DQP.

Mixed-integer conic rounding cuts [7] are developed from a two-dimensional polyhedral SOC

representation of an original SOC constraint. In the meanwhile, disjunctive conic inequalities for representing the convex hull of a disjunctive conic set are also generated as conic cuts [8]. It is notable that the theoretical research on nonlinear conic cuts has not yet been extensively studied, not to mention the performance in numerical implementations. However, with the development of commercial solvers using linear conic programming based branch-and-bound algorithms, it shall be important to explore effective conic cuts with both theoretical and numerical support.

2.3 Quadratic Programming and Linear Conic Optimization

A *convex set* $\mathcal{F} \subseteq \mathbb{R}^{m\times n}$ is a subset of $\mathbb{R}^{m\times n}$ such that

$$\lambda \boldsymbol{X}^1 + (1-\lambda)\boldsymbol{X}^2 \in \mathcal{F}, \quad \text{for any } \boldsymbol{X}^1, \boldsymbol{X}^2 \in \mathcal{F} \text{ and } \lambda \in [0,1].$$

A *cone* $\mathcal{C} \subseteq \mathbb{R}^{m\times n}$ is a subset of $\mathbb{R}^{m\times n}$ such that

$$\lambda \boldsymbol{X} \in \mathcal{C}, \quad \text{for all } \boldsymbol{X} \in \mathcal{C} \text{ and } \lambda \geq 0.$$

If $\mathcal{C}$ further satisfies that

$$\text{“}\boldsymbol{X} \in \mathcal{C} \text{ and } -\boldsymbol{X} \in \mathcal{C}\text{” if and only if “}\boldsymbol{X} = 0\text{”},$$

then we call it a *pointed* cone. $\mathcal{C}$ is called a *solid* cone if it has a nonempty interior. If $\mathcal{C}$ is pointed, solid, closed and convex, then we say $\mathcal{C}$ is a *proper* cone.

Given a set $\mathcal{F} \subseteq \mathbb{R}^{m\times n}$, the *convex hull* and *conic hull* of $\mathcal{F}$ are expressed as

$$\texttt{conv}\,\{\mathcal{F}\} = \left\{ \boldsymbol{X} \in \mathbb{R}^{m\times n} \;\middle|\; \boldsymbol{X} = \sum_{i=1}^{r} \alpha_i \boldsymbol{X}^i, \sum_{i=1}^{r} \alpha_i = 1, \boldsymbol{X}^i \in \mathcal{F}, 0 \leq \alpha_i \leq 1, \right. \\ \left. \text{for some } r \in \mathbb{N} \text{ and } i = 1, \ldots, r \right\}$$

and

$$\texttt{cone}\,\{\mathcal{F}\} = \left\{ \boldsymbol{X} \in \mathbb{R}^{m\times n} \;\middle|\; \boldsymbol{X} = \sum_{i=1}^{r} \alpha_i x^i, \boldsymbol{X}^i \in \mathcal{F}, \alpha_i \geq 0, \text{ for some } r \in \mathbb{N} \text{ and } i = 1, \ldots, r \right\},$$

respectively, which directly implies

$$\texttt{conv}\,\{\mathcal{F}\} \subseteq \texttt{cone}\,\{\mathcal{F}\} \subseteq \texttt{cl}\,\{\texttt{cone}\,\{\mathcal{F}\}\}.$$

A linear conic programming (LCoP) problem in the matrix space $\mathbb{R}^{m\times n}$ is defined by

$$\text{(LCoP)} \qquad \begin{array}{ll} \min & \boldsymbol{C}\bullet\boldsymbol{X} \\ \text{s.t.} & \boldsymbol{A}_i\bullet\boldsymbol{X}=b_i,\ i=1,\ldots,I, \\ & \boldsymbol{X}\in\mathcal{C}, \end{array}$$

where $\mathcal{C}$ is a closed convex cone and the notation "$\bullet$" is the dot product. When $m=1$ or $n=1$, (LCoP) degenerates to a linear conic programming problem in the vector space. Linear programming (LP) problems and positive semidefinite programming (SDP) problems are two commonly seen subclasses of the LCoP problem, with the cone $\mathcal{C}$ defined as $\mathbb{R}^n_+$ and $\mathcal{S}^n_+$, respectively. Both of $\mathbb{R}^n_+$ and $\mathcal{S}^n_+$ are proper cones and the corresponding LCoP problems are polynomial-time solvable. When the cone $\mathcal{C}$ is defined as $\mathtt{cl}\,\{\mathtt{cone}\,\{\mathcal{F}\}\}$ for an arbitrary set $\mathcal{F}$, the corresponding LCoP problem is in general NP-hard.

It is not hard to construct an equivalent linear conic programming reformulation for a quadratic optimization problem. Consider the following quadratic programming problem:

$$\text{(Q)} \qquad \begin{array}{ll} \min & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x}+\boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} & \boldsymbol{x}\in\mathcal{F}, \end{array}$$

where $\emptyset\neq\mathcal{F}\subseteq\mathbb{R}^n$ is a feasible domain, convex or not. We first transform (Q) into the following form:

$$\text{(Q}_\text{T}\text{)} \qquad \begin{array}{ll} \min & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{c}^T \\ \boldsymbol{c} & \boldsymbol{A} \end{pmatrix}\bullet\boldsymbol{V} \\ \text{s.t.} & \boldsymbol{V}=\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}^T,\quad \boldsymbol{x}\in\mathcal{F}. \end{array}$$

Note that the objective function of (Q$_\text{T}$) is linear in terms of $\boldsymbol{V}$. Hence, the optimal value of (Q$_\text{T}$) can be achieved on the boundary of the closure of the convex hull of its feasible domain. The closure of the convex hull of the feasible domain of (Q$_\text{T}$) is defined as

$$\mathtt{cl}\left\{\mathtt{conv}\left\{\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}^T\middle|\ \boldsymbol{x}\in\mathcal{F}\right\}\right\}.$$

Consequently, the optimal value of (Q) is equal to that of the following linear conic programming

problem:

$$(\mathrm{Q_{LCoP}}) \qquad \begin{array}{ll} \min & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{c}^T \\ \boldsymbol{c} & \boldsymbol{A} \end{pmatrix} \bullet \boldsymbol{V} \\ \text{s.t.} & V_{11} = 1, \quad \boldsymbol{V} \in \mathcal{D}^*(\mathcal{F}), \end{array}$$

where the closed convex cone $\mathcal{D}^*(\mathcal{F})$ is defined as

$$\mathtt{cl}\left\{\mathtt{cone}\left\{\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}\begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}^T \middle| \boldsymbol{x} \in \mathcal{F}\right\}\right\}.$$

Therefore, the quadratic programming problem (Q) and linear conic programming problem ($\mathrm{Q_{LCoP}}$) are equivalent.

However, reformulating a general nonconvex problem (Q) into a linear conic programming problem ($\mathrm{Q_{LCoP}}$) does not make it easier to solve. All the implicit "difficulties" implied from the original feasible domain $\mathcal{F}$ are packed into the cone $\mathcal{D}^*(\mathcal{F})$. In [83], Sturm and Zhang theoretically derived the representability of $\mathcal{D}^*(\mathcal{F})$ for a special feasible domain $\mathcal{F}$. Specifically, if $\mathcal{F}$ is defined by one quadratic constraint and one linear constraint, then the cone $\mathcal{D}^*(\mathcal{F})$ can be represented by linear matrix inequalities (LMI) over the positive semidefinite cone, which indirectly proved that the original problem (Q) is not NP-hard. However, in a general quadratically constrained case, as long as the original problem (Q) is NP-hard, the linear conic reformulation ($\mathrm{Q_{LCoP}}$) cannot be solved in polynomial time. Then, we may want to relax the "difficult" cone $\mathcal{D}^*(\mathcal{F})$ to a polynomial-time detectable cone such that lower bounds of the original problem can be efficiently obtained. Effective relaxations depend on carefully exploring special structures of the cone $\mathcal{D}^*(\mathcal{F})$ [16, 56, 65, 84, 93].

The well-known SDP relaxation is a special subclass of linear conic relaxation for quadratic programming problems with linear and quadratic constraints [55, 81]. In particular, consider the following binary quadratic programming problem:

$$(\mathrm{BQP}) \qquad \begin{array}{ll} \min & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.} & \boldsymbol{x} \in \{0,1\}^n. \end{array}$$

Note that problem (BQP) is equivalent to the following reformulated problem:

$$\begin{array}{ll} \min & \dfrac{1}{2}\begin{pmatrix} 0 & \boldsymbol{c}^T \\ \boldsymbol{c} & \boldsymbol{A} \end{pmatrix} \bullet \boldsymbol{V} \\ \text{s.t.} & X_{ii} = x_i, \quad i = 1, \ldots, n, \end{array} \tag{2.1}$$

$$\boldsymbol{V} = \begin{pmatrix} 1 & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{X} \end{pmatrix} \in \mathcal{S}_+^{n+1}, \tag{2.2}$$

$$\operatorname{rank}(\boldsymbol{V}) = 1. \tag{2.3}$$

The traditional SDP relaxation of (BQP) is derived as below by deleting the nonconvex constraint, i.e., the rank-one constraint (2.3).

$$\begin{array}{ll} \min & \frac{1}{2} \begin{pmatrix} 0 & \boldsymbol{c}^T \\ \boldsymbol{c} & \boldsymbol{A} \end{pmatrix} \bullet \boldsymbol{V} \\ \text{s.t.} & X_{ii} = x_i, \quad i = 1, \ldots, n, \\ & \boldsymbol{V} = \begin{pmatrix} 1 & \boldsymbol{x}^T \\ \boldsymbol{x} & \boldsymbol{X} \end{pmatrix} \in \mathcal{S}_+^{n+1}. \end{array}$$

The SDP relaxation of (BQP) can actually be regarded as a relaxation of the linear conic reformulation of (BQP). It is realized by relaxing the cone $\mathcal{D}^*(\mathcal{F})$ with $\mathcal{F} = \{0,1\}^n$ to the positive semidefinite cone $\mathcal{S}_+^{n+1}$ with linear constraints $X_{ii} = x_i, i = 1, \ldots, n$.

The other well-known technique for deriving a linear conic relaxation of problem (QP) is the reformulation-linearization technique (RLT). It enables us to generate some additional constraints that help tighten the linear conic relaxation of (QP). RLT initially focused on solving mixed 0-1 linear and polynomial programming problems [73, 74]. It later branched into the more general family of nonconvex polynomial programming problems with continuous variables [78, 79, 80]. Essentially, RLT automatically generates additional nonlinear valid inequalities in a reformulation step, and then replaces each product term by a single continuous variable in the linearization step. A recent review on RLT can be referred to [72].

Suppose that the set $\mathcal{F} \subseteq \mathbb{R}^n$ contains the constraints of $a_i \leqslant x_i \leqslant b_i$ with $a_i \leqslant b_i$, for $i = 1, \ldots, n$. It is not difficult to verify that the following quadratic constraints are valid inequalities with respect to $\mathcal{F}$:

$$\begin{array}{l} (x_i - a_i)(x_j - a_j) \geqslant 0, \\ (x_i - a_i)(b_j - x_j) \geqslant 0, \\ (x_i - b_i)(x_j - b_j) \geqslant 0, \end{array} \tag{2.4}$$

for $i, j = 1, \ldots, n$. These additional nonlinear constraints (2.4) can be further linearized as

$$\begin{array}{l} X_{ij} - a_i x_j - a_j x_i + a_i a_j \geqslant 0, \\ -X_{ij} + a_i x_j + b_j x_i - a_i b_j \geqslant 0, \\ X_{ij} - b_i x_j - b_j x_i + b_i b_j \geqslant 0, \end{array} \tag{2.5}$$

by using a single variable X_{ij} to replace the nonlinear term $x_i x_j$ for $i, j = 1, \ldots, n$. Expressions (2.5) are valid linear constraints with respect to the cone $\mathcal{D}^*(\mathcal{F})$, and they are called RLT constraints. We will use this technique in our study to derive a tight relaxation for DQP.

Chapter 3

A Linear Conic Relaxation Approach

In this chapter, a continuous relaxation approach is proposed for the quadratic programming problem with discrete variables (DQP). We developed a special RLT-based linear conic relaxation of DQP. We show that the proposed relaxation is tighter than the SDP relaxation. Moreover, when the proposed relaxation problem has an optimal solution with rank one or two, optimal solutions to the original DQP problem can be explicitly generated. This rank-two property is further extended to linearly constrained DQP problems. Numerical results indicate that the proposed relaxation is capable of providing high-quality and robust lower bounds for DQP.

3.1 Introduction

Consider the following problem:

$$\begin{array}{lll} & z_0 = \min & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\ \text{(DQP)} & \text{s.t.} & x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, \end{array} \tag{3.1}$$

where $\boldsymbol{A}$ is a symmetric matrix in $\mathcal{S}^n$, $\boldsymbol{c}$ is a vector in $\mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of decision variables $x_i, i = 1, \ldots, n$, each of which takes m possible discrete values $\{d_1^i, d_2^i, \ldots, d_m^i\}$ with $d_1^i < d_2^i < \ldots < d_m^i$.

With discrete variables, the problem (DQP) seems not suitable for linear conic relaxation approaches that are designed for quadratic programming with continuous variables. A different point of view comes from linearly expressing discrete variables by using additional binary 0-1 variables. Accordingly, the problem (DQP) is equivalent to the following linearly constrained

0-1 quadratic programming problem (P):

$$
\begin{aligned}
z_0 = \quad & \min \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{l=1}^{m}\sum_{s=1}^{m}(A_{ij}d_l^i d_s^j)u_l^i u_s^j + \sum_{i=1}^{n}\sum_{l=1}^{m}(c_i d_l^i)u_l^i \\
(\mathrm{P}) \qquad & \text{s.t.} \quad \sum_{l=1}^{m} u_l^i = 1, \quad i = 1,\ldots,n, && \text{(3.2a)} \\
& \qquad\quad u_l^i \in \{0,1\}, \quad i = 1,\ldots,n, \quad l = 1,\ldots,m, && \text{(3.2b)}
\end{aligned}
$$

The transformation is realized by recasting the multi-valued discrete variable x_i of problem (DQP) as

$$x_i = d_1^i u_1^i + d_2^i u_2^i + \ldots + d_m^i u_m^i, \quad i = 1,\ldots,n, \tag{3.3}$$

where $u_l^i, i = 1,\ldots,n$ and $l = 1,\ldots,m$, satisfy (3.2a) and (3.2b). The equality constraints (3.2a) imply that the 0-1 binary variables u_l^i are partitioned into n disjoint vectors $\boldsymbol{u}^i = (u_1^i, u_2^i, \ldots, u_m^i), i = 1,\ldots,n$, such that there is a unique element in $\boldsymbol{u}^i$ taking value 1 for $i = 1,\ldots,n$. Problem (P) can be regarded as a special linearly constrained 0-1 binary quadratic programming (BQP) problem. Therefore, the techniques for BQP could be applied here to solve problem (P).

Although the idea of transforming (DQP) into the BQP problem (P) is simple, to the best of our knowledge, no work has carefully studied the linear conic relaxation of (P). Motivated by the special structure of equality constraints (3.2a), which distinguish problem (P) from general BQPs, we propose a specific RLT-based linear conic relaxation of problem (P), same as saying that of (DQP). To be more specific, we first construct an equivalent RLT reformulation of (P), denoted as (LCoP). Then, we construct a relaxation problem (LCoR_1) by relaxing the convex envelope of (LCoP) to the doubly nonnegative cone. Finally, we add effective RLT inequality constraints to (LCoR_1) to form the desired linear conic relaxation problem (LCoR_2). This proposed linear conic relaxation approach is applicable to linearly and quadratically constrained DQPs as well.

The proposed linear conic relaxation (LCoR_2) is shown to be tighter than the SDP relaxation of problem (P). A sufficient condition to guarantee the zero gap between (LCoR_2) and (P) is that there exists an optimal solution of (LCoR_2) that is also feasible for (LCoP). However, the principle of RLT-based relaxation only assures that a rank-one optimal solution of (LCoR_2) is feasible for (LCoP). In this chapter, by analyzing the structure of the feasible domain of (LCoR_2), we show that any rank-two feasible solution of (LCoR_2) is also feasible for (LCoP). Therefore, if (LCoR_2) has an optimal solution with rank no more than two, the optimal value of (LCoR_2) is equal to that of (DQP) and we can generate optimal solutions of the original problem (DQP). As an extension, we derive the rank-two property for the proposed linear conic

relaxation of linearly constrained DQP problems.

Numerical experiments are conducted to compare the lower bound provided by solving (LCoR_2) with the actual optimal value of (DQP). For randomly generated instances, our results indicate that on average the gaps between (LCoR_2) and (DQP) are near zero with a small standard deviation. The proposed linear conic relaxation can then be used in a branch and bound scheme to provide high-quality lower bounds or directly generate an optimal solution of the original problem.

The organization of the rest of this chapter is as follows. In Section 3.2, we propose a new linear conic relaxation of (DQP). In Section 3.3, we analyze the structure of the proposed relaxation and study the rank-two property. In Section 3.4, numerical experiments are conducted to test the quality of the lower bounds provided by the proposed relaxation. Some extensions are discussed in Section 3.5. Section 3.6 concludes the chapter with a few remarks.

This Chapter can be referred to a published paper [60]. My contributions in [60] include proposing a specific linear conic relaxation for DQP, deriving a rank-two decomposition property of the proposed relaxation and conducting numerical experiments to verify the tightness and robustness of the proposed lower bounds.

3.2 A new linear conic relaxation problem

For simplicity, we rewrite problem (P) as

$$
\begin{aligned}
& z_0 = \min && \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q}^T\boldsymbol{u} \\
\text{(P)} \qquad & \text{s.t.} && \sum_{l=1}^{m} u_l^i = 1, \quad i = 1, \ldots, n, && \text{(3.4a)} \\
& && (u_l^i)^2 - u_l^i = 0, \quad i = 1, \ldots, n, \quad l = 1, \ldots, m, && \text{(3.4b)} \\
& && \boldsymbol{u} = (u_1^1, \ldots, u_m^1, \ldots, u_1^n, \ldots, u_m^n) \in \mathbb{R}^{nm}, && \text{(3.4c)}
\end{aligned}
$$

where

$$
\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{Q}^{11} & \ldots & \boldsymbol{Q}^{1n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{Q}^{n1} & \ldots & \boldsymbol{Q}^{nn} \end{pmatrix} \in \mathcal{S}^{nm} \text{ and } \boldsymbol{q} = \begin{pmatrix} \boldsymbol{q}^1 \\ \vdots \\ \boldsymbol{q}^n \end{pmatrix} \in \mathbb{R}^{nm},
$$

with $Q_{ls}^{ij} = d_l^i A_{ij} d_s^j$ and $q_l^i = c_i d_l^i$, for $i, j = 1, \ldots, n$ and $l, s = 1, \ldots, m$.

Let $N = nm$ and $\mathcal{F}_P$ be the feasible domain of problem (P). Based on the RLT method, we linearize the quadratic term $u_l^i u_s^j$ as U_{ls}^{ij} for $i, j = 1, \ldots, n$ and $l, s = 1, \ldots, m$. Define matrices

$\boldsymbol{U}^{ij} = (U^{ij}_{ls})_{m\times m}$ for $i, j = 1, \ldots, n$ and

$$\boldsymbol{U} = \begin{pmatrix} \boldsymbol{U}^{11} & \ldots & \boldsymbol{U}^{1n} \\ \vdots & \ddots & \vdots \\ \boldsymbol{U}^{n1} & \ldots & \boldsymbol{U}^{nn} \end{pmatrix} \in \mathcal{S}^N.$$

Then (P) has the same optimal value as the following linear conic program [30]:

$$\begin{array}{rll} & z_0 = \min & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{q}^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{pmatrix} \bullet \boldsymbol{Y} \\ \text{(LCoP)} & \text{s.t.} & \sum_{l=1}^{m} u^i_l = 1, \quad i = 1, \ldots, n, \\ & & U^{ii}_{ll} = u^i_l, \quad i = 1, \ldots, n,\ l = 1, \ldots, m, \\ & & \boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix} \in \mathcal{D}^*(\mathcal{F}_P), \end{array}$$

where $\mathcal{D}^*(\mathcal{F}_P)$ is a convex cone generated by $\mathcal{F}_P$ defined as

$$\mathcal{D}^*(\mathcal{F}_P) = \texttt{cl}\left\{ \texttt{cone}\left\{ \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix} \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix}^T \middle| \boldsymbol{x} \in \mathcal{F}_P \right\} \right\}.$$

Because (LCoP) is equivalent to (P), it is not polynomial-time solvable unless P=NP. The equality constraints in (LCoP) can be derived from $\boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix} \in \mathcal{D}^*(\mathcal{F}_P)$ and all the difficulties of solving (LCoP) are packaged into the cone $\mathcal{D}^*(\mathcal{F}_P)$. Since $\mathcal{D}^*(\mathcal{F}_P) \subseteq (\mathcal{S}^{N+1}_+ \cap \mathcal{N}^{N+1}) \subseteq \mathcal{S}^{N+1}_+$, we may relax (LCoP) by the following polynomial-time solvable problem:

$$\begin{array}{rll} & z_1 = \min & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{q}^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{pmatrix} \bullet \boldsymbol{Y} \\ \text{(LCoR}_1\text{)} & \text{s.t.} & \sum_{l=1}^{m} u^i_l = 1, \quad i = 1, \ldots, n, \\ & & U^{ii}_{ll} = u^i_l, \quad i = 1, \ldots, n,\ l = 1, \ldots, m, \\ & & \boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix} \in \mathcal{S}^{N+1}_+ \cap \mathcal{N}^{N+1}. \end{array}$$

Notice that, given a feasible solution $\boldsymbol{u}$ of (P), $\boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{u}\boldsymbol{u}^T \end{pmatrix}$ is a feasible solution of (LCoR$_1$). Since (P) is always feasible, so is problem (LCoR$_1$). Nevertheless, the optimal value of (LCoR$_1$) may be unbounded if its dual problem has no feasible solution. Actually, the lower bound z_1 may not be tight enough even when (LCoR$_1$) is bounded below.

Next, we construct some RLT inequalities for tightening the relaxation (LCoR$_1$). For $j = 1, \ldots, n$ and $s = 1, \ldots, m$, multiplying the equality constraints (3.4a) by u_s^j and $1 - u_s^j$, respectively, we have a class of redundant quadratic equality constraints over $\mathcal{F}_P$. Besides, we multiply the implicit linear constraints $0 \leqslant u_l^i \leqslant 1$, $i = 1, \ldots, n$ and $l = 1, \ldots, m$, by terms of the form u_s^j and $1 - u_s^j$, respectively, to generate redundant quadratic inequality constraints over $\mathcal{F}_P$. Then, using the identity $(u_l^i)^2 = u_l^i$ and linearizing the term $u_l^i u_s^j$ as U_{ls}^{ij} in these quadratic constraints, we obtain the following redundant linear constraints with respect to (LCoP):

$$\sum_{s=1}^{m} U_{ls}^{ij} = u_l^i, \quad i, j = 1, \ldots, n, \ l = 1, \ldots, m; \tag{3.5a}$$

$$U_{ls}^{ij} \leqslant u_l^i, \quad i, j = 1, \ldots, n, \ l, s = 1, \ldots, m; \tag{3.5b}$$

$$u_l^i + u_s^j - U_{ls}^{ij} \leqslant 1, \quad i, j = 1, \ldots, n, \ l, s = 1, \ldots, m. \tag{3.5c}$$

These RLT constraints are candidates of valid cuts to be added to the relaxation problem (LCoR$_1$). In fact, as long as $\boldsymbol{Y}$ is feasible to (LCoR$_1$), constraints (3.5b) and (3.5c) are dominated by (3.5a). The non-redundancy of (3.5a) with respect to (LCoR$_1$) can be verified in polynomial time [30] by checking

$$\boldsymbol{V}^{i,j,l} \notin \mathcal{K} + \mathcal{S}_+^{N+1} + \mathcal{N}^{N+1},$$

where $\boldsymbol{V}^{i,j,l}$ is a matrix in $\mathcal{S}^{N+1}$ such that $\boldsymbol{V}^{i,j,l} \bullet \boldsymbol{Y} = 0$ for $i, j = 1, \ldots, n$ and $l = 1, \ldots, m$, representing the constraints in (3.5a),

$$\mathcal{K} = \operatorname{cone}\left\{ \pm \begin{pmatrix} -2 & \boldsymbol{p}_i^T \\ \boldsymbol{p}_i & \boldsymbol{0} \end{pmatrix}, \pm \begin{pmatrix} 0 & -\boldsymbol{e}_j^T \\ -\boldsymbol{e}_j & \boldsymbol{E}_j \end{pmatrix} \middle| \, i = 1, \ldots, n, \ j = 1, \ldots, N \right\},$$

$\boldsymbol{p}_i$ is a vector in $\mathbb{R}^N$ such that $\boldsymbol{p}_i^T \boldsymbol{u} = 1$, $i = 1, \ldots, n$, representing the constraints (3.4a), the vector $\boldsymbol{e}_j \in \mathbb{R}^N$ has a unique nonzero value of 1 at the j-th entry and the matrix $\boldsymbol{E}_j \in \mathcal{S}^N$ has a unique nonzero value of 1 at the j-th diagonal entry, for $j = 1, \ldots, N$.

Therefore, we add the RLT equality constraints (3.5a) to (LCoR$_1$) as valid cuts and propose

the following linear conic relaxation problem (LCoR_2) for the original problem (DQP):

$$
\begin{aligned}
z_2 = \min \quad & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{q}^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{pmatrix} \bullet \boldsymbol{Y} \\
(\mathrm{LCoR}_2) \qquad \text{s.t.} \quad & \sum_{l=1}^{m} u_l^i = 1, \quad i = 1, \ldots, n, && \text{(3.6a)} \\
& U_{ll}^{ii} = u_l^i, \quad i = 1, \ldots, n,\ l = 1, \ldots, m, && \text{(3.6b)} \\
& \sum_{s=1}^{m} U_{ls}^{ij} = u_l^i, \quad i, j = 1, \ldots, n,\ l = 1, \ldots, m, && \text{(3.6c)} \\
& \boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix} \in \mathcal{S}_+^{N+1} \cap \mathcal{N}^{N+1}. && \text{(3.6d)}
\end{aligned}
$$

The optimal value z_2 of (LCoR_2) is the proposed lower bound for the optimal value of (DQP). We shall study the structure of (LCoR_2) and explore a special sufficient condition for the zero gap between (LCoR_2) and (DQP).

3.3 Special structure and properties

The optimal values of linear conic relaxations (LCoR_1) and (LCoR_2), i.e., z_1 and z_2, respectively, are low bounds for the linear conic programming reformulation (LCoP). Comparing with the exact optimal value z_0 of (LCoP), we have the following Proposition.

Proposition 3.3.1. *$z_0 \geqslant z_2 \geqslant z_1$.*

Proof. The linear conic relaxation (LCoR_2) differs from (LCoR_1) in that (LCoR_2) adds RLT constraints (3.6c). On one hand, (3.6c) is redundant with respect to the feasible domain of (LCoP), and thus $z_0 \geqslant z_2$. On the other hand, (3.6c) does not always hold for feasible solutions to (LCoR_1). Therefore, $z_2 \geqslant z_1$. □

Next, we study the structure of the feasible domain of (LCoR_2) to explore its special properties. Denote the feasible domains of (LCoP) and (LCoR_2) as $\mathcal{G}$ and $\mathcal{G}_2$, respectively. Because the equality constraints in (LCoP) are redundant with respect to $\mathcal{G}$, we rewrite $\mathcal{G}$ as

$$\mathcal{G} = \{\boldsymbol{Y} \in \mathcal{D}^*(\mathcal{F}_P) | Y_{11} = 1\}.$$

Since (LCoR_2) is a relaxation of (LCoP), we know that $\mathcal{G}_2 \supseteq \mathcal{G}$. Also notice that the feasible domain $\mathcal{F}_P$ of (DQP) is bounded, hence we have the next result.

Proposition 3.3.2. *$\mathcal{G}_2$ and $\mathcal{G}$ are convex, closed and bounded sets.*

As an RLT-based relaxation of the linearly constrained 0-1 quadratic programming problem, the next theorem follows [75].

Theorem 3.3.3. *If* $\boldsymbol{Y} \in \mathcal{G}_2$ *and* $\operatorname{rank}(\boldsymbol{Y}) = 1$*, then* $\boldsymbol{Y} \in \mathcal{G}$ *and*

$$\boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{u}\boldsymbol{u}^T \end{pmatrix}, \tag{3.7}$$

for some $\boldsymbol{u} \in \mathcal{F}_P$*.*

The following theorem further shows that any rank-two solution of (LCoR_2) is also feasible for (LCoP).

Theorem 3.3.4. *If* $\boldsymbol{Y} \in \mathcal{G}_2$ *and rank*$(\boldsymbol{Y}) = 2$*, then* $\boldsymbol{Y} \in \mathcal{G}$*. In this case, there exist* $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}} \in \mathcal{F}_P$ *and* $\bar{\lambda} \in (0, 1)$ *such that*

$$\boldsymbol{Y} = \bar{\lambda} \begin{pmatrix} 1 \\ \bar{\boldsymbol{u}} \end{pmatrix} \begin{pmatrix} 1 \\ \bar{\boldsymbol{u}} \end{pmatrix}^T + (1 - \bar{\lambda}) \begin{pmatrix} 1 \\ \bar{\boldsymbol{v}} \end{pmatrix} \begin{pmatrix} 1 \\ \bar{\boldsymbol{v}} \end{pmatrix}^T. \tag{3.8}$$

Proof. Since $\boldsymbol{Y} \in \mathcal{G}_2$, we have $\boldsymbol{Y} \in \mathcal{S}_+^{N+1}$ whose eigenvalues are all nonnegative. The fact that $\operatorname{rank}(\boldsymbol{Y}) = 2$ leads to the following decomposition:

$$\boldsymbol{Y} = \lambda_1 \begin{pmatrix} t_1 \\ \boldsymbol{u} \end{pmatrix} \begin{pmatrix} t_1 \\ \boldsymbol{u} \end{pmatrix}^T + \lambda_2 \begin{pmatrix} t_2 \\ \boldsymbol{v} \end{pmatrix} \begin{pmatrix} t_2 \\ \boldsymbol{v} \end{pmatrix}^T,$$

for some $\lambda_1, \lambda_2 > 0, t_1, t_2 \in \{0, 1\}$, and $\boldsymbol{u} = (u_1^1, \ldots, u_m^1, \ldots, u_1^n, \ldots, u_m^n), \boldsymbol{v} = (v_1^1, \ldots, v_m^1, \ldots, v_1^n, \ldots, v_m^n) \in \mathbb{R}^N$. Because $\lambda_1 t_1^2 + \lambda_2 t_2^2 = Y_{11} = 1$, at least one of t_1 and t_2 should be nonzero. There are two cases.

Case I. $t_1 = t_2 = 1$:

In this case, $\lambda_1 + \lambda_2 = 1$ and

$$\boldsymbol{Y} = \begin{pmatrix} 1 & \lambda_1 \boldsymbol{u}^T + \lambda_2 \boldsymbol{v}^T \\ \lambda_1 \boldsymbol{u} + \lambda_2 \boldsymbol{v} & \lambda_1 \boldsymbol{u}\boldsymbol{u}^T + \lambda_2 \boldsymbol{v}\boldsymbol{v}^T \end{pmatrix} \in \mathcal{G}_2.$$

We analyze all the constraints of $\mathcal{G}_2$ for $\boldsymbol{Y}$:

- From (3.6a), we have $\sum_{l=1}^m (\lambda_1 u_l^i + \lambda_2 v_l^i) = 1$, which is equivalent to

$$\lambda_1 (1 - \sum_{l=1}^m u_l^i) + \lambda_2 (1 - \sum_{l=1}^m v_l^i) = 0, \quad i = 1, \ldots, n; \tag{3.9}$$

- From (3.6b), we have $\lambda_1(u_l^i)^2 + \lambda_2(v_l^i)^2 = \lambda_1 u_l^i + \lambda_2 v_l^i$, hence,

$$\lambda_1(1-u_l^i)u_l^i + \lambda_2(1-v_l^i)v_l^i = 0, \quad i = 1,\ldots,n,\ l = 1,\ldots,m; \tag{3.10}$$

- From (3.6c), we have $\sum_{s=1}^m(\lambda_1 u_l^i u_s^j + \lambda_2 v_l^i v_s^j) = \lambda_1 u_l^i + \lambda_2 v_l^i$, which implies that

$$\lambda_1(1-\sum_{s=1}^m u_s^j)u_l^i + \lambda_2(1-\sum_{s=1}^m v_s^j)v_l^i = 0, \quad i,j = 1,\ldots,n,\ l = 1,\ldots,m; \tag{3.11}$$

- From (3.6d), we have $\boldsymbol{Y} \in \mathcal{N}^{N+1}$, consequently,

$$\lambda_1 u_l^i + \lambda_2 v_l^i \geqslant 0, \quad i = 1,\ldots,n,\ l = 1,\ldots,m, \tag{3.12}$$

and

$$\lambda_1 u_l^i u_s^j + \lambda_2 v_l^i v_s^j \geqslant 0, \quad i,j = 1,\ldots,n,\ l,s = 1,\ldots,m. \tag{3.13}$$

Following (3.9) and (3.11) under $j = i$, we have, for $i = 1,\ldots,n$,

$$\begin{aligned}
& \lambda_1(1-\sum_{l=1}^m u_l^i)^2 + \lambda_2(1-\sum_{l=1}^m v_l^i)^2 \\
= & \left[\lambda_1(1-\sum_{l=1}^m u_l^i) + \lambda_2(1-\sum_{l=1}^m v_l^i)\right] - \sum_{s=1}^m\left[\lambda_1(1-\sum_{l=1}^m u_l^i)u_s^i + \lambda_2(1-\sum_{l=1}^m v_l^i)v_s^i\right] \\
= & 0 - \sum_{s=1}^m 0 \\
= & 0,
\end{aligned}$$

which leads to

$$1-\sum_{l=1}^m u_l^i = 1-\sum_{l=1}^m v_l^i = 0, \quad i = 1,\ldots,n. \tag{3.14}$$

Next we discuss the structure of $\boldsymbol{u}^i$ and $\boldsymbol{v}^i$ for fixed $i \in \{1,\ldots,n\}$. According to (3.10), we know $u_l^i \in \{0,1\}$ if and only if $v_l^i \in \{0,1\}$, $l = 1,\ldots,m$. If $\boldsymbol{u}^i \notin \{0,1\}^m$, there exist $u_{p(i)}^i, v_{p(i)}^i \notin \{0,1\}$ for some index $p(i) \in \{1,\ldots,m\}$. From (3.10), we further know that one of $u_{p(i)}^i$ and $v_{p(i)}^i$ falls into the interval of $(0,1)$. Since $\sum_{l=1}^m u_l^i = 1$, there exists an index $q(i) \neq p(i)$ such that $u_{q(i)}^i \notin \{0,1\}$ and $v_{q(i)}^i \notin \{0,1\}$.

Following (3.10), (3.11) under $j = i$ and (3.13), we have

$$\lambda_1 u_s^i u_l^i + \lambda_2 v_s^i v_l^i = 0, \quad i = 1,\ldots,n,\ s \neq l. \tag{3.15}$$

Plugging $l = p(i)$ and $s = q(i)$ into (3.15), we have $(u^i_{p(i)}v^i_{p(i)})(u^i_{q(i)}v^i_{q(i)}) < 0$. W.l.o.g, we may assume that $u^i_{p(i)}v^i_{p(i)} < 0$ and $u^i_{q(i)}v^i_{q(i)} > 0$. From (3.15), $u^i_{p(i)}v^i_{p(i)} < 0$ implies $u^i_l v^i_l \geqslant 0$ for any $l \neq p(i)$. According to (3.10), we further have $u^i_l \geqslant 0$ and $v^i_l \geqslant 0$ for any $l \neq p(i)$. Then $u^i_{p(i)} + u^i_{q(i)} \leqslant 1$ and $u^i_{p(i)} + u^i_{q(i)} \leqslant 1$ follow from (3.14). We obtain the following result by subtracting (3.15) from (3.10) with $l = p(i)$ and $s = q(i)$:

$$\lambda_1 u^i_{q(i)}(u^i_{p(i)} + u^i_{q(i)} - 1) + \lambda_2 v^i_{q(i)}(v^i_{p(i)} + v^i_{q(i)} - 1) = 0. \tag{3.16}$$

Since $u^i_{q(i)}v^i_{q(i)} > 0$, the addends in (3.16) have the same signs, which lead to

$$u^i_{p(i)} + u^i_{q(i)} = v^i_{p(i)} + v^i_{q(i)} = 1.$$

Then, for any $l \neq p(i)$ and $l \neq q(i)$, the nonnegativity of u^i_l and v^i_l leads to $u^i_l = v^i_l = 0$. From the above discussions, we conclude that one of the following two situations occurs:

1.1 There exist $t(i), t'(i) \in \{1, \ldots, m\}$ such that $u^i_{t(i)} = v^i_{t'(i)} = 1$, and $u^i_l = v^i_{l'} = 0$ for any $l \neq t(i), l' \neq t'(i)$. In this case, $\boldsymbol{u}^i, \boldsymbol{v}^i \in \{0, 1\}^m$.

1.2 There exist $p(i), q(i) \in \{1, \ldots, m\}$ and $p(i) \neq q(i)$ such that $u^i_{p(i)} + u^i_{q(i)} = v^i_{p(i)} + v^i_{q(i)} = 1$ and $u^i_l = v^i_l = 0$ for any $l \neq p(i)$ and $l \neq q(i)$. Particularly, $u^i_{q(i)}v^i_{q(i)} > 0$ and $u^i_{p(i)}v^i_{p(i)} < 0$.

If all the indices of $i = 1, \ldots, n$ fall in situation **1.1**, then $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{F}_P$. By taking $\bar{\lambda} = \lambda_1, \bar{\boldsymbol{u}} = \boldsymbol{u}$, and $\bar{\boldsymbol{v}} = \boldsymbol{v}$, we get the desired result of the theorem. Otherwise, there exists an index $j \in \{1, \ldots, n\}$ that falls in situation **1.2**. Since $u^j_{p(j)}v^j_{p(j)} < 0$, w.l.o.g, we may assume that $u^j_{p(j)} > 0$ and $v^j_{p(j)} < 0$. Then, we can claim the following two results:

(i) $t(i) = t'(i)$ holds for any index i falling in situation **1.1**.

(ii) $u^i_{p(i)} = u^j_{p(j)}$ and $v^i_{p(i)} = v^j_{p(j)}$ hold for any index $i \neq j$ falling in situation **1.2**.

To show (i), suppose that $t(i) \neq t'(i)$ for some index i falling in situation **1.1**. Then $u^i_{t'(i)} = 0, v^i_{t'(i)} = 1$ and

$$\lambda_1 u^i_{t'(i)}u^j_{p(j)} + \lambda_2 v^i_{t'(i)}v^j_{p(j)} = \lambda_2 v^j_{p(j)} < 0,$$

which contradicts (3.13) under $l = t'(i)$ and $s = p(j)$.

To show (ii), we consider any index $i \neq j$ falling in situation **1.2**. Plugging $l = p(i), s = p(j)$ into (3.13), we have

$$\lambda_1 u^i_{p(i)}u^j_{p(j)} + \lambda_2 v^i_{p(i)}v^j_{p(j)} \geqslant 0. \tag{3.17}$$

Since $u^i_{p(i)}v^i_{p(i)} < 0$, we see that $u^i_{p(i)} > 0$ and $v^i_{p(i)} < 0$ from (3.17). We further derive $\frac{u^i_{p(i)}}{v^i_{p(i)}} \leqslant -\frac{\lambda_2 v^j_{q(j)}}{\lambda_1 u^j_{q(j)}} = \frac{u^j_{p(j)}}{v^j_{p(j)}} \leqslant -\frac{\lambda_2 v^i_{q(i)}}{\lambda_1 u^i_{q(i)}} = \frac{u^i_{p(i)}}{v^i_{p(i)}}$, by plugging $l = p(i), s = q(j)$ into (3.13) for the first

inequality and $l = q(i), s = p(j)$ into (3.13) for the second inequality, respectively, and using (3.15) for the equalities. Consequently, we have $\frac{u^i_{p(i)}}{v^i_{p(i)}} = \frac{u^j_{p(j)}}{v^j_{p(j)}}$. Under $l = p(i)$, (3.10) can be rewritten as an equation of $u^i_{p(i)}$ and the ratio $\frac{u^i_{p(i)}}{v^i_{p(i)}}$, which implies that the value of $u^i_{p(i)}$ is determined by $\frac{u^i_{p(i)}}{v^i_{p(i)}}$. Thus $u^i_{p(i)} = u^j_{p(j)}$ and then $v^i_{p(i)} = v^j_{p(j)}$.

Now we can construct $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}} \in \{0,1\}^N$ by setting $\bar{u}^i_l = \bar{v}^i_{l'} = 1$ if and only if $l = l' = t(i)$ for index i falling in situation **1.1** or $l = p(i), l' = q(i)$ for index i falling in situation **1.2**. By taking $\bar{\lambda} = \lambda_1 u^j_{p(j)} + \lambda_2 v^j_{p(j)}$, where j is the index that falls in situation **1.2**, we have the desired result of the theorem.

Case II. $t_1 t_2 = 0$:

W.l.o.g, we may assume that $t_1 = 1, t_2 = 0$. Similar to ***Case I***, for any index i, one of the following two situations holds:

2.1 There exists a $t(i) \in \{1, \ldots, m\}$ such that $u^i_{t(i)} = 1$ and $u^i_l = 0$ for any $l \neq t(i)$; meanwhile, $v^i_l = 0$ for $l = 1, \ldots, m$. In this case, $\boldsymbol{u}^i \in \{0,1\}^m$ and $\boldsymbol{v}^i = \boldsymbol{0}_{m\times 1}$.

2.2 There exist $p(i), q(i) \in \{1, \ldots, m\}$ and $p(i) \neq q(i)$ such that $0 < u^i_{p(i)} \leqslant u^i_{q(i)} < 1$, $u^i_{p(i)} + u^i_{q(i)} = 1$ and $u^i_l = 0$ for any $l \neq p(i)$ and $l \neq q(i)$; meanwhile, $v^i_l = \pm\sqrt{u^i_l - (u^i_l)^2}$ for $l = 1, \ldots, m$.

If all of the indices $i = 1, \ldots, n$ fall in situation **2.1**, then $\boldsymbol{u} \in \mathcal{F}_P$ and $\boldsymbol{v} = \boldsymbol{0}_{N\times 1}$. This contradicts the fact of $\mathrm{rank}(\boldsymbol{Y}) = 2$. Thus there must exist an index j in situation **2.2**. For any index $i \neq j$ that also falls in situation **2.2**, we can then show that $v^i_{p(i)} = v^j_{p(j)}$.

Like in ***Case I***, we can construct $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}} \in \{0,1\}^N$ by setting $\bar{u}^i_l = \bar{v}^i_{l'} = 1$ if and only if $l = l' = t(i)$ for index i falling in situation **2.1** or $l = p(i), l' = q(i)$ for index i falling in situation **2.2**. By taking $\bar{\lambda} = v^j_{p(j)}$, where j is the index that falls in situation **2.2**, we have the desired result of the theorem. □

Remark 1. We see that constraints (3.6c) under $j \neq i$ are not required in the proof of Theorem 3.3.4. Consequently, the rank-two property holds for a more general relaxation problem in the following form:

$$
\begin{aligned}
\min \quad & \frac{1}{2}\begin{pmatrix} 0 & \boldsymbol{q}^T \\ \boldsymbol{q} & \boldsymbol{Q} \end{pmatrix} \bullet \boldsymbol{Y} \\
\text{s.t.} \quad & \sum_{l=1}^{m} u^i_l = 1, \quad i = 1, \ldots, n, \\
& U^{ii}_{ll} = u^i_l, \quad i = 1, \ldots, n,\ l = 1, \ldots, m,
\end{aligned}
$$

$$\sum_{l\neq s}^{m} U_{ls}^{ii} = 0, \quad i = 1, \ldots, n,$$
$$\boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix} \in \mathcal{S}_+^{N+1} \cap \mathcal{N}^{N+1}.$$

Based on Theorems 3.3.3 and 3.3.4, we have the following sufficient condition that guarantees the zero gap between (LCoR$_2$) and (LCoP).

Theorem 3.3.5. *If (LCoR$_2$) has an optimal solution $\boldsymbol{Y}^*$ with rank$(\boldsymbol{Y}^*) \leqslant 2$, then $\boldsymbol{Y}^*$ solves (LCoP) with $z_2 = z_0$. In this case, the optimal solutions of the original problem (DQP) can be explicitly generated from $\boldsymbol{Y}^*$.*

Proof. According to Theorems 3.3.3 and 3.3.4, $\boldsymbol{Y}^*$ is also feasible for (LCoP). Since the objective value of $\boldsymbol{Y}^*$ in (LCoP) is equal to z_2, we have $z_2 \geqslant z_0$. Notice that z_2 is a lower bound for z_0, hence $z_2 = z_0$.

If rank$(\boldsymbol{Y}^*) = 1$, the vector $\boldsymbol{u} \in \mathcal{F}_P$ generated by (3.7) is an optimal solution of (P). If rank$(\boldsymbol{Y}^*) = 2$, according to the proof of Theorem 3.3.4, we can explicitly generate the vectors $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}} \in \mathcal{F}_P$ satisfying (3.8). Therefore, we can generate an optimal solution $\boldsymbol{x}^*$ of (DQP) from the optimal solution of (P) using the linear transformation (3.3). □

In summary, given a problem (DQP), we may solve (LCoR$_2$) using an SDP solver to obtain its optimal solution $\boldsymbol{Y}^*$ and the optimal value z_2. If rank$(\boldsymbol{Y}^*) \leqslant 2$, we find an optimal solution of (DQP) and the optimal value of (DQP) is equal to z_2. Otherwise, we return z_2 as a lower bound for the optimal value of (DQP). Next, we present an example to show that optimal solutions of the original DQP problem can be explicitly generated from a rank-two optimal solution of the proposed relaxation problem.

Example 3.3.1. Consider the following example:

$$\begin{aligned} \min \quad & x_1^2 - 1.5x_2^2 - 0.5x_3^2 - 2x_1x_2 - 2x_2x_3 + 3x_1 + 3x_2 - 2x_3 \\ \text{s.t.} \quad & x_1 \in \{5, 6, 7, 12\}, \\ & x_2 \in \{1, 3, 5, 8\}, \\ & x_3 \in \{4, 7, 9, 11\}. \end{aligned}$$

By solving the corresponding linear conic relaxation problem (LCoR$_2$) using an SDP solver, we obtain an optimal solution

$$\boldsymbol{Y}^* = \begin{pmatrix} 1 & 0 & 0.47 & 0.53 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.47 & 0 & 0.47 & 0 & 0 & 0 & 0 & 0 & 0.47 & 0 & 0 & 0 & 0.47 \\ 0.53 & 0 & 0 & 0.53 & 0 & 0 & 0 & 0 & 0.53 & 0 & 0 & 0 & 0.53 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0.47 & 0.53 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0.47 & 0.53 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

with an objective value of -372.5. Because $\text{rank}(\boldsymbol{Y}^*) = 2$, the known rank-one result does not apply here. However, Theorem 3.3.4 generates $\bar{\boldsymbol{u}} = (0,1,0,0,0,0,0,1,0,0,0,1)$ and $\bar{\boldsymbol{v}} = (0,0,1,0,0,0,0,1,0,0,0,1)$ which are optimal solutions to the corresponding problem (P) of Example 3.3.1. Using (3.3), we find $(6,8,11)$ and $(7,8,11)$ to be optimal solutions of Example 3.3.1 with an optimal objective value of -372.5.

3.4 Effectiveness and robustness of lower bounds

In this section, numerical experiments are conducted to test the effectiveness and robustness of the lower bounds provided by the proposed linear conic relaxation of DQP problems. To address this issue, we compare the lower bounds z_1 and z_2 with the optimal value of (DQP) by solving (LCoR$_1$) and (LCoR$_2$), respectively. The mixed integer linear programming (MILP) reformulation of [46, 89] is used to find the exact optimal value z_0 of the original problem (DQP).

All of our experiments were run on a personal computer with 2.50 GHz Intel Core i7-3537U CPU and 8Gb of RAM. We used the MATLAB R2013a-based SeDuMi [82] solver with a feasibility/optimality tolerance of 1e-9 to solve semidefinite programming problems and CPLEX solver with a feasibility/optimality tolerance of 1e-6 to solve mixed integer linear programming problems.

A set of (DQP) instances with various values of n and m were tested. For each test problem, d_l^i are different integers randomly generated from the uniform distribution over $[1, 3m]$ for $l = 1, \ldots, m,\ i = 1, \ldots, n$. The linear coefficient vector $\boldsymbol{c} \in \mathbb{R}^n$ of the quadratic function was randomly generated from the uniform distribution over $[-1, 1]^n$. We generated the quadratic coefficient matrix by $\boldsymbol{A} = \mathbf{P}\boldsymbol{\Lambda}\mathbf{P}^T \in \mathcal{S}^n$, where $\boldsymbol{\Lambda} \in \mathcal{S}^n$ is a diagonal matrix with the eigenvalues of $\boldsymbol{A}$ on its diagonal positions and $\mathbf{P} \in \mathcal{S}^n$ is an orthogonal matrix. The positive and negative eigenvalues of $\boldsymbol{A}$ were randomly generated from the uniform distribution over $[0, 1]$ and $[-1, 0]$, respectively. The orthogonal matrix $\mathbf{P}$ was obtained by the QR decomposition of a matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$, which was generated from the normal distribution over $[0, 1]^{n\times n}$. We chose all the eigenvalues to be positive for constructing convex quadratic functions, and select half of the

eigenvalues to be negative for non-convex quadratic functions. Fifty instances were randomly generated for each combination of n and m.

We tested the problem (DQP) of size $n, m = 6, 8, 10$, respectively. Notice that we need the exact optimal value z_0 to evaluate the effectiveness of the lower bounds. The largest problem size of our experiments is limited to $n = 8, m = 10$ by the difficulty in obtaining z_0 by solving the MILP reformulation involving 10^8 possible combinations. We conducted experiments on (DQP) instances with convex and non-convex quadratic objective functions, respectively. The average values and standard deviations of the gaps are shown in Tables 3.1 and 3.2 with the following notations:

$$\text{relgapP}_1 = \left|\frac{\text{z}_1 - \text{z}_0}{\text{z}_0}\right|, \quad \text{absgapP}_1 = |\text{z}_1 - \text{z}_0|\,;$$
$$\text{relgapP}_2 = \left|\frac{\text{z}_2 - \text{z}_0}{\text{z}_0}\right|, \quad \text{absgapP}_2 = |\text{z}_2 - \text{z}_0|\,.$$

Table 3.1: (DQP) with convex quadratic objective function

		z_0		absgapP_1		relgapP_1		absgapP_2		relgapP_2	
n	m	ave	std	ave	std	ave	std	ave	std	ave	std
6	6	13.65	9.61	150.04	138.05	22.76	40.34	0.00	0.00	0.00	0.00
6	8	15.86	13.71	380.92	240.07	110.33	271.99	0.00	0.00	0.00	0.00
6	10	17.05	16.44	845.33	501.61	451.20	1402.15	0.00	0.00	0.00	0.00
8	6	21.36	13.01	233.17	119.62	45.42	187.38	0.00	0.00	0.00	0.00
8	8	22.12	15.62	661.16	351.22	57.66	87.32	0.00	0.00	0.00	0.00
8	10	20.28	14.90	1357.76	718.65	214.30	515.50	0.00	0.00	0.00	0.00

Table 3.2: (DQP) with non-convex quadratic objective function

		z_0		absgapP_1		relgapP_1		absgapP_2		relgapP_2	
n	m	ave	std	ave	std	ave	std	ave	std	ave	std
6	6	196.67	127.07	436.89	198.14	3.83	5.14	0.00	0.00	0.00	0.00
6	8	395.20	250.03	1093.98	485.16	4.11	4.91	0.07	0.49	0.00	0.00
6	10	657.75	424.55	2438.59	1136.52	5.24	4.98	0.00	0.00	0.00	0.00
8	6	264.71	141.37	652.16	228. 46	4.07	6.90	0.01	0.00	0.00	0.00
8	8	529.32	263.50	1814.21	587.90	4.09	1.91	0.00	0.00	0.00	0.00
8	10	791.09	431.35	3340.62	1245.05	5.02	1.99	0.06	0.27	0.00	0.00

From Tables 3.1 and 3.2, we have some observations on the lower bounds z_1 and z_2 depending on the convexity of objective function of (DQP):

1. The relative and absolute gaps with respect to z_1 are large in both convex and non-convex cases. Therefore, the relaxation (LCoR_1) is not tight for general instances.

2. The gaps with respect to z_2 are robust and close to zero in both convex and non-convex cases. This indicates that the proposed RLT equality cuts are effective for tightening (LCoR_1). Solving the proposed linear conic relaxation (LCoR_2) can provide a high-quality and robust lower bound for (DQP).

3. Regardless of whether the objective function of (DQP) is convex or not, as m or n increases, the relative gap between z_1 and z_0 becomes larger. It fits our general understanding that the tractable convex relaxations of the cone $\mathcal{D}^*(\mathcal{F}_P)$ are less accurate for larger-size problems.

4. Both of the lower bounds z_1 and z_2 are more effective and robust for (DQP) with convex rather than non-convex objective functions. Since the proposed approach is one kind of convex relaxation, it is reasonable that both of the relaxations (LCoR_1) and (LCoR_2) may perform better in the convex case than the non-convex case.

As to the computational efficiency of the proposed relaxation (LCoR_2), we present the computational time t_2 of solving (LCoR_2) in Table 3.3. We see that in either the convex case or the non-convex case the computational efficiency of obtaining high-quality lower bounds by solving (LCoR_2) is reasonable and stable.

Table 3.3: Computational time t_2 (seconds) of solving (LCoR_2)

		Convex		Nonconvex	
n	m	ave	std	ave	std
6	6	8.19	1.46	5.08	2.27
6	8	34.85	4.80	20.87	5.75
6	10	118.01	18.13	71.42	11.26
8	6	32.08	3.84	30.75	7.31
8	8	158.43	15.59	121.80	25.78
8	10	669.79	339.58	417.08	63.39

3.5 Extensions to constrained DQP problems

From Section 3.3, we know that the rank-two sufficient condition holds for (LCoR_2) to guarantee the zero gap. Numerical results in Section 3.4 further illustrate the effectiveness and robustness of the lower bounds provided by (LCoR_2). Notice that problem (DQP) has no explicit constraints except each variable being restricted to a finite set of discrete values. In this subsection, we would like to extend the proposed linear conic relaxation framework to the discrete quadratic program with linear or quadratic constraints.

First, we consider the following linearly constrained discrete quadratic programming problem:

$$
\begin{aligned}
(\mathrm{DQP_L}) \qquad & \min \quad \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\
& \text{s.t.} \quad \boldsymbol{b}_s^T\boldsymbol{x} \geqslant L_s, \quad s = 1, \ldots, S, && (3.18a) \\
& \qquad\; x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, && (3.18b)
\end{aligned}
$$

where $\boldsymbol{A} \in \mathcal{S}^n$, $\boldsymbol{c}, \boldsymbol{b}_s \in \mathbb{R}^n$ and $L_s \in \mathbb{R}$ for $s = 1, \ldots, S$. Following the approach introduced in Section 3.1, we can reformulate ($\mathrm{DQP_L}$) as

$$
\begin{aligned}
(\mathrm{DQP_{L1}}) \qquad & \min \quad \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q}^T\boldsymbol{u} \\
& \text{s.t.} \quad (3.5a), (3.5b), \\
& \qquad\; \boldsymbol{g}_s^T\boldsymbol{u} \geqslant L_s, \quad s = 1, \ldots, S, && (3.19)
\end{aligned}
$$

where $\boldsymbol{Q} \in \mathcal{S}^N$ and $\boldsymbol{q}, \boldsymbol{g}_s \in \mathbb{R}^N$, $s = 1, \ldots, S,\ i = 1, \ldots, n$. Based on the relaxation approach discussed in Section 3.2, we may generate additional RLT inequalities from the inequality constraints and choose the non-redundant ones as valid cuts. Then the proposed linear conic relaxation for ($\mathrm{DQP_{L1}}$) becomes

$$
\begin{aligned}
(\mathrm{DQP_{L2}}) \qquad & \min \quad \frac{1}{2}\boldsymbol{Q} \bullet \boldsymbol{U} + \boldsymbol{q}^T\boldsymbol{u} \\
& \text{s.t.} \quad (3.6a) - (3.6d), \\
& \qquad\; \boldsymbol{g}_s^T\boldsymbol{u} \geqslant L_s, \quad s = 1, \ldots, S, && (3.20a) \\
& \qquad\; (\boldsymbol{g}_s\boldsymbol{e}_j^T) \bullet \boldsymbol{U} - L_s\boldsymbol{e}_j^T\boldsymbol{u} \geqslant 0, \quad s = 1, \ldots, S,\ j = 1, \ldots, N, && (3.20b) \\
& \qquad\; -(\boldsymbol{g}_s\boldsymbol{e}_j^T) \bullet \boldsymbol{U} + (L_s\boldsymbol{e}_j + \boldsymbol{g}_s)^T\boldsymbol{u} \geqslant L_s, \quad s = 1, \ldots, S,\ j = 1, \ldots, N. && (3.20c)
\end{aligned}
$$

Notice that (3.6a)-(3.6d) define the feasible domain of (LCoR_2). From Theorem 3.3.4, we further

extend the rank-two property to (DQP$_{\rm L2}$).

Theorem 3.5.1. *If (DQP$_{\rm L2}$) has a rank-two optimal solution, then the gap between (DQP$_{\rm L2}$) and (DQP$_{\rm L}$) is zero. In this case, the optimal solution of (DQP$_{\rm L}$) can be explicitly generated.*

Proof. Assume that $\boldsymbol{Y} = \begin{pmatrix} 1 & \boldsymbol{u}^T \\ \boldsymbol{u} & \boldsymbol{U} \end{pmatrix}$ is a rank-two optimal solution of (DQP$_{\rm L2}$). Since $\text{rank}(\boldsymbol{Y}) = 2$ and $\boldsymbol{Y}$ satisfies (3.6a)-(3.6d), Theorem 3.3.4 explicitly generates $\bar{\lambda} \in (0,1), \bar{\boldsymbol{u}}, \bar{\boldsymbol{v}} \in \mathbb{R}^N$ such that

$$\boldsymbol{Y} = \bar{\lambda} \begin{pmatrix} 1 \\ \bar{\boldsymbol{u}} \end{pmatrix} \begin{pmatrix} 1 \\ \bar{\boldsymbol{u}} \end{pmatrix}^T + (1-\bar{\lambda}) \begin{pmatrix} 1 \\ \bar{\boldsymbol{v}} \end{pmatrix} \begin{pmatrix} 1 \\ \bar{\boldsymbol{v}} \end{pmatrix}^T, \tag{3.21}$$

and $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ satisfy (3.5a) and (3.5b). We now show that $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ satisfy constraints (3.19). Plugging $\boldsymbol{u} = \bar{\lambda}\bar{\boldsymbol{u}} + (1-\bar{\lambda})\bar{\boldsymbol{v}}$ and $\boldsymbol{U} = \bar{\lambda}\bar{\boldsymbol{u}}\bar{\boldsymbol{u}}^T + (1-\bar{\lambda})\bar{\boldsymbol{v}}\bar{\boldsymbol{v}}^T$ into (3.20b)-(3.20c), we have

$$\bar{\lambda}\bar{\boldsymbol{u}}_j(\boldsymbol{g}_s^T\bar{\boldsymbol{u}} - L_s) + (1-\bar{\lambda})\bar{\boldsymbol{v}}_j(\boldsymbol{g}_s^T\bar{\boldsymbol{v}} - L_s) \geqslant 0, \tag{3.22}$$

$$\bar{\lambda}(1-\bar{\boldsymbol{u}}_j)(\boldsymbol{g}_s^T\bar{\boldsymbol{u}} - L_s) + (1-\bar{\lambda})(1-\bar{\boldsymbol{v}}_j)(\boldsymbol{g}_s^T\bar{\boldsymbol{v}} - L_s) \geqslant 0, \tag{3.23}$$

for $j = 1, \ldots, N$ and $s = 1, \ldots, S$. Because the 0-1 vectors $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ satisfy (3.2a) and $\bar{\boldsymbol{u}} \neq \bar{\boldsymbol{v}}$, there exists an index $j^* \in \{1, \ldots, N\}$ such that $\bar{\boldsymbol{u}}_{j^*} = 1$ and $\bar{\boldsymbol{v}}_{j^*} = 0$. Under $j = j^*$, (3.22) and (3.23) are equivalent to $(\boldsymbol{g}_s^T\bar{\boldsymbol{u}} - L_s) \geqslant 0$ and $(\boldsymbol{g}_s^T\bar{\boldsymbol{v}} - L_s) \geqslant 0$, respectively, for $s = 1, \ldots, S$. Therefore, we know that $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ are feasible solutions of (DQP$_{\rm L1}$). Notice that $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ and $\boldsymbol{Y}$ share the same objective value in (DQP$_{\rm L1}$) and (DQP$_{\rm L2}$), respectively. Due to the optimality of $\boldsymbol{Y}$, its objective value is a lower bound for (DQP$_{\rm L1}$). Consequently, $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ are optimal solutions of (DQP$_{\rm L1}$). In this case, the corresponding solutions of (DQP$_{\rm L}$) are optimal and there is no gap between (DQP$_{\rm L2}$) and (DQP$_{\rm L}$). □

Theorem 3.5.1 shows that the rank-two property holds for (DQP$_{\rm L2}$). However, when quadratic constraints are added to (DQP$_{\rm L}$), the rank-two property may no longer hold for the corresponding linear conic relaxation. Consider the following quadratically constrained discrete quadratic optimization problem:

$$\begin{aligned} & & \min \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \\ & (\text{DQP}_{\rm Q}) & \text{s.t.} \quad & (3.18a), (3.18b), \\ & & & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{A}_t\boldsymbol{x} + \boldsymbol{c}_t^T\boldsymbol{x} \geqslant R_t, \quad t = 1, \ldots, T, \end{aligned} \tag{3.24}$$

where $\boldsymbol{A}, \boldsymbol{A}_t \in \mathcal{S}^n$, $\boldsymbol{A}_t \neq \boldsymbol{0}_{n\times n}$, $\boldsymbol{c}, \boldsymbol{c}_t \in \mathbb{R}^n$ and $R_t \in \mathbb{R}$ for $t = 1, \ldots, T$. We propose the following 0-1 reformulation (DQP$_{\rm Q1}$) and its linear conic relaxation (DQP$_{\rm Q2}$):

$$
\begin{aligned}
& & \min \quad & \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}\boldsymbol{u} + \boldsymbol{q}^T\boldsymbol{u} \\
& (\mathrm{DQP}_{\mathrm{Q1}}) & \text{s.t.} \quad & (3.5a), (3.5b), (3.19), \\
& & & \frac{1}{2}\boldsymbol{u}^T\boldsymbol{Q}_t\boldsymbol{u} + \boldsymbol{q}_t^T\boldsymbol{u} \geqslant R_t, \quad t = 1, \ldots, T,
\end{aligned} \tag{3.25}
$$

where $\boldsymbol{Q}, \boldsymbol{Q}_t \in \mathcal{S}^N$ and $\boldsymbol{q}, \boldsymbol{q}_t \in \mathbb{R}^N$, $t = 1, \ldots, T$;

$$
\begin{aligned}
& & \min \quad & \frac{1}{2}\boldsymbol{Q} \bullet \boldsymbol{U} + \boldsymbol{q}^T\boldsymbol{u} \\
& (\mathrm{DQP}_{\mathrm{Q2}}) & \text{s.t.} \quad & (3.6a) - (3.6d), (3.20a) - (3.20c), \\
& & & \frac{1}{2}\boldsymbol{Q}_t \bullet \boldsymbol{U} + \boldsymbol{q}_t^T\boldsymbol{u} \geqslant R_t, \quad t = 1, \ldots, T.
\end{aligned} \tag{3.26}
$$

Notice that the quadratic constraints (3.25) are relaxed to (3.26) in ($\mathrm{DQP}_{\mathrm{Q2}}$), but there is no related RLT constraint to tighten the relaxation. Following the proof of Theorem 3.5.1, given any rank-two feasible solution of ($\mathrm{DQP}_{\mathrm{Q2}}$), we can generate $\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}$ that satisfy the constraints (3.6a)-(3.6d) and (3.20a)-(3.20c). Plugging the decomposition formula (3.21) into (3.26), we have

$$
\bar{\lambda}(\frac{1}{2}\bar{\boldsymbol{u}}^T\boldsymbol{Q}_t\bar{\boldsymbol{u}} + \boldsymbol{q}_t^T\bar{\boldsymbol{u}}) + (1-\bar{\lambda})(\frac{1}{2}\bar{\boldsymbol{v}}^T\boldsymbol{Q}_t\bar{\boldsymbol{v}} + \boldsymbol{q}_t^T\bar{\boldsymbol{v}}) \geqslant R_t, \quad t = 1, \ldots, T,
$$

which do not lead to $\frac{1}{2}\bar{\boldsymbol{u}}^T\boldsymbol{Q}_t\bar{\boldsymbol{u}} + \boldsymbol{q}_t^T\bar{\boldsymbol{u}} \geqslant R_t$ or $\frac{1}{2}\bar{\boldsymbol{v}}^T\boldsymbol{Q}_t\bar{\boldsymbol{v}} + \boldsymbol{q}_t^T\bar{\boldsymbol{v}} \geqslant R_t$. Consequently, we may not generate feasible solutions of ($\mathrm{DQP}_{\mathrm{Q1}}$) from the rank-two feasible solution of ($\mathrm{DQP}_{\mathrm{Q2}}$).

Nevertheless, if we obtain a rank-two solution $\boldsymbol{Y}^*$ by solving the relaxation problem ($\mathrm{DQP}_{\mathrm{Q2}}$) via an SDP solver, the corresponding $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ which can be explicitly generated using the proof of Theorem 3.3.4 are still useful. The best situation is that both of $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ satisfy the quadratic constraints (3.25), under which they become optimal solutions of ($\mathrm{DQP}_{\mathrm{Q1}}$). If one of $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ is feasible to (3.25), the feasible one may be close to an optimal solution of ($\mathrm{DQP}_{\mathrm{Q1}}$). If none of $\bar{\boldsymbol{u}}$ and $\bar{\boldsymbol{v}}$ satisfies the quadratic constraints, they can be used as starting solutions for additional processing.

3.6 Summary

In this chapter, we have proposed an RLT-based linear conic relaxation for solving discrete quadratic optimization problems. By analyzing the feasible domain of the proposed relaxation problem, we further showed that if it has a rank-two optimal solution, then the gap between the proposed relaxation and the original DQP becomes zero and we can explicitly derive optimal

solutions to the original DQP problem. We have also conducted numerical experiments on randomly generated DQP problems. The results indicate that the proposed linear conic relaxation can provide robust high-quality lower bounds for the original DQP problem. These bounds may be incorporated in a branch and bound scheme for providing optimal solutions to DQP problems in a more effective manner. The proposed linear conic relaxation approach and the rank-two property have been extended. We showed that the rank-two property holds for the corresponding linear conic relaxation of linearly constrained DQP problems.

DQP is a less studied problem. This chapter is the first to propose a special RLT-based linear conic relaxation for DQP. Numerical results have shown that the proposed relaxation can provide tight and robust lower bounds. We have derived a rank-two property for explicitly generating optimal solutions of DQP. The rank-two property has also been extended to DQP problems with linear constraints.

Chapter 4

A Linearization Method

In this chapter, we focus on the linearization methods for providing effective 0-1 mixed-integer linear programming reformulations of DQP. A Super Logarithmic Method is first proposed to reformulate a signomial programming program with discrete variables (DSP), which includes DQP as a special case. The proposed method requires a minimum number of binary variables and inequality constraints to reformulate DSP as a 0-1 mixed integer linear programming problem. Both the theoretical analysis and numerical results strongly support its superior performance compared with other state-of-the-art linearization methods. In addition, we provide a class of representable programming problems that can be linearly reformulated by the proposed method. At last, the application of the proposed Super Logarithmic Method on linearizing DQP is presented.

4.1 Introduction

Consider the following signomial programming problem with discrete variables:

$$\text{(DSP)} \qquad \begin{array}{ll} \min & \sum_{j=1}^{J} \left(c_j \prod_{i=1}^{n} (x_i)^{b_{i,j}} \right) \\ \text{s.t.} & x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, \end{array} \tag{4.1}$$

where $c_j, b_{i,j} \in \mathbb{R}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, J$. When each signomial term only involves two variables and 0-1 valued exponents, (DSP) reduces to a DQP problem. The problem (DSP) occurs frequently in real-life applications such as engineering design [11, 15], resource allocation [3, 28, 42] and network planning [86].

For linearizing the problem (DSP) as a 0-1 mixed integer linear programming (MILP) problem, the key is the linearization of signomial terms with discrete variables. For the purpose of

illustration, we first consider the following general signomial term:

$$w = (x_1)^{a_1} \dots (x_n)^{a_n}, \tag{4.2}$$

in which discrete variables $x_i \in \{d_1^i, d_2^i, \dots, d_m^i\}$ with $0 \leq d_1^i \leq d_2^i \dots \leq d_m^i$ and real exponents $a_i \in \mathbb{R}, i = 1, \dots, n$.

The first technique for linearizing w can be traced back to [33, 41]. We present it as below.

Basic Method 1 :

$$x_i = \sum_{l=1}^{m} d_l^i u_l^i, \text{ for } i = 1, \dots, n, \tag{4.3a}$$

$$\sum_{l=1}^{m} u_l^i = 1, \text{ for } i = 1, \dots, n, \tag{4.3b}$$

$$x_i \geqslant 0, \boldsymbol{u}^i \in \{0,1\}^m, \text{ for } i = 1, \dots, n, \tag{4.3c}$$

$$\prod_{i=1}^{n} (d_{l(i)}^i)^{a_i} + M(\sum_{i=1}^{n} u_{l(i)}^i - n) \leqslant w, \text{ for } l(i) = 1, \dots, m, \ i = 1, \dots, n, \tag{4.3d}$$

$$w \leqslant M(n - \sum_{i=1}^{n} u_{l(i)}^i) + \prod_{i=1}^{n} (d_{l(i)}^i)^{a_i}, \text{ for } l(i) = 1, \dots, m, \ i = 1, \dots, n, \tag{4.3e}$$

where $M = \prod_{i=1}^{n} \max_{l=1,\dots,m} (d_l^i)^{a_i}$.

Basic Method 1 introduces m binary variables to exactly represent each discrete variable x_i, $i = 1, \dots, n$. Conditions (4.3a)-(4.3c) ensure that for each $i = 1, \dots, n$, $x_i = d_{l(i)}^i$ if and only if there exists an $l(i) \in \{1, \dots, m\}$ such that $u_{l(i)}^i = 1$ and $u_l^i = 0$, for $l \neq l(i)$. Inequalities (4.3d)-(4.3e) are the so called Big-M constraints [37, 90]. Here the constant M, denoted as the Big-M value, is an upper bound of w. The required number of Big-M constraints in Basic Method 1 is $2 \prod_{i=1}^{n} m$, which is twice of the total number of possible values that vector $\boldsymbol{x}$ may take.

Several studies reported that Big-M constraints are not tight in the continuous relaxation and thus the way of forming Big-M constraints significantly affects the computational effectiveness [9, 90]. The more Big-M constraints we have, the slower the MILP solvers function. A way to reduce the number of Big-M constraints in Basic Method 1 was investigated by [22, 77]. The corresponding linear system used for linearizing w is given below.

Basic Method 2 :

$(4.3a) - (4.3c)$,

$$v_1 = \sum_{l=1}^{m} (d_l^1)^{a_1} u_l^1, \tag{4.4a}$$

$$M_i(u_l^i - 1) + (d_l^i)^{a_i} v_{i-1} \leqslant v_i, \text{ for } l = 1, \ldots, m, \ i = 2, \ldots, n, \tag{4.4b}$$

$$v_i \leqslant M_i(1 - u_l^i) + (d_l^i)^{a_i} v_{i-1}, \text{ for } l = 1, \ldots, m, \ i = 2, \ldots, n, \tag{4.4c}$$

$$v_2, \ldots, v_n \geqslant 0, \tag{4.4d}$$

where $M_i = \prod_{j=1}^{i-1} \max_{l=1,\ldots,m} (d_l^j)^{a_j}$ for $i = 2, \ldots, n$ and $v_n \triangleq w$.

While keeping the same number of binary variables, Basic Method 2 introduces new continuous variables v_i to represent the sub-terms $w_i = (x_1)^{a_1} \ldots (x_i)^{a_i}$ of w, $i = 1, \ldots, n$. Based on (4.3a)-(4.3c), Big-M constraints (4.4b)-(4.4c) can guarantee that $v_i = (x_i)^{a_i} v_{i-1}$. Consequently, sub-terms w_i are inductively linearized in the sequence of $i = 1, \ldots, n$. In this way, the required number of Big-M constraints decreases to $2 \sum_{i=2}^{n} m$.

Recently, [2, 46, 49, 89] developed a logarithmic technique to further reduce the number of binary variables required for representing a discrete variable. The basic concept of the logarithmic method is that for any integer r with $1 \leqslant r \leqslant m$, the following expression is unique:

$$r = 1 + \sum_{k=1}^{h} 2^{k-1} \lambda_k, \tag{4.5}$$

where $h = \lceil \log_2 m \rceil$ and $\lambda_k \in \{0, 1\}$ for $k = 1, \ldots, h$. Consequently, the integer $r \in \{1, \ldots, m\}$ and the binary h-vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_h)$ have a one-to-one correspondence. For a given integer k with $1 \leqslant k \leqslant h$, let G_k be the subset of all indices $r \in \{1, \ldots, m\}$ with $\lambda_k = 1$ in (4.5). Then, the logarithmic method is formally presented as follows [46].

Theorem 4.1.1. *For a positive integer $m \in \mathbb{Z}$, let $h = \lceil \log_2 m \rceil$. An n-vector $\boldsymbol{\delta} = (\delta_1, \ldots, \delta_m)$ with $\delta_r \geqslant 0$ and $\sum_{r=1}^{m} \delta_r = 1$ is a binary vector, if there is a binary h-vector $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_h)$ satisfying the following linear system:*

$$\begin{aligned}
&\sum_{r \in G_k} \delta_r = \lambda_k, \quad k = 1, \ldots, h, \\
&\delta_r \geqslant 0, \quad r = 1, \ldots, m, \\
&\lambda_k \in \{0, 1\}, \quad k = 1, \ldots, h.
\end{aligned}$$

This technique was applied to improve Basic Method 2, leading to the following linear system with fewer binary variables:

Logarithmic Method :

$$(4.3a)-(4.3b),(4.4a)-(4.4d),$$

$$\sum_{l\in G_k} u_l^i = \lambda_k^i,\ \text{for } i=1,\ldots,n,\ k=1,\ldots,h, \tag{4.6a}$$

$$x_i, u_l^i \geqslant 0, \lambda_k^i \in \{0,1\},\ \text{for } l=1,\ldots,m,\ k=1,\ldots,h,\ i=1,\ldots,n, \tag{4.6b}$$

where $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ and the values of $g_{k,l}$ are binary numbers satisfying $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ for $l = 1, \ldots, m$.

In the Logarithmic Method, variables $\boldsymbol{u}^i$ that are used to linearly express x_i are no longer binary. For a given binary vector $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i)$, equality constraints (4.6a) imply that there exists an $l(i) \in \{1, \ldots, m\}$ such that $u_{l(i)}^i = 1$ and $u_l^i = 0$, for $l \neq l(i)$. Therefore, the number of binary variables in this method is reduced to $\sum_{i=1}^{n} \lceil \log_2 m \rceil$, a half order of $\left(\sum_{i=1}^{n} m \right)$ in Basic Method 2. Among the above three methods, the Logarithmic Method linearizes a signomial term w with the fewest binary variables.

Based on the Logarithmic Method, we consider to further improve the way of linearizing the signomial terms in the problem (DSP). The Big-M constraints are redefined in terms of vectors $\boldsymbol{\lambda}^i$ instead of $\boldsymbol{u}^i$. Consequently, the required number of Big-M constraints can be reduced to $2\sum_{i=1}^{n} \lceil \log_2 m \rceil$ for linearizing a single signomial term w, while that of binary variables remains to be $\sum_{i=1}^{n} \lceil \log_2 m \rceil$. Hence, we call the proposed method as the Super Logarithmic Method.

A 0-1 mixed integer linear programming (MILP) reformulation of (DSP) can be obtained by gathering all of the linear systems that represent different signomial terms. This allows us to further compare the Super Logarithmic Method with the Logarithmic Method in terms of the effectiveness for linearizing multiple signomial terms. An interesting observation is that the Super Logarithmic Method enables more variables and constraints to be shared by different linear systems. As a direct consequence, the 0-1 MILP reformulation of (DSP) using the Super Logarithmic Method may require much fewer Big-M constraints than using the Logarithmic Method.

Following this mechanism, we can use the Super Logarithmic Method to linearize more complicated problems involving product and fractional terms with free-sign discrete variables, which consist of a class of "representable programming problems" [40]. For linearizing "representable programming problems", our comparable results indicate that the Super Logarithmic Method requires fewer Big-M inequality constraints than the Logarithmic Method, and thus is expected to be more effective. Computational experiments on the large-size (DSP) problems and "representable programming problems" are conducted. The numerical results clearly show that the computational time for solving the corresponding 0-1 MILP reformulations using the Super Logarithmic Method is much less than using the Logarithmic Method.

At last, we apply the Super Logarithmic Method to effectively reformulate DQP as a 0-1 mixed-integer linear programming problem. Compared with the Logarithmic Method and a Conventional Method that follows the linearization of quadratic terms in 0-1 binary variables, the Super Logarithmic Method requires fewer inequality constraints, especially when the problem size m is large in practice. In addition, we propose to enhance the reformulation of DQP by the Super Logarithmic Method. We show that the Big-M constraints for different quadratic terms could be aggravated and may be further reduced by taking the problem coefficients' structure into consideration.

The rest of this chapter is organized as follows. Section 4.2 proposes the Super Logarithmic Method for treating a signomial term with discrete variables. In Section 4.3, the Super Logarithmic Method is compared with other methods for the effectiveness of linearizing a single signomial term as well as that of multiple signomial terms. Section 4.4 extends the Super Logarithmic Method to treat more complicated terms and summarizes the class of representable programming models. Numerical experiments on some engineering optimization problems are tested in Section 4.5. The application of the Super Logarithmic Method on reformulating DQP is presented in Section 4.6. Section 4.7 concludes this chapter.

This Chapter can be referred to a published paper [45]. My contribution in [45] is investigating the principle, conducting theoretical analysis and exploring extensions of the proposed linearization method, and also documenting the results in the format of a paper.

4.2 Proposed Super Logarithmic Method

In this section, we propose an effective method to linearize the signomial term w with discrete variables defined by (4.2). The proposed method intends to reduce the number of Big-M constraints in currently known methods.

To illustrate the idea, we first consider the following simple case with $n = 2$:

$$w = (x_1)^{a_1}(x_2)^{a_2},$$

where $x_i \in \{d_1^i, \ldots, d_m^i\}$ with $0 \leqslant d_1^i < \ldots < d_m^i$ and $a_i \in \mathbb{R}$, for $i = 1, 2$. Our procedure is to linearize the sub-terms $w_1 = (x_1)^{a_1}$ and then $w_2 = (x_1)^{a_1}(x_2)^{a_2}$ in sequence.

Recall that the Logarithmic Method uses $h = \lceil \log_2 m \rceil$ binary variables to indicate the index choice of each discrete variable x_i, $i = 1, 2$. The corresponding linear system is expressed as

$$x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \sum_{l=1}^{m} u_l^i = 1, \text{ for } i = 1, 2, \tag{4.7a}$$

$$\sum_{l \in G_k} u_l^i = \lambda_k^i, \text{ for } k = 1, \ldots, h, \ i = 1, 2, \tag{4.7b}$$

$$u_l^i \geqslant 0,\ \lambda_k^i \in \{0,1\},\ \text{for } l = 1,\ldots,m,\ k = 1,\ldots,h,\ i = 1,2, \tag{4.7c}$$

Firstly, w_1 can be represented by a continuous variable $v_1 = \sum_{l=1}^{m} (d_l^1)^{a_1} u_l^1$. Consequently, $w_2 = v_1(x_2)^{a_2}$ can be uniquely determined by the value of x_2. The Logarithmic Method uses the following Big-M constraints to guarantee that v_2 is a valid representation of w_2:

$$M(u_l^2 - 1) + (d_l^2)^{a_2} v_1 \leqslant v_2 \leqslant M(1 - u_l^2) + (d_l^2)^{a_2} v_1,\ \text{for } l = 1,\ldots,m, \tag{4.8}$$

where $M = \max_{l=1,\ldots,m} (d_l^1)^{a_1}$. Each of these m Big-M constraints is defined in terms of the continuous variable u_l^2, for $l = 1,\ldots,m$. However, as shown in (4.7a)-(4.7c), the index choice of x_2 is determined by binary variables λ_k^2, $k = 1,\ldots,h$. Hence, $w_2 = v_1(x_2)^{a_2}$ can actually be linearized in terms of $\lambda_1^2,\ldots,\lambda_h^2$ instead of $u_1^2,\ldots,u_m^2$. In this way, the corresponding Big-M constraints may be reduced.

With the linear system (4.7a)-(4.7c) and the continuous variable $v_1 = \sum_{l=1}^{m} (d_l^1)^{a_1} u_l^1$ in mind, the key of the proposed Super Logarithmic Method comes with the introduction of a new nonnegative vector $\boldsymbol{p} = (p_1,\ldots,p_m)$ for defining

$$v_2 = \sum_{l=1}^{m} (d_l^2)^{a_2} p_l. \tag{4.9}$$

To force v_2 to be a valid representation of $w_2 = v_1(x_2)^{a_2}$, $\boldsymbol{p}$ should be able to indicate the index choice of x_2 and yield the value of v_1 as well. The former implies that $\boldsymbol{p}$ has a unique nonzero element whose index corresponds with the value of x_2 while the latter requires the nonzero element takes the value of v_1, i.e.,

$$p_{l^*} = v_1 \text{ and } p_l = 0 \text{ for all } l \neq l^* \Longleftrightarrow x_2 = d_{l^*}^2 \text{ for some } l^* \in \{1,\ldots,m\}. \tag{4.10}$$

The next step is to linearly express $\boldsymbol{p}$ in terms of $\lambda_1^2,\ldots,\lambda_h^2$ and v_1. From (4.10), we see that vectors $\boldsymbol{p} \in \mathbb{R}_+^m$ and $\boldsymbol{u}^2 \in \mathbb{R}_+^m$ share the same structure, except that different values of respective nonzero elements are used. Referring to (4.7a)-(4.7c) that connect $\boldsymbol{u}^2$ with $\boldsymbol{\lambda}^2$, we propose the following linear system for $\boldsymbol{p}$ to satisfy:

$$\sum_{l=1}^{m} p_l = v_1, \tag{4.11a}$$

$$M(\lambda_k^2 - 1) + v_1 \leqslant \sum_{l \in G_k} p_l \leqslant M\lambda_k^2,\ \text{for } k = 1,\ldots,h, \tag{4.11b}$$

$$p_l \geqslant 0,\ \lambda_k^2 \in \{0,1\},\ \text{for } l = 1,\ldots,m,\ k = 1,\ldots,h. \tag{4.11c}$$

Combining (4.7a)-(4.7c) and (4.11a)-(4.11c), we are sure that (4.10) holds. Consequently, the continuous variable v_2 defined by (4.9) is a valid representation of w_2. Therefore, the proposed Super Logarithmic Method linearizes the term $w = (x_1)^{a_1}(x_2)^{a_2}$ with discrete variables x_1 and x_2 in the system of (4.7a)-(4.7c), (4.9) and (4.11a)-(4.11c), where expressions in (4.11b) are regarded as Big-M constraints.

For linearizing $w = (x_1)^{a_1}(x_2)^{a_2}$, the real difference between the Logarithmic Method and the Super Logarithmic Method happens in the second step of linearizing $w_2 = v_1(x_2)^{a_2}$, where v_1 is the continuous variable that represents $w_1 = (x_1)^{a_1}$. The Logarithmic Method requires $2m$ Big-M constraints, while the Super Logarithmic Method requires only $2\lceil \log_2 m \rceil$ Big-M constraints with additional m continuous variables and one equality constraint.

Now, we consider the linearization of a general signomial term $w = (x_1)^{a_1}(x_2)^{a_2}\dots(x_n)^{a_n}$ with discrete variables $x_1, \dots, x_n$ by the proposed Super Logarithmic Method. As before, the basic idea is to linearize the sub-terms $w_i = (x_1)^{a_1}\dots(x_i)^{a_i}$, $i = 1, \dots, n$ of w in sequence. Using the induction formula of $w_i = (x_i)^{a_i} w_{i-1}$, we introduce a non-negative vector $\boldsymbol{p}^i = (p_1^i, \dots, p_m^i)$ to linearly express w_i based on the linearization of w_{i-1} for $i = 2, \dots, n$. A key proposition is presented below.

Proposition 4.2.1. *Given $0 \leqslant d_1^i < \dots < d_m^i$, $i = 1, 2, \dots, n$, let $w = (x_1)^{a_1}(x_2)^{a_2}\dots(x_n)^{a_n}$ be a signomial term with discrete variables $x_i \in \{d_1^i, \dots, d_m^i\}$ and exponents $a_i \in \mathbb{R}$, $i = 1, 2, \dots, n$. Then $w = v_n$, where v_n is part of a solution to the following linear system:*

$$
\begin{aligned}
& x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \text{for } i = 1, \dots, n, \\
& \sum_{l=1}^{m} u_l^i = 1, \quad \text{for } i = 1, \dots, n, \\
& \sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \text{for } i = 1, \dots, n,\ k = 1, \dots, h, \\
& v_1 = \sum_{l=1}^{m} (d_l^1)^{a_1} u_l^1, \\
& \sum_{l=1}^{m} p_l^i = v_{i-1}, \quad v_i = \sum_{l=1}^{m} (d_l^i)^{a_i} p_l^i, \quad \text{for } i = 2, \dots, n, \\
& M_i(\lambda_k^i - 1) + \sum_{l=1}^{m} p_l^i \leqslant \sum_{l \in G_k} p_l^i \leqslant \lambda_k^i M_i, \quad \text{for } i = 2, \dots, n,\ k = 1, \dots, h, \\
& x_i,\ u_l^i \geqslant 0,\ \lambda_k^i \in \{0, 1\}, \quad \text{for } i = 1, \dots, n,\ l = 1, \dots, m,\ k = 1, \dots, h, \\
& p_l^i \geqslant 0, \quad \text{for } i = 2, \dots, n,\ l = 1, \dots, m,
\end{aligned}
$$

in which $M_i = \prod_{j=1}^{i-1} \max_{l=1,\ldots,m} (d_l^j)^{a_j}$, *for* $i = 2, \ldots, n$, $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ *and the values of* $g_{k,l}$ *are binary numbers satisfying* $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ *for* $l = 1, \ldots, m$ *and* $i = 1, \ldots, n$.

Remark 2. In Proposition 4.2.1, the variables $v_1, v_2, \ldots, v_n$ are explicitly listed in the linear system such that the proposed Super Logarithmic Method may be better understood. In practice, they are not actually required to be variables for linearizing w and we may omit them.

The main result is presented in the next theorem:

Theorem 4.2.2. *Given* $0 \leqslant d_1^i < \ldots < d_m^i$, $i = 1, 2, \ldots, n$, *let* $w = (x_1)^{a_1}(x_2)^{a_2} \ldots (x_n)^{a_n}$ *be a signomial term with discrete variables* $x_i \in \{d_1^i, \ldots, d_m^i\}$ *and exponents* $a_i \in \mathbb{R}$, $i = 1, 2, \ldots, n$. *Then* w *can be linearly represented by* $w = \sum_{l=1}^{m} (d_l^n)^{a_n} p_l^n$, *where* $\{p_l^n\}_{l=1}^m$ *is part of a solution to the following linear system:*

$$x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \text{for } i = 1, \ldots, n, \tag{4.12a}$$

$$\sum_{l=1}^{m} u_l^i = 1, \quad \text{for } i = 1, \ldots, n, \tag{4.12b}$$

$$\sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \text{for } i = 1, \ldots, n,\ k = 1, \ldots, h, \tag{4.12c}$$

$$\sum_{l=1}^{m} p_l^2 = \sum_{l=1}^{m} (d_l^1)^{a_1} u_l^1 \tag{4.12d}$$

$$\sum_{l=1}^{m} p_l^i = \sum_{l=1}^{m_{i-1}} (d_l^{i-1})^{a_{i-1}} p_l^{i-1}, \quad \text{for } i = 3, \ldots, n, \tag{4.12e}$$

$$M_i(\lambda_k^i - 1) + \sum_{l=1}^{m} p_l^i \leqslant \sum_{l \in G_k} p_l^i \leqslant \lambda_k^i M_i, \quad \text{for } i = 2, \ldots, n,\ k = 1, \ldots, h, \tag{4.12f}$$

$$x_i,\ u_l^i \geqslant 0,\ \lambda_k^i \in \{0, 1\}, \quad \text{for } i = 1, \ldots, n,\ l = 1, \ldots, m,\ \ k = 1, \ldots, h, \tag{4.12g}$$

$$p_l^i \geqslant 0, \quad \text{for } i = 2, \ldots, n,\ l = 1, \ldots, m, \tag{4.12h}$$

in which $M_i = \prod_{j=1}^{i-1} \max_{l=1,\ldots,m} (d_l^j)^{a_j}$, *for* $i = 2, \ldots, n$, $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ *and the values of* $g_{k,l}$ *are binary numbers satisfying* $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ *for* $l = 1, \ldots, m$ *and* $i = 1, \ldots, n$.

Proof. Denote sets

$$S_1 = \{(\boldsymbol{x}, w) : w = (x_1)^{a_1}(x_2)^{a_2} \ldots (x_n)^{a_n},\ x_i \in \{d_1^i, \ldots, d_m^i\} \text{ for } i = 1, \ldots, n\}$$

and

$$S_2 = \{(\boldsymbol{x}, w) : \boldsymbol{x} \text{ defined by } (4.12a) - (4.12h), w = \sum_{l=1}^{m} (d_l^n)^{a_n} p_l^n\}.$$

We need to prove that $S_1 = S_2$.

Suppose that $(\boldsymbol{x}, w) \in S_1$, say $x_i = d^i_{l(i)}$ for some $l(i) \in \{1, \ldots, m\}$, $i = 1, \ldots, n$. Additionally, we define that

(i) a nonnegative vector $\boldsymbol{u}^i = (u_1^i, \ldots, u_m^i)$ satisfying $u^i_{l(i)} = 1$ and $u_l^i = 0$ for $l \neq l(i)$ and $i = 1, \ldots, n$;

(ii) a nonnegative vector $\boldsymbol{p}^i = (p_1^i, \ldots, p_m^i)$ such that $p^i_{l(i)} = \prod_{j=1}^{i-1} (d^j_{l(j)})^{a_j}$ and $p_l^i = 0$ for $l \neq l(i)$ and $i = 2, \ldots, n$;

(iii) a 0-1 binary vector $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i)$ with $\lambda_k^i = 1$ if and only if $l(i) \in G_k$, for $i = 1, \ldots, n$.

It can be easily verified that variables $\boldsymbol{x}, \boldsymbol{u}^1, \ldots, \boldsymbol{u}^n, \boldsymbol{p}^2, \ldots, \boldsymbol{p}^n, \boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^n$ satisfy the linear system of (4.12a)-(4.12h) and $w = \sum_{l=1}^{m} (d_l^n)^{a_n} p_l^n$. Thus we have $(\boldsymbol{x}, w) \in S_2$.

Conversely, given $(\boldsymbol{x}, w) \in S_2$, then there exist nonnegative vectors $\boldsymbol{u}^i = (u_1^i, \ldots, u_m^i), i = 1, \ldots, n$; $\boldsymbol{p}^i = (p_1^i, \ldots, p_m^i), i = 2, \ldots, n$ and 0-1 binary vectors $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i), i = 1, \ldots, n$ such that $w = \sum_{l=1}^{m} (d_l^n)^{a_n} p_l^n$ and (4.12a)-(4.12h) hold.

For $i = 1, \ldots, n$, let $K_i = \{k \in \{1, \ldots, h\} : \lambda_k^i = 1\}$ be an index set and define $l(i) = 1 + \sum_{k \in K_i} 2^{k-1}$. Obviously, $l(i)$ is the unique index that ensures $k \in K_i$ if and only if $l(i) \in G_k$. For any $l' \in \{1, \ldots, n\}$ with $l \neq l(i)$, there exists some $k' \notin K_i$ such that $l' \in G_{k'}$. Hence, (4.12c) implies that

$$u^i_{l'} \leqslant \sum_{l \in G_{k'}} u_l^i = \lambda^i_{k'} = 0, \ i = 1, \ldots, n.$$

The non-negativity of $u^i_{l'}$ further leads to

$$u^i_{l'} = 0, \text{ for } l' \neq l(i); \ i = 1, \ldots, n. \tag{4.13}$$

Plugging (4.13) into (4.12b), we have $u^i_{l(i)} = 1$. Then $x_i = d^i_{l(i)}$ is obtained by carrying the values of $\{u_l^i\}_{l=1}^m$ into (4.12a), for $i = 1, \ldots, n$.

Similarly, from (4.12f) we have

$$p^i_{l'} \leqslant \sum_{l \in G_{k'}} p_l^i = \lambda^i_{k'} = 0, \ i = 2, \ldots, n,$$

and the non-negativity of and $p^i_{l'}$ implies

$$p^i_{l'} = 0, \text{ for } l' \neq l(i);\ i = 2, \ldots, n. \tag{4.14}$$

Following to (4.14) and (4.12d), we get $p^2_{l(2)} = \sum_{l=1}^{m} (d^1_l)^{a_1} u^1_l = (d^1_{l(1)})^{a_1}$. Then, for $i = 3, \ldots, n$, the value of $\sum_{l=1}^{m} p^i_l$ is obtained by taking $\boldsymbol{p}^{(i-1)}$ into (4.12e). Hence, we have $p^i_{l(i)} = (d^{i-1}_{l(i-1)})^{a_{i-1}} p^{i-1}_{l(i-1)} = \ldots = \prod_{j=1}^{i-1} (d^j_{l(j)})^{a_j}$. Finally, w is calculated by $w = (d^n_l)^{a_n} p^n_{l(n)} = \prod_{i=1}^{n} (d^i_{l(i)})^{a_i}$. Therefore, $\boldsymbol{x}$ and w satisfy $w = (x_1)^{a_1}(x_2)^{a_2} \ldots (x_n)^{a_n}$, which means that $(\boldsymbol{x}, w) \in S_1$. □

Note that the Super Logarithmic Method is designed for linearizing general terms in the form of $w = (x_1)^{a_1} \ldots (x_n)^{a_n}$ with discrete variables $x_1, \ldots, x_n$ and real exponents $a_1, \ldots, a_n$. Subsequently, while involving additional constraints defined by signomial functions, the problem (DSP) can also be transformed into a 0-1 MILP problem by the Super Logarithmic Method.

In real applications, the values that discrete variables may take are usually equidifferent, i.e., $x_i \in \{b_i + r_i l : l = 1, 2, \ldots, m\}$ with $b_i, r_i \in \mathbb{R}_+$, for $i = 1, \ldots, n$. In this special case, x_i can be linearly expressed by using the binary variables $\boldsymbol{\lambda}^i$ only, without introducing the continuous variables $\boldsymbol{u}^i$ [89].

Lemma 4.2.1. *Given that $x_i \in \{b_i + r_i l : l = 1, 2, \ldots, m\}$ with $b_i, r_i \geqslant 0$, then the discrete variable x_i can be linearly represented as*

$$x_i = b_i + r_i \left(1 + \sum_{k=1}^{h} 2^{k-1} \lambda^i_k\right),$$

in which $h = \lceil \log_2 m \rceil$ and $\lambda^i_k \in \{0, 1\}$, for $k = 1, \ldots, h$.

Notice that the Big-M constraints in the Super Logarithmic Method do not use $\boldsymbol{u}^i$ either. Hence, when the discrete variables take equidifferent values, the proposed Super Logarithmic Method no longer requires those continuous variables $\boldsymbol{u}^i$ to linearize a signomial term. From Lemma 4.2.1 and Theorem 4.2.2, we have the next corollary that shows the linearization of a signomial term with special discrete variables in the Super Logarithmic Method.

Corollary 4.2.3. *Given $b_i, r_i \geqslant 0$, $i = 1, 2, \ldots, n$, let $w = (x_1)^{a_1}(x_2)^{a_2} \ldots (x_n)^{a_n}$ be a signomial term with discrete variables $x_i \in \{b_i + r_i l : l = 1, 2, \ldots, m\}$ and exponents $a_i \in \mathbb{R}$, $i = 1, \ldots, n$. Then w can be linearly represented by $w = \sum_{l=1}^{m} (b_n + r_n l)^{a_n} p^n_l$, where $\{p^n_l\}_{l=1}^{m}$ is part of a solution to the following linear system:*

$$x_i = b_i + r_i \left(1 + \sum_{k=1}^{h} 2^{k-1} \lambda^i_k\right), \quad \text{for } i = 1, \ldots, n,$$

$$
\begin{aligned}
&\sum_{l=1}^{m} p_l^1 = 1,\\
&\sum_{l\in G_k} p_l^1 = \lambda_k^1, \quad \textit{for } k = 1,\dots,h,\\
&\sum_{l=1}^{m} p_l^2 = \sum_{l=1}^{m} (b_1 + r_1 l)^{a_1} p_l^1,\\
&\sum_{l=1}^{m} p_l^i = \sum_{l=1}^{m} (b_{i-1} + r_{i-1} l)^{a_{i-1}} p_l^{i-1}, \quad \textit{for } i = 3,\dots,n,\\
&M_i(\lambda_k^i - 1) + \sum_{l=1}^{m} p_l^i \leqslant \sum_{l\in G_k} p_l^i \leqslant \lambda_k^i M_i, \quad \textit{for } i = 2,\dots,n,\ k = 1,\dots,h,\\
&x_i,\ p_l^i \geqslant 0,\ \lambda_k^i \in \{0,1\}, \quad \textit{for } i = 1,\dots,n,\ l = 1,\dots,m,\ k = 1,\dots,h,
\end{aligned}
$$

in which $M_i = \prod_{j=1}^{i-1} \max_{l=1,\dots,m} (d_l^j)^{a_j}$, *for* $i = 2,\dots,n$, $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ *and the values of* $g_{k,l}$ *are binary numbers satisfying* $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ *for* $l = 1,\dots,m$.

From the above discussions, we see that the proposed Super Logarithmic Method significantly reduces the required number of Big-M constraints for linearizing a signomial term with discrete variables. When the discrete variables take equidifferent values, both of the numbers of continuous variables and equality constraints required by the Super Logarithmic Method can be further reduced.

4.3 Comparison with other methods

In this section, we compare the proposed Super Logarithmic Method with other referred methods for linearizing a single signomial term. The comparison can be easily extended to that for (DSP) problems involving multiple signomial terms.

For linearizing a signomial term $w = (x_1)^{a_1}(x_2)^{a_2}\dots(x_n)^{a_n}$ with $a_1,\dots,a_n \in \mathbb{R}$, Table 4.1 lists the required numbers of binary variables, continuous variables, linear equality constraints and inequality constraints required by Basic Method 1, Basic Method 2, Logarithmic Method and Super Logarithmic Method (referring to Theorem 4.2.2).

From Table 4.1, we have the following observations:

(i) The Basic Method 2 improves Basic Method 1 by significantly reducing the required number of inequality constraints at the cost of using more continuous variables.

(ii) Compared to Basic Method 2, the Logarithmic Method further reduces the required number of binary variables by half an order, while more continuous variables and equality

Table 4.1: Comparison of four methods for linearizing one signomial term

Methods	# of 0-1 variables	# of continuous variables	# of equality constraints	# of inequality constraints
Basic Method 1	nm	$1+n$	$2n$	$2m^n$
Basic Method 2	nm	$2n$	$2n$	$2(n-1)m$
Logarithmic	$n\lceil\log_2 m\rceil$	$2n+nm$	$1+2n+n\lceil\log_2 m\rceil$	$2(n-1)m$
Super Logarithmic	$n\lceil\log_2 m\rceil$	$n+(2n-1)m$	$3n+n\lceil\log_2 m\rceil-1$	$2(n-1)\lceil\log_2 m\rceil$

constraints are included.

(iii) Following the Logarithmic Method, the proposed Super Logarithmic Method again reduces the required number of inequality constraints by half an order, at the expense of requiring more continuous variables and equality constraints.

Among the above four methods, the Super Logarithmic Method requires the fewest binary variables and inequality constraints at the cost of using more continuous variables and equality constraints to linearize a signomial term with discrete variables and real exponents. According to [9], the performance of MILP solvers is dominated by the numbers of binary variables and inequality constraints, and less affected by the numbers of continuous variables and equality constraints. Therefore, it is reasonable to expect that the proposed Super Logarithmic Method could run much faster than other referred methods for solving (DSP) problems.

4.4 Extensions of the Super Logarithmic Method

The comparison results in Section 4.3 showed that, for linearizing signomial terms with discrete variables, fewer binary variables and Big-M inequality constraints are required by the proposed Super Logarithmic Method than other known linearization methods. Solving the 0-1 MILP reformulation of problem (DSP) by the Super Logarithmic Method is expected to be more efficient. In this section, we show that the Super Logarithmic Method is applicable linearize more complicated terms and provide a class of "representable programming problems" using the Super Logarithmic Method.

4.4.1 General product term

Recall that Theorem 4.2.2 generates a linear system of continuous variables $v_1, \ldots, v_n$ such that v_n represents a signomial term $w = (x_1)^{a_1} \ldots (x_n)^{a_n}$ with discrete variables $x_i \in \{d_1^i, \ldots, d_m^i\}$, by using the proposed Super Logarithmic Method. Defining a function $f_i^*(x_i) = (x_i)^{a_i}$ for $i = 1, \ldots, n$, we observe that the linear system with $(d_l^i)^{a_i}$ being replaced by $f_i^*(d_l^i), i = 1, \ldots, n$, still

guarantees that $v_n = w$. If the function f_i^* is further extended to a general nonnegative valued function $f_i : \mathbb{R}_+ \mapsto \mathbb{R}_+$, $i = 1, \ldots, n$, we claim that the modified system can linearly express a corresponding product term with discrete variables in the form of $w' = f_1(x_1)f_2(x_2)\ldots f_n(x_n)$.

Note that the key of linearizing w' shall be using continuous variables $p_1^i, \ldots, p_m^i$ to linearly express $f_1(x_1)\ldots f_i(x_i)$, for $i = 2, \ldots, n$. This mechanism motivates us to consider a more general term $w'' = f_1(x_1)\ldots f_n(x_n)y$, where y is a bounded nonnegative continuous variable. To linearize w'', we may require additional continuous variables $p_1^1, \ldots, p_m^1$ to represent $f_1(x_1)y$, and then, the linearization of $f_1(x_1)\ldots f_i(x_i)y$ follows in sequence for $i = 2, \ldots, n$. We present the system that linearizes w'' by using the Super Logarithmic Method in the next theorem.

Theorem 4.4.1. *Let $f_i(\cdot) : \mathbb{R}_+ \mapsto \mathbb{R}_+$ be any nonnegative valued function and $0 \leqslant d_1^i < \ldots < d_m^i$, for $i = 1, \ldots, n$, and $0 \leqslant \underline{y} \leqslant \bar{y}$. If*

$$w'' = f_1(x_1)f_2(x_2)\ldots f_n(x_n)y \tag{4.15}$$

is a product term of $f_i(x_i)$ and y with $x_i \in \{d_1^i, \ldots, d_m^i\}$, $i = 1, \ldots, n$ and $y \in [\underline{y}, \bar{y}]$, then w'' can be linearly represented by $w'' = \sum_{l=1}^{m} f_n(d_l^n)p_l^n$, where $\{p_l^n\}_{l=1}^m$ is part of a solution to the following linear system:

$$x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \text{for } i = 1, \ldots, n, \tag{4.16a}$$

$$\sum_{l=1}^{m} u_l^i = 1, \quad \text{for } i = 1, \ldots, n, \tag{4.16b}$$

$$\sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \text{for } i = 1, \ldots, n,\ k = 1, \ldots, h, \tag{4.16c}$$

$$y = \sum_{l=1}^{m} p_l^1, \quad \sum_{l=1}^{m} f_1(d_l^1)p_l^1 = \sum_{l=1}^{m} p_l^2, \tag{4.16d}$$

$$\sum_{l=1}^{m} p_l^i = \sum_{l=1}^{m} f_{i-1}(d_l^{i-1})p_l^{i-1}, \quad \text{for } i = 3, \ldots, n, \tag{4.16e}$$

$$\bar{y}(\lambda_k^1 - 1) + y \leqslant \sum_{l \in G_k} p_l^1 \leqslant \lambda_k^1 \bar{y}, \quad \text{for } k = 1, \ldots, h, \tag{4.16f}$$

$$M_i(\lambda_k^i - 1) + \sum_{l=1}^{m} p_l^{i-1} \leqslant \sum_{l \in G_k} p_l^i \leqslant \lambda_k^i M_i, \quad \text{for } i = 2, \ldots, n,\ k = 1, \ldots, h, \tag{4.16g}$$

$$\underline{y} \leqslant y \leqslant \bar{y},\ x_i,\ u_l^i,\ p_l^i \geqslant 0,\ \lambda_k^i \in \{0, 1\}, \quad \text{for } i = 1, \ldots, n,\ l = 1, \ldots, m,\ k = 1, \ldots, h, \tag{4.16h}$$

in which $M_i = \prod_{j=1}^{i-1} \max_{l=1,\ldots,m} f_j(d_l^j)$, *for* $i = 2, \ldots, n$, $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ *and the values of* $g_{k,l}$ *are binary numbers satisfying* $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ *for* $l = 1, \ldots, m$ *and* $i = 1, \ldots, n$.

Proof. Denote sets

$$S_1 = \{(\boldsymbol{x}, y, w'') : w'' \text{ defined by (4.15)}, \ y \in [\underline{y}, \bar{y}], \ x_i \in \{d_1^i, \ldots, d_m^i\} \text{ for } i = 1, \ldots, n\}$$

and

$$S_2 = \{(\boldsymbol{x}, y, w'') : \boldsymbol{x}, y \text{ defined by (4.16}a\text{)} - \text{(4.16}h\text{)}, \ w'' = \sum_{l=1}^{m} f_n(d_l^n) p_l^n\}.$$

We need to prove that $S_1 = S_2$.

Suppose that $(\boldsymbol{x}, y, w'') \in S_1$, say $x_i = d_{l(i)}^i$ for some $l(i) \in \{1, \ldots, m\}$, $i = 1, \ldots, n$. Additionally, for $i = 1, \ldots, n$, we define that

(i) a nonnegative vector $\boldsymbol{u}^i = (u_1^i, \ldots, u_m^i)$ satisfying $u_{l(i)}^i = 1$ and $u_l^i = 0$ for $l \neq l(i)$;

(ii) a nonnegative vector $\boldsymbol{p}^i = (p_1^i, \ldots, p_m^i)$ such that $p_{l(1)}^1 = y$ and $p_l^1 = 0$ for $l \neq l(1)$; $p_{l(i)}^i = \prod_{j=1}^{i-1} f_j(x_j) y$ and $p_l^i = 0$ for $l \neq l(i), i = 2, \ldots, n$;

(iii) a 0-1 binary vector $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i)$ with $\lambda_k^i = 1$ if and only if $l(i) \in G_k$.

It can be easily verified that variables $\boldsymbol{x}, y, \boldsymbol{u}^i, \boldsymbol{p}^i, \boldsymbol{\lambda}^i, i = 1, \ldots, n$ satisfy the linear system of (4.16a)-(4.16h) and $w'' = \sum_{l=1}^{m} f_n(d_l^n) p_l^n$. Thus we have $(\boldsymbol{x}, y, w'') \in S_2$.

Conversely, given $(\boldsymbol{x}, y, w'') \in S_2$, then there exist nonnegative vectors $\boldsymbol{u}^i = (u_1^i, \ldots, u_m^i)$, $\boldsymbol{p}^i = (p_1^i, \ldots, p_m^i)$ and 0-1 binary vectors $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i)$, $i = 1, \ldots, n$ such that $w'' = \sum_{l=1}^{m} f_n(d_l^n) p_l^n$ and (4.16a)-(4.16h) hold.

For $i = 1, \ldots, n$, let $K_i = \{k \in \{1, \ldots, h\} : \lambda_k^i = 1\}$ be an index set and define $l(i) = 1 + \sum_{k \in K_i} 2^{k-1}$. Obviously, $l(i)$ is the unique index that ensures $k \in K_i$ if and only if $l(i) \in G_k$. For any $l' \in \{1, \ldots, n\}$ with $l \neq l(i)$, there exists some $k' \notin K_i$ such that $l' \in G_{k'}$. Hence, (4.16c) implies that

$$u_{l'}^i \leqslant \sum_{l \in G_{k'}} u_l^i = \lambda_{k'}^i = 0.$$

From (4.16f) and (4.16g), we have

$$p_{l'}^i \leqslant \sum_{l \in G_{k'}} p_l^i = \lambda_{k'}^i = 0.$$

The non-negativity of $u^i_{l'}$ and $p^i_{l'}$ further leads to

$$u^i_{l'} = p^i_{l'} = 0, \text{ for } l' \neq l(i). \tag{4.17}$$

Plugging (4.17) into (4.16b), we have $u^i_{l(i)} = 1$. Then $x_i = d^i_{l(i)}$ is obtained by carrying the values of $\{u^i_l\}_{l=1}^m$ into (4.16a), for $i = 1, \ldots, n$.

From (4.16d) and (4.17), we see that $p^1_{l(1)} = y$. Following to (4.17) and (4.16e), the value of $\sum_{l=1}^{m} p^i_l$ is obtained by taking $p^{(i-1)}_{l(i-1)}$ into (4.16e). Hence, we have $p^i_{l(i)} = \prod_{j=1}^{i-1} f_j(d^j_{l(j)})y$ for $i = 2, \ldots, n$. At last, $w'' = \sum_{l=1}^{m} f_n(d^n_l)p^n_l$ is calculated by $w'' = \prod_{i=1}^{n} f_i(d^i_{l(i)})y = \prod_{i=1}^{n} f_i(x_i)y$. Consequently, $\boldsymbol{x}, y$ and w'' satisfy (4.15), which means that $(\boldsymbol{x}, y, w'') \in S_1$. □

4.4.2 Fractional term

In this subsection, we show that the Super Logarithmic Method can be extended to treat the following fractional term with a discrete valued denominator:

$$w''' = G(\boldsymbol{x}, \boldsymbol{y})/F(\boldsymbol{x}),$$

where $\boldsymbol{y}$ is a bounded nonnegative continuous vector, function $G(\boldsymbol{x}, \boldsymbol{y})$ is a linear combination of terms in the form of $w'' = f_1(x_1)f_2(x_2)\ldots f_n(x_n)y$ and function $F(\boldsymbol{x})$ is a nonzero linear combination of terms in the form of $w' = f_1(x_1)f_2(x_2)\ldots f_n(x_n)$.

Since w''' satisfies the equation of $G(\boldsymbol{x}, \boldsymbol{y}) = F(\boldsymbol{x})w'''$, it suffices to simultaneously linearize $G(\boldsymbol{x}, \boldsymbol{y})$ and $F(\boldsymbol{x})w'''$. Notice that both of $G(\boldsymbol{x}, \boldsymbol{y})$ and $F(\boldsymbol{x})w'''$ are linear combinations of product terms in discrete variables and a continuous variable, which are representable by the Super Logarithmic Method and the Logarithmic Method. Hence, the fractional term w''' can be linearly expressed by both methods. From previous comparisons, it is expected that the Super Logarithmic Method is more effective for linearizing w''' than the Logarithmic Method. For illustration purposes, we present the following example:

Example 4.4.1. Consider the following fractional term:

$$\bar{w} = \frac{g_1(x_1)y}{f_1(x_1) + f_2(x_2) + f_3(x_3)}, \tag{4.18}$$

where $x_i \in \{d^i_1, \ldots, d^i_m\}$ with $0 \leqslant d^i_1 < \ldots < d^i_m, i = 1, 2, 3$, $y \in [\underline{y}, \bar{y}]$ with $0 \leqslant \underline{y} \leqslant \bar{y}$ and $g_1, f_1, f_2, f_3 : \mathbb{R}_+ \mapsto \mathbb{R}_+$ are nonnegative valued functions.

The idea is to consider $g_1(x_1)y = f_1(x_1)\bar{w} + f_2(x_2)\bar{w} + f_3(x_3)\bar{w}$ by regarding $\bar{w}$ as a continuous variable. In this way, we can linearize $g_1(x_1)y$, $f_1(x_1)\bar{w}$, $f_2(x_2)\bar{w}$ and $f_3(x_3)\bar{w}$ simultaneously

using the Super Logarithmic Method. Consequently, the system that linearizes $\bar{w}$ becomes

$$
\begin{aligned}
&(4.16a)-(4.16c),\\
&\bar{w}=\sum_{l=1}^{m} p_l^i, \quad \text{for } i=1,2,3,\\
&y=\sum_{l=1}^{m} q_l^1,\\
&\sum_{l=1}^{m} g_1(d_l^1)q_l^1=\sum_{i=1}^{3}\sum_{l=1}^{m} f_i(d_l^i)p_l^i,\\
&M(\lambda_k^i-1)+y\leqslant \sum_{l\in G_k} p_l^i\leqslant \lambda_k^i M, \quad \text{for } k=1,\ldots,h,\ i=1,2,3,\\
&\bar{y}(\lambda_k^1-1)+y\leqslant \sum_{l\in G_k} q_l^1\leqslant \lambda_k^1\bar{y}, \quad \text{for } k=1,\ldots,h,\\
&\bar{w}\in[0,M],\ y\in[\underline{y},\bar{y}],\ q_l^1,\ p_l^i\geqslant 0,\ \lambda_k^i\in\{0,1\}, \quad \text{for } l=1,\ldots,m,\ k=1,\ldots,h,\ i=1,2,3,
\end{aligned}
$$

in which $h=\lceil \log_2 m\rceil$ and M is a calculable upper bound of $\bar{w}$.

4.4.3 Representable programming problems

Our Super Logarithmic Method was initially proposed for linearizing signomial terms with discrete variables that are involved in the problem (DSP). In Sections 4.4.1 and 4.4.2, we extend the proposed method to treat more complicated terms in the forms of w', w'' and w''', respectively. Therefore, the proposed Super Logarithmic Method becomes applicable to linearize more complicated programming problems, which we gather into a class of "representable programming problems" in this subsection.

Given $0\leqslant d_1^i<\ldots<d_m^i$, $i=1,\ldots,n$ and $0\leqslant \underline{y}_j\leqslant \bar{y}_j$, $j=1,\ldots,J$, let discrete variables $x_i\in\{d_1^i,\ldots,d_m^i\}$ and continuous variables $y_j\in[\underline{y}_j,\bar{y}_j]$. The first type of representable functions using the Super Logarithmic Method can be expressed as

$$F(\mathbf{x})=\sum_{p\in P} c_p f_{p,1}(x_1)f_{p,2}(x_2)\ldots f_{p,n}(x_n),$$

where P is a finite index set, $c_p\in\mathbb{R}$ and $f_{p,i}(\cdot):\mathbb{R}_+\mapsto\mathbb{R}_+$, for $p\in P$ and $i=1,\ldots,n$. Complicating with continuous variables, the first type of representable functions can be extended to

$$G(\boldsymbol{x},\boldsymbol{y})=\sum_{q\in Q} b_q g_{q,1}(x_1)g_{q,2}(x_2)\ldots g_{q,n}(x_n)(y_{\alpha_q})^{\beta_q}, \tag{4.19}$$

where Q is a finite index set, $\alpha_q \in \{1, \ldots, J\}, \beta_q \in \{0, 1\}$, $b_q \in \mathbb{R}$ and $g_{q,i}(\cdot) : \mathbb{R}_+ \mapsto \mathbb{R}_+$, for $q \in Q$ and $i = 1, \ldots, n$. Combining them, we may further consider the fractional term of

$$\bar{F}(\mathbf{x}, \mathbf{y}) = \frac{G(\mathbf{x}, \mathbf{y})}{F(\mathbf{x})}, \tag{4.20}$$

where function $F(\mathbf{x})$ does not achieve zero value.

Therefore, the Super Logarithmic Method is applicable to linearize a programming problem whose objective and constraint functions take the form of $\bar{F}(\boldsymbol{x}, \boldsymbol{y})$ defined by (4.20). Notice that any constraint $\bar{F}(\mathbf{x}, \mathbf{y}) \leqslant b$ with constant $b \in \mathbb{R}$ can be transformed into $G(\mathbf{x}, \mathbf{y}) - bF(\mathbf{x}) \leqslant 0$, i.e., $G_1(\boldsymbol{x}, \boldsymbol{y}) \leqslant 0$ for some function G_1 in the form of (4.19). Consequently, we have the following class of "representable programming problems" to which the Super Logarithmic Method applies:

$$\begin{array}{lll} & \min & \bar{F}(\mathbf{x}, \mathbf{y}) \\ \text{(P)} & s.t. & G_t(\mathbf{x}, \mathbf{y}) \leqslant 0, \quad t \in T, \\ & & G_s(\mathbf{x}, \mathbf{y}) = 0, \quad s \in S, \\ & & x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, \\ & & y_j \in [\underline{y}_j, \bar{y}_j], \quad j = 1, \ldots, J, \end{array}$$

where T and S are finite index sets, $0 \leqslant d_1^i < \ldots < d_m^i$ for $i = 1, \ldots, n$, $0 \leqslant \underline{y}_j \leqslant \bar{y}_j$ for $j = 1, \ldots, J$, $\bar{F}$ is a function in the form of (4.20), and G_t, G_s are functions in the form of (4.19) for $t \in T$ and $s \in S$.

4.5 Numerical experiments

Various practical engineering design optimization problems are tested in this section. The experiments are run on a PC equipped with the Intel Core2 Duo CPU, 4GB RAM and Windows 7 (32 bit) operating system. These programs are solved by [35] using the MIP Solver for solving integer LP-formulations of the Logarithmic Method and the Super Logarithmic Method.

4.5.1 Compression spring design optimization problem

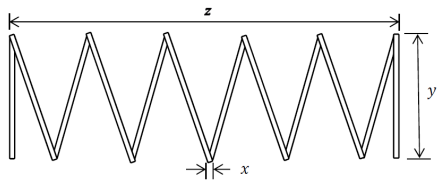
A spring design optimization problem in industrial engineering minimizes the weight of a tension/compression spring (shown in Figure 4.1) subjected to feasible constraints of minimum deflection, shear stress, surge frequency and limits on outside diameter. There are three design variables including: (i) the wire diameter x, (ii) the mean coil diameter y , and (iii) the number of active coils z. By referring to [5, 87], the problem can be modified as a signomial programming

problem with positive discrete variables as follows:

$$
\begin{aligned}
\min \quad & f(x,y,z) = x^2y(z+2) \\
\text{s.t.} \quad & 7178x^4 - y^3z \leqslant 0, \\
& 4x^2y^2 + \frac{1}{5108}(7458x^3y - x^4) - 12566x^5y + x^6 \leqslant 0, \\
& y^2z - 140.45x \leqslant 0, \\
& y + x - 1.5 \leqslant 0, \\
& x \in \left\{ 0.05 + \frac{2-0.05}{m-1} l : l = 0, 1, \ldots, m-1 \right\}, \\
& y \in \left\{ 0.25 + \frac{1.3-0.05}{m-1} l : l = 0, 1, \ldots, m-1 \right\}, \\
& z \in \{2, 2.5, 3.1, 4, 5.2, 6.3, 7.1, 8.5, 9, 10, 11.7, 12.1, 12.3, 13.7, 14.3, 15\},
\end{aligned}
$$

where m means the number of discrete points of x between 0.05 and 2, and the number of discrete points of y between 0.25 and 1.3.

Figure 4.1: Compression spring design optimization problem



The above problem is indeed an example of (DSP). Table 4.2 lists the problem size, $300 \leqslant m \leqslant 700$, the number of 0-1 variables, the number of continuous variables, the number of equality constraints, the number of inequality constraints, the number of iterations, the CPU time, the solution and the objective value from GUROBI (2014) for the Logarithmic Method (Log) and the Super Logarithmic Method (Super Log).

It illustrates that the proposed Super Logarithmic Method needs to use more continuous variables. However, it is computationally more efficient than the current Logarithmic Method for large m. Taking $m = 700$ as an example, solving the Logarithmic Method takes 96.7 CPU seconds while solving the Super Logarithmic Method takes 0.4 CPU seconds only. The advantage of working on the Super Logarithmic Method is apparent.

Table 4.2: Experimental results of compression spring design optimization problem

m	Methods	# of 0-1 variables	# of continuous variables	# of equalities	# of inequalities	Iterations	CPU Time (seconds)	Solutions (x, y, x)	Objective value
300	Log	22	628	28	5,404	52,049	12.7	(0.05, 0.28282,	0.002828
	Super Log	22	1,826	28	76	464	0.1	2)	
400	Log	22	828	28	7,204	53,751	13.1	(0.05, 0.2826,	0.002826
	Super Log	22	2,426	28	76	605	0.1	2)	
500	Log	22	1,028	28	9,004	57,300	20.1	(0.05, 0.28247,	0.002825
	Super Log	22	3,026	28	76	667	0.2	2)	
600	Log	24	1,228	30	10,804	63,075	21.3	(0.05, 0.28239,	0.002825
	Super Log	24	3,626	29	84	848	0.2	2.0101)	
700	Log	24	1,428	30	12,604	267,756	96.7	(0.05, 0.28233,	0.002822
	Super Log	24	4,226	29	84	618	0.4	2)	

4.5.2 Pressure vessel optimization problem

For designing a pressure vessel depicted in Figure 4.2, an engineer needs to decide the values of the spherical heal thickness x_1, the shell thickness x_2, the radius x_3, and the length of the shell x_4 thus to minimize the cost of manufacturing this pressure vessel subjected to the constraints such as tolerable pressure and feasible sizes. For the cases that x_1, x_2, x_3 and x_4 are continuous variables, this problem has been discussed by [69, 87]. For the purpose of demonstration, here we modify this problem as a mixed discrete problem where x_1, x_2, x_3 are discrete variables and x_4 is a continuous variable. The modified vessel design problem is expressed as

$$
\begin{aligned}
\min \quad & 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\
\text{s.t.} \quad & -x_1 + 0.0193x_3 \leqslant 0, \\
& -\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1,296,000 \leqslant 0, \\
& -x_2 + 0.00954x_3 \leqslant 0, \\
& x_1, x_2 \in \left\{ \frac{2.919}{m-1}(l-1) : l = 1, \ldots, m \right\}, \\
& x_3 \in \left\{ \frac{500}{m-1}(l-1) : l = 1, \ldots, m \right\}, \\
& 10 \leqslant x_4 \leqslant 240,
\end{aligned}
$$

where m means the number of discrete points of x_1, x_2 and x_3.

This problem contains signomial terms with discrete variables and a continuous variable, and thus is an example of (P). Instances with various m values, $200 \leqslant m \leqslant 400$, are tested for the Logarithmic Method and the Super Logarithmic Method. The computational results are reported in Table 4.3. The results illustrate that the Super Logarithmic Method uses much fewer inequality constraints and its computational time is over 10 times shorter than the Logarithmic Method.

Figure 4.2: Tube and end section of pressure vessel

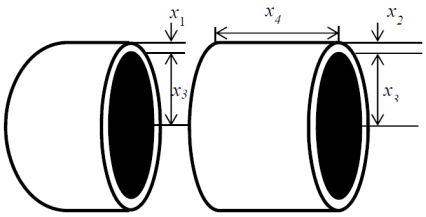


Table 4.3: Experimental results of compression spring design optimization problem

m	Methods	# of 0-1 variables	# of continuous variables	# of equalities	# of inequalities	Iterations	CPU Time (seconds)	Solutions $(x_1, \ldots, x_4)$	Objective value
200	Log	24	1,009	16	1,004	9,228	6.9	(0.777, 0.396, 40.201, 201.65787)	5,925.454
	Super Log	24	1,209	22	83	3,868	0.4		
300	Log	27	1,509	16	1,504	19,228	7.9	(0.752, 0.371, 38.462, 227.58066)	5,911.765
	Super Log	27	2,809	23	93	14,313	0.7		
400	Log	27	2009	16	2,004	97,107	26.2	(0.05, 0.28247, 38.847, 221.56726)	5,877.141
	Super Log	27	2409	23	93	15,796	0.9		

4.5.3 Speed reducer design problem

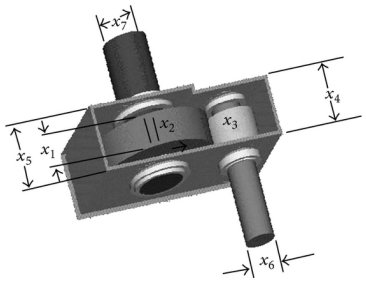
The design of the speed reducer in Figure 4.3 is considered with the face width x_1, module of teeth x_2, number of teeth on pinion x_3, length of the first shaft between bearings x_4, length of the second shaft between bearings x_5, diameter of the first shaft x_6, and diameter of the second shaft x_7. The weight of the speed reducer is to be minimized subject to feasible constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts and stresses in the shaft. The problem is formulated as below:

$$
\begin{aligned}
\min \quad & 0.7854x_1x_2^2(3.3333x_3^2 + 14.9334x_3 - 43.0934) - 1.508x_1(x_6^2 + x_7^2) \\
& + 7.47777(x_6^3 + x_7^3) + 0.7854(x_4x_6^2 + x_5x_7^2) \\
\text{s.t.} \quad & \frac{27}{x_1x_2^2x_3} \geqslant 1, \quad \frac{397.5}{x_1x_2^2x_3^2} \geqslant 1, \\
& \frac{1.93x_4^3}{x_2x_3x_6^4} \geqslant 1, \quad \frac{1.93x_5^3}{x_2x_3x_7^4} \geqslant 1, \\
& \frac{1}{110x_6^3}\sqrt{\left(\frac{745x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6} \leqslant 1, \\
& \frac{1}{85x_7^3}\sqrt{\left(\frac{745x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6} \leqslant 1, \\
& \frac{x_2x_3}{40} \leqslant 1, \quad \frac{5x_2}{x_1} \leqslant 1, \quad \frac{x_1}{12x_2} \leqslant 1, \\
& \frac{1.5x_6 + 1.9}{x_4} \leqslant 1, \quad \frac{1.1x_7 + 1.9}{x_5} \leqslant 1,
\end{aligned}
$$

$$2.6 \leqslant x_1 \leqslant 3.6, \quad 0.7 \leqslant x_2 \leqslant 0.8, \quad 17 \leqslant x_3 \leqslant 28,$$
$$7.3 \leqslant x_4, x_5 \leqslant 8.3, \quad 2.9 \leqslant x_6 \leqslant 3.9, \quad 5 \leqslant x_7 \leqslant 5.5,$$

where each variable x_i takes m discrete equidifferent values between the respective lower and upper bounds, for $i = 1, \ldots, 7$.

Figure 4.3: Speed reducer



Consisting of fractional terms with discrete variables, the above problem is an example of (P) as well. Instances with various values of $100 \leqslant m \leqslant 1000$ are tested. We report the computational results in Table 4.4. The results also clearly indicate that solving the Super Logarithmic Method is much more computationally efficient than solving the Logarithmic Method, especially when m become large. Taking $m = 100$ as an example, solving the Logarithmic Method takes 103.6 CPU seconds while solving the Super Logarithmic Method takes 0.5 CPU seconds only. For $m \geqslant 500$, solving the Logarithmic Method simply failed to produce a solution in over 10 hours of CPU time, while the Super Logarithmic Method can still find the exact solution within a short time.

Table 4.4: Experimental results of compression spring design optimization problem

m	Methods	# of 0-1 variables	# of continuous variables	# of equalities	# of inequalities	Iterations	CPU Time (seconds)	value
100	Log	42	621	6	2,811	136,4060	103.6	2,996.551
	Super Log	42	1,821	77	35	1,985	0.5	
130	Log	48	801	6	3,651	26,001,169	2,471.4	2,996.473
	Super Log	48	2,361	83	35	2,379	1	
150	Log	48	921	6	4,211	49,113,425	5,937.9	2,997.439
	Super Log	48	2,721	83	35	3,215	1.5	
500	Log	54	3,021	6	14,011	-	-	2,995.266
	Super Log	54	9,021	89	35	4,158	3.5	
1000	Log	60	6,021	6	28,011	-	-	2,994.641
	Super Log	60	18,021	95	35	5,604	9.7	

4.6 Linearization of DQP

In previous sections, we have proposed a Super Logarithmic Method for linearizing signomial programming problems with discrete variables (DSP). Its superior performance on the computational efficiency of solving the resulting reformulation has been supported by numerical results. In this section, we intend to customize the Super Logarithmic Method to linearly represent DQP in an efficient manner.

4.6.1 A single discrete-valued quadratic term

Note that DQP can be viewed as a DSP problem whose objective function consists of quadratic terms only, hence, the Super Logarithmic Method can be naturally extended to treating DQP. As a direct result of Theorem 4.2.2, the linearization of a single quadratic term using the Super Logarithmic Method is presented in the next theorem.

Theorem 4.6.1. *For discrete variables $x_i \in \{d_1^i, \ldots, d_m^i\}$ with $0 \leqslant d_1^i < \ldots < d_m^i$, $i = 1, \ldots, n$, we have $x_i x_j = w_{ij}$, where w_{ij} is a solution to the following linear system:*

$$x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad x_j = \sum_{l=1}^{m} d_l^j u_l^j, \quad \sum_{l=1}^{m} u_l^i = 1, \quad \sum_{l=1}^{m} u_l^j = 1, \tag{4.21a}$$

$$\sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \sum_{l \in G_k} u_l^j = \lambda_k^j, \ \textit{for } k = 1, \ldots, \lceil \log_2 m \rceil, \tag{4.21b}$$

$$x_i = \sum_{l=1}^{m} p_l^{ij}, \quad \sum_{l=1}^{m} d_l^j p_l^{ij} = w_{ij}, \tag{4.21c}$$

$$d_m^i(\lambda_k^j - 1) + x_i \leqslant \sum_{l \in G_k} p_l^{ij} \leqslant d_m^i \lambda_k^j, \ \textit{for } k = 1, \ldots, \lceil \log_2 m \rceil, \tag{4.21d}$$

$$x_i,\ x_j,\ w_{ij},\ u_l^i,\ u_l^j,\ p_l^{ij} \geqslant 0,\ \lambda_k^i,\ \lambda_k^j \in \{0,1\},\ \textit{for } l = 1, \ldots, m,\ k = 1, \ldots, h, \tag{4.21e}$$

in which $i, j \in \{1, \ldots, n\}$ with $i \neq j$, $G_k = \{l : g_{k,l} = 1\}$ and the values of $g_{k,l}$ are binary numbers satisfying $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ for $l = 1, \ldots, m,\ k = 1, \ldots, \lceil \log_2 m \rceil$.

Based on the logarithmic representation (4.21a)-(4.21b) and (4.21e) of discrete variables, there are other state-of-the-art linearization methods that guarantee the continuous variable w_{ij} to be a valid representation of $x_i x_j$. As introduced in Section 1, one is using the following Big-M inequality constraints provided by the Logarithmic Method:

$$(d_m^i d_m^j)(u_l^j - 1) + d_l^j x_i \leq w_{ij} \leq (d_m^i d_m^j)(1 - u_l^j) + d_l^j x_i, \ \text{for } l = 1, \ldots, m.$$

Besides, the Conventional Method of linearizing quadratic terms in 0-1 binary variables [22]

provides the following linear system in which w_{ij} is a valid representation of $x_i x_j$:

$$v_{ls}^{ij} \leq u_l^i, \quad v_{ls}^{ij} \leq u_s^j, \quad v_{ls}^{ij} \geq u_l^i + u_s^j - 1, \text{ for } l, s = 1, \ldots, m.$$
$$w_{ij} = \sum_{l=1}^{m} \sum_{s=1}^{m} d_l^i d_s^j v_{ls}^{ij}, \quad v_{ls}^{ij} \geq 0, \text{ for } l, s = 1, \ldots, m.$$

The comparison of above three methods for linearizing a quadratic term with discrete variables in m possible discrete values is given in the next table.

Table 4.5: Comparison of three methods for linearizing one quadratic term

Methods	# of 0-1 variables	# of continuous variables	# of equality constraints	# of inequality constraints
Conventional	$2\lceil \log_2 m \rceil$	$3 + 2m + m^2$	$5 + 2\lceil \log_2 m \rceil$	$3m^2$
Logarithmic	$2\lceil \log_2 m \rceil$	$3 + 2m$	$4 + 2\lceil \log_2 m \rceil$	$2m$
Super Logarithmic	$2\lceil \log_2 m \rceil$	$3 + 3m$	$6 + 2\lceil \log_2 m \rceil$	$2\lceil \log_2 m \rceil$

From Table 4.5, we see that the Super Logarithmic Method outperforms the Logarithmic Method for linearizing a discrete-valued quadratic term due to the half-order reduction of Big-M inequality constraints. Adopting the conventional approach of treating 0-1 valued quadratic term, the Conventional Method requires $O(m^2)$ continuous variables and inequalities constraints, which is one-order more than those by the Logarithmic Method and the Super Logarithmic Method. As m becomes large, the large number of continuous variables and inequalities in the Conventional Method may significantly aggravate the computational burden.

4.6.2 DQP with multiple discrete-valued quadratic terms

When it comes to reformulating DQP, we need to linearize multiple discrete-valued quadratic terms. The linearization of each quadratic term using the Super Logarithmic Method follows the linear system of (4.21a)-(4.21e) in Theorem 4.6.1. The corresponding 0-1 mixed-integer linear reformulation of DQP is then given in the next theorem.

Theorem 4.6.2. *Given $\boldsymbol{A} \in \mathcal{S}^n$, $\boldsymbol{c} \in \mathbb{R}^n$ and $0 \leqslant d_1^i < \ldots < d_m^i$, $i = 1, \ldots, n$, the following DQP problem:*

$$\begin{aligned} \min \quad & \tfrac{1}{2}\boldsymbol{x}^T \boldsymbol{A}\boldsymbol{x} + \boldsymbol{c}^T \boldsymbol{x} \\ s.t. \quad & x_i \in \{d_1^i, \ldots, d_m^i\}, \quad i = 1, \ldots, n, \end{aligned} \tag{4.22}$$

has an equivalent 0-1 mixed-integer linear programming reformulation as below.

$$\min \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{l=1}^{m} A_{ii}(d_l^i)^2 u_l^i + \sum_{i=1}^{n-1}\sum_{j>i} A_{ij}w_{ij} + \boldsymbol{c}^T\boldsymbol{x} \tag{4.23a}$$

$$s.t. \quad x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \sum_{l=1}^{m} u_l^i = 1, \ \textit{for } i = 1,\dots,n, \tag{4.23b}$$

$$\sum_{l\in G_k} u_l^i = \lambda_k^i, \quad \textit{for } i = 1,\dots,n, \ k = 1,\dots,h, \tag{4.23c}$$

$$x_i = \sum_{l=1}^{m} p_l^{ij}, \quad \sum_{l=1}^{m} d_l^j p_l^{ij} = w_{ij}, \ \textit{for } i,j = 1,\dots,n, \ i<j, \ k = 1,\dots,h, \tag{4.23d}$$

$$d_m^i(\lambda_k^j - 1) + x_i \leqslant \sum_{l\in G_k} p_l^{ij} \leqslant d_m^i\lambda_k^j, \ \textit{for } i,j = 1,\dots,n, \ i<j, \ k = 1,\dots,h, \tag{4.23e}$$

$$\boldsymbol{x} \in \mathbb{R}_+^n, \ \boldsymbol{u}^i \in \mathbb{R}_+^m, \ \boldsymbol{\lambda}^i \in \{0,1\}^h, \ \textit{for } i = 1,\dots,n, \tag{4.23f}$$

$$w_{ij} \geq 0, \ \boldsymbol{p}^{ij} \in \mathbb{R}_+^m, \ \textit{for } i,j = 1,\dots,n, \ i<j, \tag{4.23g}$$

where $h = \lceil \log_2 m \rceil$, $G_k = \{l : g_{k,l} = 1\}$ *and the values of* $g_{k,l}$ *are binary numbers satisfying* $l = 1 + \sum_{k=1}^{h} 2^{k-1} g_{k,l}$ *for* $l = 1,\dots,m, \ k = 1,\dots,h$.

The comparison of the Super Logarithmic Method, the Logarithmic Method and the Conventional Method for reformulating a DQP problem follows the results in Table 4.5. Next we shall further exploit the structure of reformulation in this simple quadratic case to improve the effectiveness of the Super Logarithmic Method for linearizing DQP.

4.6.3 Enhancement

From Theorem 4.6.2, the linearization of each quadratic term x_ix_j with $i<j$ requires a continuous vector $\boldsymbol{p}^{ij} \in \mathbb{R}_+^m$ and corresponding $2h$ Big-M inequality constraints in terms of $\boldsymbol{\lambda}^i$. We find that the Big-M constraints for different terms $x_1x_j,\dots,x_{j-1}x_j$ that share the same variable x_j can be further aggregated, for $j = 2,\dots,n$.

Theorem 4.6.3. *The DQP problem (4.22) is equivalent to a modified 0-1 mixed-integer linear programming problem (4.23a)-(4.23g) with (4.23e) replaced by the following constraints:*

$$\sum_{i<j} d_m^i(\lambda_k^j - 1) + \sum_{i<j} x_i \leqslant \sum_{i<j}\sum_{l\in G_k} p_l^{ij} \leqslant \sum_{i<j} d_m^i\lambda_k^j, \ \textit{for } j = 2,\dots,n, \ k = 1,\dots,h, \tag{4.24}$$

Proof. Denote sets

$$S_1 = \{(\boldsymbol{x},\boldsymbol{w}) : w_{ij} = x_ix_j, \ x_i \in \{d_1^i,\dots,d_m^i\} \text{ for } i,j = 1,\dots,n\}$$

and

$$S_2 = \{(\boldsymbol{x}, \boldsymbol{w}) : \boldsymbol{x}, \boldsymbol{w} \text{ defined by } (4.23a) - (4.23d), (4.23f) - (4.23g) \text{ and } (4.24)\}.$$

We need to prove that $S_1 = S_2$.

Following the proof of Theorem 4.2.2, it is easy to know that any $(\boldsymbol{x}, \boldsymbol{w}) \in S_1$ satisfies $(\boldsymbol{x}, \boldsymbol{w}) \in S_2$. Conversely, given $(\boldsymbol{x}, \boldsymbol{w}) \in S_2$, then there exist nonnegative vectors $\boldsymbol{u}^i = (u_1^i, \ldots, u_m^i), i = 1, \ldots, n$; $\boldsymbol{p}^{ij} = (p_1^{ij}, \ldots, p_m^{ij}), i, j = 1, \ldots, n$ with $i < j$ and 0-1 binary vectors $\boldsymbol{\lambda}^i = (\lambda_1^i, \ldots, \lambda_h^i), i = 1, \ldots, n$ such that (4.23a)-(4.23d), (4.23f)-(4.23g) and (4.24) hold. Next we verify that $(\boldsymbol{x}, \boldsymbol{w}) \in S_1$, similar to the proof of Theorem 4.2.2.

For $i = 1, \ldots, n$, let $K_i = \{k \in \{1, \ldots, h\} : \lambda_k^i = 1\}$ be an index set and define $l(i) = 1 + \sum\limits_{k \in K_i} 2^{k-1}$. Obviously, $l(i)$ is the unique index that ensures $k \in K_i$ if and only if $l(i) \in G_k$. For any $l' \in \{1, \ldots, n\}$ with $l \neq l(i)$, there exists some $k' \notin K_i$ such that $l' \in G_{k'}$. Then, (4.23b)-(4.23c) implies that $u_{l(i)}^i = 1$, $u_{l'}^i = 0$ for $l' \neq l(i)$, and $x_i = d_{l(i)}^i$, for $i = 1, \ldots, n$.

Similarly, given $j = 2, \ldots, n$, we have the following results for all $l' \neq l(j)$:

$$p_{l'}^{ij} \leqslant \sum_{i<j} \sum_{l \in G_{k'}} p_l^{ij} \leqslant \sum_{i<j} d_m^i \lambda_{k'}^j = 0, \ i = 1, \ldots, n \text{ with } i < j,$$

and the non-negativity of $p_{l'}^{ij}$ implies

$$p_{l'}^{ij} = 0, \text{ for } l' \neq l(j); \ i = 1, \ldots, n \text{ with } i < j. \tag{4.25}$$

Plugging (4.25) into (4.23d), we get $p_{l(j)}^{ij} = x_i$ and $w_{ij} = \sum_{l=1}^m d_l^j p_l^{ij} = x_i d_{l(j)}^j = x_i x_j$, for $i = 1, \ldots, n$ with $i < j$. Therefore, we obtain $(\boldsymbol{x}, \boldsymbol{w}) \in S_1$. □

Compared with the reformulation in Theorem 4.6.2, the number of Big-M constraints in the refined reformulation is reduced from $n(n-1)h/2$ to $(n-1)h$. This improvement motivates us to further investigate the connection among multiple quadratic terms.

In practice, it is very likely that not all of the variables x_i and quadratic terms $x_i x_j$ appear in DQP. A special case is when the coefficients of $\boldsymbol{A}$ and $\boldsymbol{c}$ are sparse. Note that each quadratic term $x_i x_j$ associates with a set of Big-M constraints in terms of either $\boldsymbol{\lambda}^i$ or $\boldsymbol{\lambda}^j$, and the Big-M constraints for quadratic terms sharing variables may be aggravated as shown in Theorem 4.6.2. Therefore, given a DQP problem with partial quadratic terms, properly choosing $\boldsymbol{\lambda}^i$ or $\boldsymbol{\lambda}^j$ to construct Big-M constraints for each term $x_i x_j$ could possibly reduce the total number of Big-M constraints required in the reformulation by the Super Logarithmic Method.

Given a DQP problem with coefficients $\boldsymbol{A} \in \mathcal{S}^n$ and $\boldsymbol{c} \in \mathbb{R}^n$, we design a minimum vertex cover problem to minimize the number of Big-M constraints required in the reformulation of

DQP by the Super Logarithmic Method. Specifically, define a graph $G = (V, E)$ with vertex set $V = \{i \in \{1, \ldots, n\} |\ c_i \neq 0 \text{ or } A_{ij} \neq 0 \text{ for some } j = 1, \ldots, n\}$ and edge set $E = \{(i, j) |\ A_{ij} \neq 0,\ i, j = 1, \ldots, n,\ i < j\}$. A vertex cover of graph G is a subset $VC \subseteq V$ such that each edge of G is incident to at least one vertex of VC. Then, a minimum vertex cover of G is a vertex cover G with the smallest number of vertices, denoted by VC^*. The problem of finding a minimum vertex cover of a graph is a typical example of NP-hard optimization problems with efficient approximation algorithms available. Given the minimum vertex cover VC^* of G, for any quadratic term $x_i x_j$ appeared in DQP, we have either $i \in VC^*$ or $j \in VC^*$. Therefore, we only need to generate Big-M constraints using variables $\boldsymbol{\lambda}^j$, $j \in VC^*$ for linearizing DQP. The corresponding reformulation of DQP is stated in the next theorem.

Theorem 4.6.4. *Given the DQP problem (4.22) and its corresponding graph $G = (V, E)$, let VC^* be a minimum vertex cover of G. For each $j \in VC^*$, define $V_j = \{i \in VC^* |\ (i, j) \in E\} \cup \{i \in V \backslash VC^* |\ (i, j) \in E \text{ or } (j, i) \in E\}$. Then DQP is equivalent to the following 0-1 mixed-integer linear programming problem:*

$$\min \quad \frac{1}{2} \sum_{i \in V} \sum_{l=1}^{m} A_{ii} (d_l^i)^2 u_l^i + \sum_{(i,j) \in E} A_{ij} w_{ij} + \sum_{i \in V} c_i x_i \tag{4.26a}$$

$$s.t. \quad x_i = \sum_{l=1}^{m} d_l^i u_l^i, \quad \sum_{l=1}^{m} u_l^i = 1, \ \textit{for } i \in V \tag{4.26b}$$

$$\sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \textit{for } i \in V, \ k = 1, \ldots, h, \tag{4.26c}$$

$$x_i = \sum_{l=1}^{m} p_l^{ij}, \quad \sum_{l=1}^{m} d_l^j p_l^{ij} = w_{ij}, \ \textit{for } j \in VC^*, i \in V_j, \ k = 1, \ldots, h, \tag{4.26d}$$

$$\sum_{i \in V_j} d_m^i (\lambda_k^j - 1) + \sum_{i \in V_j} x_i \leqslant \sum_{i \in V_j} \sum_{l \in G_k} p_l^{ij} \leqslant \sum_{i \in V_j} d_m^i \lambda_k^j, \ \textit{for } j \in VC^*, \ k = 1, \ldots, h, \tag{4.26e}$$

$$\boldsymbol{x} \in \mathbb{R}_+^{|V|}, \ \boldsymbol{u}^i \in \mathbb{R}_+^m, \ \boldsymbol{\lambda}^i \in \{0, 1\}^h, \ \textit{for } i \in V, \tag{4.26f}$$

$$w_{ij} \geq 0, \ \boldsymbol{p}^{ij} \in \mathbb{R}_+^m, \ \textit{for } j \in VC^*, i \in V_j, \tag{4.26g}$$

Proof. Denote sets

$$S_1 = \{(\boldsymbol{x}, \boldsymbol{w}) : x_i \in \{d_1^i, \ldots, d_m^i\} \text{ for } i \in V, \ w_{ij} = x_i x_j, \ \text{for } (i, j) \in E\}$$

and

$$S_2 = \{(\boldsymbol{x}, \boldsymbol{w}) : \boldsymbol{x}, \boldsymbol{w} \text{ defined by } (4.26b) - (4.26g)\}.$$

The proof of $S_1 = S_2$ follows that in the proof of Theorem 4.6.3. □

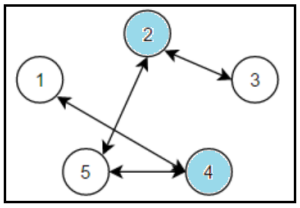
The number of Big-M constraints required in the reformulation given by Theorem 4.6.4 is $|VC^*|h$, compared to $(n-1)h$ as in Theorem 4.6.3. Note that $|VC^*| \leq (n-1)$ always holds. In particular, the difference between $|VC^*|$ and $(n-1)$ could be large for DQP with sparse or network-structure involved coefficients. Next we use a small-size example to illustrate the process of generating the reformulation of DQP in Theorem 4.6.4 and compare with the other two reformulations in Theorems 4.6.2 and 4.6.3, respectively.

Example 4.6.1. Consider the following example:

$$\begin{aligned} \min \quad & 5x_1x_4 - x_2x_3 + x_2x_5 - 4x_4x_5 \\ \text{s.t.} \quad & x_i \in \{1, \ldots, 10\}, \quad i = 1, \ldots, 5. \end{aligned}$$

This DQP problem only involves partial quadratic terms. The associated graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5\}$ and $E = \{(1, 4), (2, 3), (2, 5), (4, 5)\}$ is drawn in the Figure 4.4. We can easily find a minimum vertex cover $\{2, 4\}$ for G. According to Theorem 4.6.4, the corresponding 0-1 mixed-integer linear programming reformulation of this DQP example is presented as below.

Figure 4.4: The graph G associated with DQP example



$$\begin{aligned} \min \quad & 5w_{14} - w_{23} - w_{25} + w_{45} \\ \text{s.t.} \quad & x_i = \sum_{l=1}^{10} l u_l^i, \quad \sum_{l=1}^{10} u_l^i = 1, \text{ for } i = 1, \ldots, 5, \\ & \sum_{l \in G_k} u_l^i = \lambda_k^i, \quad \text{for } i = 1, \ldots, 5, \ k = 1, \ldots, 4, \\ & x_1 = \sum_{l=1}^{10} p_l^{14}, \quad \sum_{l=1}^{10} l p_l^{14} = w_{14}, \qquad x_5 = \sum_{l=1}^{10} p_l^{45}, \quad \sum_{l=1}^{10} l p_l^{45} = w_{45}, \\ & x_3 = \sum_{l=1}^{10} p_l^{23}, \quad \sum_{l=1}^{10} l p_l^{23} = w_{23}, \qquad x_5 = \sum_{l=1}^{10} p_l^{25}, \quad \sum_{l=1}^{10} l p_l^{25} = w_{25}, \end{aligned}$$

$$200(\lambda_k^4 - 1) + x_1 + x_5 \leqslant \sum_{l \in G_k} p_l^{14} + \sum_{l \in G_k} p_l^{45} \leqslant 200\lambda_k^4, \text{ for } k = 1, \ldots, 4,$$
$$200(\lambda_k^2 - 1) + x_3 + x_5 \leqslant \sum_{l \in G_k} p_l^{23} + \sum_{l \in G_k} p_l^{25} \leqslant 200\lambda_k^2, \text{ for } k = 1, \ldots, 4,$$
$$x_i,\ \boldsymbol{u}^i \in \mathbb{R}_+^{10},\ \boldsymbol{\lambda}^i \in \{0,1\}^4,\ \text{for } i = 1, \ldots, 5,$$
$$w_{ij} \geq 0,\ \boldsymbol{p}^{ij} \in \mathbb{R}_+^{10},\ \text{for } (i,j) \in \{(1,4),(2,3),(2,5),(4,5)\},$$

where $G_1 = \{2,4,6,8\}, G_2 = \{3,4,7,8,10\}, G_3 = \{5,6,7,8\}$ and $G_4 = \{9,10\}$.

Two other related reformulations using the Super Logarithmic Method can be derived according to Theorems 4.6.2 and 4.6.3, respectively. The comparison of these three reformulations is shown in the next table. From Table 4.6, we see that the number of required Big-M constraints can be significantly reduced by using the aggravation technique and taking problem's network structure into consideration.

Table 4.6: Comparison of three SLM based reformulations of DQP example

Reformulations	# of 0-1 variables	# of continuous variables	# of equality constraints	# of inequality constraints
Theorem 4.6.2	20	99	38	80
Theorem 4.6.3	20	99	38	16
Theorem 4.6.4	20	99	38	8

From the above discussions, the Super Logarithmic Method can be customized to provide an effective 0-1 mixed-integer linear programming reformulation for DQP. In the case when the coefficients of DQP are sparse or have special network structures, the Big-M constraints in the formulation may be further reduced for better efficiency.

4.7 Summary

In this chapter, we have proposed a Super Logarithmic Method for linearizing signomial programming problems with discrete variables (DSP). The proposed method requires half an order fewer Big-M inequality constraints to formulate a 0-1 mixed integer linear program than the Logarithmic Method, which is the state-of-the-art method for linearizing DSP problems. Since the Big-M inequality constraints significantly affect the computational effectiveness, the proposed Super Logarithmic Method is shown to be more effective. In addition, the Super Logarithmic Method has been applied to treat more complicated product terms that may contain continuous

variables and fractional terms. Accordingly, we provide a class of "representable programming problems" to which the Super Logarithmic Method is applicable. The computational results support our theoretical findings. We have also customized the Super Logarithmic Method to linearly represent DQP problems and proposed potential enhancement of the reformulation by examining network structures embedded in the DQP problem.

Chapter 5

l^1-norm Constrained Convex Quadratic Programming with Discrete Variables

In previous chapters, we have studied two important methods for quadratic programming with discrete variables (DQP). One is a linear conic relaxation approach for finding approximate solutions to DQP, while the other is a linearization method that provides effective reformulations for finding exact solutions to DQP. For a closely related DQP problem, called l^1-norm constrained convex quadratic programming with integer variables (l^1-DQP), the proposed two methods are naturally applicable. In this chapter, a new type of mixed-integer conic cuts is further developed for solving (l^1-DQP) by exploiting the conic structure embedded in the l^1-norm constraint. We incorporate the proposed conic cuts into a branch-and-bound scheme to find exact solutions. Numerical results show that the new conic cuts dominate the known cuts for improving the computational efficiency of the branch-and-bound scheme.

5.1 Introduction

Since the last two decades, the l^1-norm optimizer has been widely applied in statistics and engineering areas due to the favorable properties of sparsity and outlier-robustness associated with the l^1-norm. Consider the following l^1-norm constrained convex quadratic programs with integer variables:

$$(l^1\text{-DQP}) \qquad \begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} \\ \text{s.t.} \quad & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leqslant r \end{aligned} \tag{5.1}$$

$$\boldsymbol{x} \in \mathbb{Z}^n \cap [\boldsymbol{l}, \boldsymbol{u}],$$

where $\boldsymbol{Q} \in \mathcal{S}^n$ is a positive semi-definite symmetric matrix, $\boldsymbol{A} \in \mathbb{R}^{p \times n}$ is a matrix, $\boldsymbol{q} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^p$ are vectors, $r \in \mathbb{R}_+$, $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$ and $\mathbb{Z}^n$ is the set of n-dimensional integer vectors. Without loss of generality, we may assume that $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n_+$, and hence $\boldsymbol{x} \in \mathbb{Z}^n_+ \cap [\boldsymbol{l}, \boldsymbol{u}]$. Since the l^1-norm function is piecewise linear, we may reformulate (l^1-DQP) as a linearly constrained DQP problem. The DQP solution concepts reviewed in Chapter 2 can then be considered. In particular, the linear conic relaxation approach and linearization method that we have proposed in Chapters 3 and 4 are directly applicable here.

However, (l^1-DQP) is more than a general linearly constrained DQP problem. Geometrically speaking, the l^1-norm constraint represents a cross section of a "first-order cone" (FOC in short), which shares similar conic structures with the "second-order cone" (SOC in short) that has been extensively studied in linear conic programming theory. The solution concepts of mixed-integer second-order cone programming (MISOCP) may become valuable to the study of (l^1-DQP). In addition, exploiting the conic structure embedded in the l^1-norm constraint may allow us to develop new solution methods for solving (l^1-DQP).

MISOCP has wide applications in engineering and operations management, such as network design, portfolio optimization and transportation scheduling. As a result of the advanced research on polynomial-time interior-point algorithms of second-order cone programs (SOCP), commercial SOCP-solvers have been specifically designed and widely adopted for solving practical problems. To find an exact solution to MISOCP, a branch-and-bound scheme that solves an SOCP as a continuous relaxation at each node of the branching tree can be adopted. For computational efficiency of the branch-and-bound scheme, an important technique is incorporating effective conic cuts induced by the integer requirement to generate better lower bounds for branching. In the recent literature, several nonlinear conic cuts of MISOCP have been developed. One is a class of mixed-integer SOC rounding cuts [7], developed from a two-dimensional polyhedral SOC representation of an original SOC constraint. And the other is the so-called "disjunctive SOC inequality" designed for representing the convex hull of a disjunctive SOC set [8]. It is notable that these mixed-integer nonlinear cuts are specific to the nonlinear(conic) structure of MISOCP.

Motivated by the research on mixed-integer conic cuts of MISOCP, in this chapter, we explore some new mixed-integer conic cuts for (l^1-DQP) and investigate the computational efficiency of incorporating the conic cuts into a branch-and-bound scheme. Since the convex quadratic objective function of (l^1-DQP) is SOC representable, replacing the objective function by an auxiliary variable and adding a corresponding SOC constraint, we may obtain some mixed-integer SOC cuts of (l^1-DQP) easily. Note that the l^1-norm constraint captures an FOC-based conic structure and can be represented by a set of two-dimensional polyhedral FOC

constraints, which is similar to the representation of an SOC constraint for deriving mixed-integer SOC rounding cuts. Hence, we extend the method in [7] to generate mixed-integer FOC cuts based on the l^1-norm constraint and integer requirement of the problem. Observe that a two-dimensional polyhedral FOC constraint is also an SOC constraint. We further aggregate the quadratic objective function and l^1-norm constraint to derive comprehensive mixed-integer conic cuts for (l^1-DQP). The proposed comprehensive conic cuts are expected to be more effective than the conventional SOC cuts.

Numerical experiments are conducted to test the impact of adding conventional SOC cuts and new comprehensive conic cuts on the computational efficiency of a branch-and-bound algorithm. Numerical results show that the comprehensive conic cuts are more effective than the SOC cuts in improving the computational efficiency of solving (l^1-DQP).

The contributions of this study are two-fold. First, we initiate the research of exploring mixed-integer conic cuts that are specific to the conic structure of (l^1-DQP). Second, our preliminary numerical results provide insights for incorporating the proposed conic cuts to design effective branch-and-bound algorithms for solving (l^1-DQP).

The rest of this chapter is organized as follows. In Section 5.2, we introduce some basic properties of l^1-norm constraint and mixed-integer conic rounding cuts. In Section 5.3, we propose a new type of mixed-integer conic cuts of (l^1-DQP). In Section 5.4, numerical experiments are conducted to test the performance of proposed cuts for solving (l^1-DQP). Section 5.5 concludes this chapter with several remarks.

5.2 Preliminaries

In this section, we introduce some basic properties of l^1-norm constraints and review the methods of generating mixed-integer SOC rounding cuts [7].

5.2.1 Basic properties of l^1-norm constraints

For easy illustrations, here we consider a simple l^1-norm constraint $\|\boldsymbol{x}\|_1 \leq 1$. The discussions may naturally apply to the general l^1-norm constraint shown in (5.1).

Recall that the l^p-norm of a vector $\boldsymbol{x} \in \mathbb{R}^n$ is given by $\|\boldsymbol{x}\|_p = (\sum_{i=1}^n |x_i|^p)^{1/p}$, for $p = 1, 2, 3, \ldots$. Consequently, the l^1-norm function is piecewise linear. In general, we have three different ways to linearly represent the l^1-norm constraint $\|\boldsymbol{x}\|_1 \leq 1$:

(i) From the linear realization of $|x_1|, \ldots, |x_n|$, an equivalent linear system with 2^n facet constraints can be expressed as

$$\boldsymbol{w}^T \boldsymbol{x} \leq 1, \text{ for } \boldsymbol{w} \in \{-1, 1\}^n.$$

(ii) For each variable x_i, introducing a two-dimensional l^1-norm constraint $|x_i| \leq t_i$ with the auxiliary variable $t_i \in \mathbb{R}_+$, $i = 1, \ldots, n$, we have $\sum_{i=1}^n t_i \leq 1$. The two-dimensional constraints can also be written as

$$-t_i \leq x_i \leq t_i, \text{ for } i = 1, \ldots, n.$$

(iii) Since the feasible domain of $\|\boldsymbol{x}\|_1 \leq 1$ is a bounded polyhedron formed as the convex hull of its extreme points [34], the constraint can then be replaced by $\boldsymbol{x} = \boldsymbol{\lambda} - \boldsymbol{\mu}$, where

$$\sum_{i=1}^{n} (\lambda_i + \mu_i) = 1, \text{ and } \lambda_i, \mu_i \geq 0, \text{ for } i = 1, \ldots, n.$$

The above three linear systems are equivalent, yet with respective meanings. We shall adopt a proper system to represent the l^1-norm constraint in later sections.

The cone induced by the l^1-norm is called a first-order cone (FOC) defined by $\{(\boldsymbol{x}, t_0) \in \mathbb{R}^n \times \mathbb{R}_+ : \|\boldsymbol{x}\|_1 \leq t_0\}$. Notice that FOC is a closed, convex and pointed polyhedral cone and the domain of the l^1-norm constraint $\|\boldsymbol{x}\|_1 \leq 1$ is the cross section of FOC with $t_0 = 1$.

5.2.2 Mixed-integer conic rounding cuts

In this subsection, we briefly review a method of generating mixed integer conic rounding cuts [7]. Consider the following mixed-integer second-order cone (SOC) set:

$$C := \{(\boldsymbol{x}, t_0) \in \mathbb{Z}_+^n \times \mathbb{R}_+ : \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leq t_0\},$$

where $\boldsymbol{A} \in \mathbb{R}^{p \times n}$ is a real matrix and $\boldsymbol{b} \in \mathbb{R}^p$ is a real vector. The continuous relaxation domain of C is defined as $\text{relax}(C) := \{(\boldsymbol{x}, t_0) \in \mathbb{R}_+^n \times \mathbb{R}_+ : \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leq t_0\}$. A valid inequality of C induced by the integer requirement that may cut off some portion of $\text{relax}(C)$ is called a mixed-integer cut of C.

Introducing auxiliary variables $\boldsymbol{t} = (t_1, \ldots, t_p) \in \mathbb{R}^p$, the SOC constraint $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leq t_0$ can be reformulated as

$$|\boldsymbol{A}_i \boldsymbol{x} - b_i| \leq t_i, \ i = 1, \ldots, p \tag{5.2a}$$

$$\|\boldsymbol{t}\|_2 \leq t_0, \tag{5.2b}$$

where $\boldsymbol{A}_i$ denotes the ith row of matrix $\boldsymbol{A}$, $i = 1, \ldots, p$. Constraints in the form of (5.2a) are called "*polyhedral SOC constraints*". Based on this reformulation, mixed-integer rounding procedures are conducted on each polyhedral SOC constraint of (5.2a) to generate rounding inequalities for C. For $i = 1, \ldots, p$, consider the mixed-integer domain $C_i := \{(\boldsymbol{x}, t_i) \in \mathbb{Z}_+^n \times \mathbb{R}_+ :$

$|\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i\}$ associated with the i-th polyhedral SOC constraint. The continuous relaxation C_i is given by $\text{relax}(C_i) := \{(\boldsymbol{x}, t_i) \in \mathbb{R}^n_+ \times \mathbb{R}_+ : |\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i\}$.

Definition 1 (Ref. Definition 1, [7])**.** For $0 \leq f < 1$, the mixed-integer conic rounding function $\phi_f : \mathbb{R} \to \mathbb{R}$ is defined by

$$\phi_f(c) = \begin{cases} (1-2f)\lfloor c \rfloor - (c - \lfloor c \rfloor), & \text{if } c - \lfloor c \rfloor < f, \\ (1-2f)\lfloor c \rfloor + (c - \lfloor c \rfloor) - 2f, & \text{if } c - \lfloor c \rfloor \geq f. \end{cases}$$

Here we may directly extend the domain of ϕ_f to a vector space such that for any given $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{y} \triangleq \phi_f(\boldsymbol{c}) \in \mathbb{R}^n$ with $y_i = \phi_f(c_i)$, $i = 1, \ldots, n$. The main result of SOC rounding inequalities is then given below.

Theorem 5.2.1 (Ref. Theorem 1 and Proposition 4, [7])**.** *Given $\boldsymbol{A} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{b} \in \mathbb{R}^p$, for $i = 1, \ldots, n$, the mixed-integer domain $C_i := \{(\boldsymbol{x}, t_i) \in \mathbb{Z}^n_+ \times \mathbb{R}_+ : |\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i\}$ has a valid inequality*

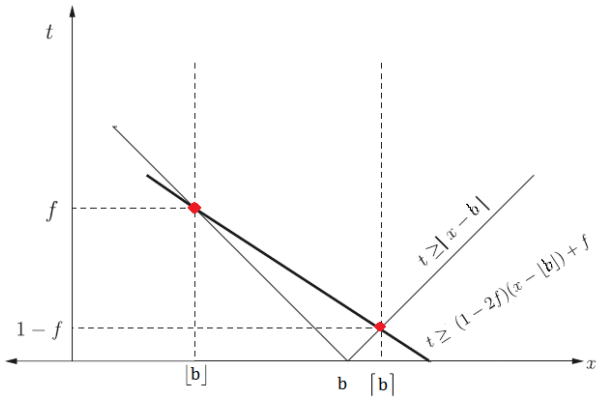
$$\left(\phi_{f_\alpha}(\frac{\boldsymbol{A}_i^T}{\alpha})\right)^T \boldsymbol{x} - \phi_{f_\alpha}(\frac{b_i}{\alpha}) \leq \frac{t_i}{|\alpha|} \tag{5.3}$$

with $\alpha \neq 0$ and $f_\alpha = \frac{b_i}{\alpha} - \lfloor \frac{b_i}{\alpha} \rfloor$. Moreover, the valid inequalities (5.3) with $\alpha = A_{ij}, j = 1, \ldots, n$ are sufficient to cut off all fractional extreme points of the continuous relaxation of domain C_i.

Plugging the valid inequalities (5.3) to the reformulation (5.2a)-(5.2b), we may obtain nonlinear conic rounding inequalities for the original mixed-integer set C. Take a simple example $C := \{(x, y, t_0) \in \mathbb{Z} \times \mathbb{R} \times \mathbb{R}_+ : \sqrt{(x-b)^2 + y^2} \leq t_0\}$ for instance. Here the integer polyhedral SOC set is $C_1 := \{(x, t) \in \mathbb{Z} \times \mathbb{R}_+ : |x - b| \leq t\}$. The conic rounding inequality with $\alpha = 1$ is then given by $(1-2f)(x - \lfloor b \rfloor) + f \leq t$, where $f := b - \lfloor b \rfloor$. From Figure 5.1, we see that the conic rounding inequality cuts off all points in $\text{relax}(C_1)\backslash\text{conv}(C_1)$. It further yields a nonlinear mixed-integer conic rounding inequality $\sqrt{[(1-2f)(x - \lfloor b \rfloor) + f]^2 + y^2} \leq t_0$ for C, which is valid and cuts off all points in $\text{relax}(C)\backslash\text{conv}(C)$.

In theory, the mixed-integer conic rounding inequalities (5.3) based on the reformulation (5.2a)-(5.2b) are not guaranteed to provide a valid cut-off of the continuous relaxation of the original mixed-integer domain C. However, numerical results in [7] support that the mixed-integer conic rounding inequalities are capable of providing effective conic cuts that can reduce the continuous relaxation gap and improve the solvability of conic mixed-integer programs. We shall extend the work in [7] to generate some new mixed-integer conic cuts for (l^1-DQP).

Figure 5.1: Simple conic rounding cut for $|x - b| \leq t$



5.3 Conic Cuts of (l^1-DQP)

Since the problem (l^1-DQP) involves integer variables, a branch-and-bound scheme is most likely to be employed for finding exact solutions of (l^1-DQP). When such a scheme is applied, at each node of the branching tree, we need to solve a continuous relaxation problem based on the linearization of l^1-norm constraint for finding a lower bound. To improve the efficiency of the branch-and-bound scheme, we propose some new conic cuts induced by the integer requirement of variables to tighten the continuous relaxation at the root node.

For the purpose of illustration, we use the following form of (l^1-DQP) in which the convex quadratic objective function is written as an l^2-norm function:

$$
\begin{aligned}
& & \min \quad & \|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{g}\|_2 && \text{(5.4a)} \\
& (l^1\text{-DQP}) & \text{s.t.} \quad & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leqslant r && \text{(5.4b)} \\
& & & \boldsymbol{x} \in \mathbb{Z}^n_+ \cap [\boldsymbol{l}, \boldsymbol{u}] \ i = 1, \ldots, n,
\end{aligned}
$$

where $\boldsymbol{B} \in \mathbb{R}^{n \times n}$, $\boldsymbol{g} \in \mathbb{R}^n$ and $\boldsymbol{A}, \boldsymbol{b}, r, \boldsymbol{l}, \boldsymbol{u}$ follow those defined in (5.1).

5.3.1 Conventional SOC cuts

Since the convex quadratic objective function is SOC representable, mixed-integer SOC rounding cuts can be first derived from the objective function (5.4a) following [7]. More specifically, we may introduce an SOC constraint from the original objective function and rewrite (l^1-DQP) as

$$\min \quad v_0$$

$$\begin{aligned} \text{s.t.} \quad & \|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{g}\|_2 \leq v_0 \\ & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leqslant r \\ & v_0 \in \mathbb{R}_+,\ \boldsymbol{x} \in \mathbb{Z}^n_+ \cap [\boldsymbol{l}, \boldsymbol{u}]. \end{aligned}$$

Note that the SOC constraint $\|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{g}\|_2 \leq v_0$ has the following reformulation:

$$\begin{cases} |\boldsymbol{B}_j\boldsymbol{x} - g_j| \leq v_j,\ j = 1, \ldots, n \\ \|\boldsymbol{v}\|_2 \leq v_0,\ \boldsymbol{v} = (v_1, \ldots, v_n) \in \mathbb{R}^n_+, \end{cases} \tag{5.5}$$

where $\boldsymbol{B}_j$ is the j-th row of matrix $\boldsymbol{B} \in \mathbb{R}^{n\times n}$, $j = 1, \ldots, n$. For each $j = 1, \ldots, n$, based on the integer requirement on $\boldsymbol{x}$, we can then generate mixed-integer linear rounding inequalities in terms of $(\boldsymbol{x}, \boldsymbol{v})$ as below according to Theorem 5.2.1.

$$\left(\phi_{f_\alpha}(\frac{\boldsymbol{B}_j^T}{\alpha})\right)^T \boldsymbol{x} - \phi_{f_\alpha}(\frac{g_j}{\alpha}) \leq \frac{v_j}{|\alpha|}, \tag{5.6}$$

where $f_\alpha = \frac{g_j}{\alpha} - \lfloor \frac{g_j}{\alpha} \rfloor$ with $\alpha > 0$. In the reformulation (5.5) there remains a simple SOC constraint $\|\boldsymbol{v}\|_2 \leq v_0$ on the auxiliary variable $\boldsymbol{v} \in \mathbb{R}^n_+$. Therefore, projected onto the space of $(\boldsymbol{x}, v_0)$, the rounding inequalities (5.6) turn to be in a quadratic form, called "SOC conic inequalities". An SOC conic inequality that may cut off a fractional solution is denoted as a mixed-integer SOC conic cut.

5.3.2 New conic cuts

In this subsection, we show that the l^1-norm constraint (5.4b) provides a new source to generate mixed-integer conic inequalities for (l^1-DQP). From Section 5.2.1, $\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|_1 \leq r$ can be linearly reformulated as

$$\begin{cases} |\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i,\ i = 1, \ldots, p \\ \boldsymbol{e}^T\boldsymbol{t} \leq r,\ \boldsymbol{t} = (t_1, \ldots, t_p) \in \mathbb{R}^p_+, \end{cases} \tag{5.7}$$

where $\boldsymbol{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^p$ and $\boldsymbol{A}_i$ is the i-th row of matrix $\boldsymbol{A} \in \mathbb{R}^{p\times n}$, $i = 1, \ldots, p$. The linear system (5.7) uses similar polyhedral constraints as that in (5.5). For each $i = 1, \ldots, p$, we may then follow Theorem 5.2.1 to generate the following mixed-integer linear rounding inequalities in term of $(\boldsymbol{x}, \boldsymbol{t})$:

$$\left(\phi_{f_\alpha}(\frac{\boldsymbol{A}_i^T}{\alpha})\right)^T \boldsymbol{x} - \phi_{f_\alpha}(\frac{b_i}{\alpha}) \leq \frac{t_i}{|\alpha|}, \tag{5.8}$$

where $f_\alpha = \frac{b_i}{\alpha} - \lfloor \frac{b_i}{\alpha} \rfloor$ with $\alpha > 0$. The piecewise linearity of l^1-norm constraint is thoroughly delivered to the polyhedral constraints in terms of $(\boldsymbol{x}, \boldsymbol{t})$, while the remaining restriction on the auxiliary variable $\boldsymbol{t} \in \mathbb{R}^p_+$ is linearity only. Consequently, the rounding inequalities with respect to $\boldsymbol{x}$ are called "FOC conic inequalities". An FOC conic inequality that cuts off a fractional solution is denoted as a mixed-integer FOC conic cut.

Both SOC and FOC conic cuts preserve the structure of constraints where they are derived from. Given a continuous relaxation solution of (l^1-DQP), an SOC conic cut generated from the objective function $\|\boldsymbol{B}\boldsymbol{x} - \boldsymbol{g}\|_2$ guarantees a direct improvement of the objective value, while an FOC conic cut derived from $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 \leqslant r$ tightens the relaxation of the integral feasible domain. The improvement of the objective value and a tighter approximation of the feasible domain are both important in a branch-and-bound scheme. Next we propose a new type of conic cut for (l^1-DQP) that may capture both features.

The idea is motivated by a conic aggregation strategy that was introduced in [7] for practical implementations. For instance, given $\boldsymbol{A} \in \mathbb{R}^{p\times n}$ and $\boldsymbol{b} \in \mathbb{R}^p$, consider the polyhedral SOC constraints $|\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i$, $i = 1, \ldots, p$ with variables $\boldsymbol{x} \in \mathbb{Z}^n_+$ and $\boldsymbol{t} \in \mathbb{R}^p_+$. For $i = 1, \ldots, p$, taking μ_i and λ_i as the multipliers on the left and right terms of $-t_i \leq \boldsymbol{A}_i\boldsymbol{x} - b_i \leq t_i$, respectively, we have $-\mu_i t_i \leq \mu_i(\boldsymbol{A}_i\boldsymbol{x} - b_i)$ and $\lambda_i(\boldsymbol{A}_i\boldsymbol{x} - b_i) \leq \lambda_i t_i$. The p polyhedral SOC constraints can then be aggregated into the following single polyhedral SOC constraint:

$$\left| \left(\frac{\boldsymbol{\mu}+\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{A}\boldsymbol{x} + \left(\frac{\boldsymbol{\mu}-\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{t} - \left(\frac{\boldsymbol{\mu}+\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{b} \right| \leq \left(\frac{\boldsymbol{\mu}-\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{A}\boldsymbol{x} + \left(\frac{\boldsymbol{\mu}+\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{t} - \left(\frac{\boldsymbol{\mu}-\boldsymbol{\lambda}}{2}\right)^T \boldsymbol{b}, \tag{5.9}$$

where $\boldsymbol{\mu}, \boldsymbol{\lambda} \in \mathbb{R}^p_+$. Subsequently, we may derive the corresponding mixed-integer conic rounding inequalities from (5.9) as below.

$$\left(\phi_{f_\alpha}(\frac{\bar{\boldsymbol{A}}}{\alpha}) - \frac{\underline{\boldsymbol{A}}}{|\alpha|}\right)^T \boldsymbol{x} - \left(\phi_{f_\alpha}(\frac{\bar{b}}{\alpha}) - \frac{\underline{b}}{|\alpha|}\right) \leq \frac{(\boldsymbol{\mu}+\boldsymbol{\lambda})^T\boldsymbol{t}}{|\alpha|}, \tag{5.10}$$

where $\bar{\boldsymbol{A}}, \underline{\boldsymbol{A}} \in \mathbb{R}^n$ and $\bar{b}, \underline{b} \in \mathbb{R}$ are given by $\begin{bmatrix} \bar{\boldsymbol{A}}^T & \bar{b} \\ \underline{\boldsymbol{A}}^T & \underline{b} \end{bmatrix} = \frac{1}{2}[\boldsymbol{\mu}+\boldsymbol{\lambda} \quad \boldsymbol{\mu}-\boldsymbol{\lambda}]^T[\boldsymbol{A} \quad \boldsymbol{b}]$, and $f_\alpha = \frac{\bar{b}}{\alpha} - \lfloor \frac{\bar{b}}{\alpha} \rfloor$ with $\alpha > 0$.

The conic aggregation (5.9) utilizes multiple polyhedral SOC constraints in deriving conic rounding inequalities. In the special case when $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ with $\boldsymbol{\mu} = \boldsymbol{\lambda}$ are unit vectors in $\mathbb{R}^p$, (5.9) degenerates into an original polyhedral SOC constraint $|\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i$, for some $i = 1, \ldots, p$. More discussions of the impact of aggregation multipliers $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ on the resulted conic inequality (5.10) shall be presented in the next subsection.

With the conic aggregation strategy in mind, observing that an FOC polyhedral constraint

is also an SOC polyhedral constraint, we may connect the objective function with l^1-norm constraint to derive more comprehensive conic inequalities. We gather all the polyhedral SOC constraints $|\boldsymbol{B}_j\boldsymbol{x} - g_j| \leq v_j,\ j = 1, \ldots, n$, from the objective function and polyhedral FOC constraints $|\boldsymbol{A}_i\boldsymbol{x} - b_i| \leq t_i,\ i = 1, \ldots, p$, from the l^1-norm constraint, and then conduct conic aggregations according to (5.9). The resulting linear rounding inequalities in terms of $(\boldsymbol{x}, \boldsymbol{v}, \boldsymbol{t})$ can be regarded as nonlinear conic inequalities in the original space of $(\boldsymbol{x}, v_0)$. A comprehensive conic inequality that may cut off a fractional solution is denoted as a mixed-integer comprehensive conic cut. Involving the information of both the objective function and the l^1-norm constraint, this new type of conic cut is promising to tighten the continuous approximation of the integer feasible domain along "good" directions that improve the current objective value.

In this subsection, we have developed new mix-integer conic cuts for (l^1-DQP), namely, the FOC cuts and comprehensive conic cuts. Since the comprehensive conic cuts are more general than FOC cuts, we shall focus on the comprehensive conic cuts in the rest of this study.

5.3.3 Analysis of conic aggregation strategy

Recall that the conic aggregation strategy unitizes several polyhedral SOC/FOC inequalities by using multipliers. In the numerical experiments of [7], the conic aggregation strategy was adopted on all possible pairs of constraints with all possible 0-1 valued multipliers. In this subsection, we examine the impact of multipliers $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ on the effectiveness of the resulting conic cuts to provide more insights for numerical implementation. For the purpose of illustration, we only discuss 0-1 valued multipliers as in [7].

Consider the following two-dimensional l^1-norm constraint on $\mathbb{Z}^2_+$:

$$|2.5435x_1 - 0.1738x_2 - 0.6685| + |1.0182x_1 + 0.9930x_2 - 0.8946| \leq 1, \tag{5.11}$$

where $x_1, x_2 \in \mathbb{Z}_+$. Let (λ_1, μ_1) be the multipliers for $|2.5435x_1 - 0.1738x_2 - 0.6685| \leq t_1$ and (λ_2, μ_2) be the multipliers for $|1.0182x_1 + 0.9930x_2 - 0.8946| \leq t_2$, respectively, where $t_1, t_2 \in \mathbb{R}_+$ satisfying $t_1 + t_2 \leq 1$, i.e.,

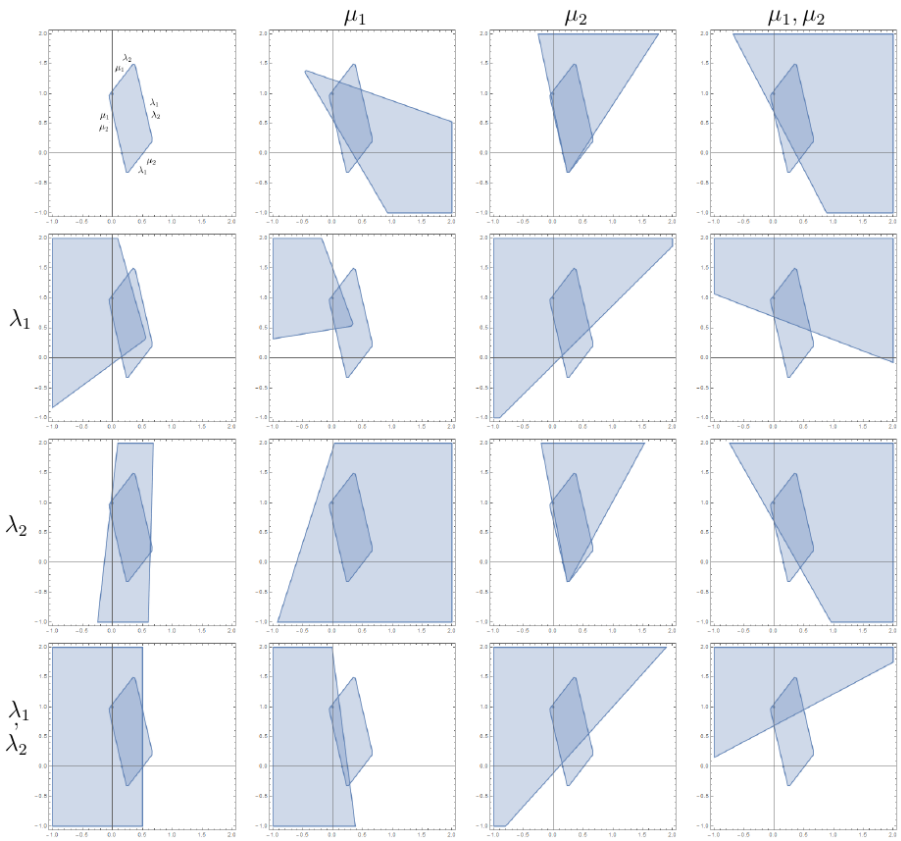
$$(\lambda_1, \mu_1): \quad |2.5435x_1 - 0.1738x_2 - 0.6685| \leq t_1, \tag{5.12a}$$

$$(\lambda_2, \mu_2): \quad |1.0182x_1 + 0.9930x_2 - 0.8946| \leq t_2. \tag{5.12b}$$

The continuous relaxation domain of (5.11) and the conic inequalities under various choices of $\lambda_1, \mu_1, \lambda_2, \mu_2 \in \{0, 1\}$ are drawn in Figure 5.2. The top row and left column indicate the non-zero multipliers in $\{\mu_1, \mu_2\}$ and $\{\lambda_1, \lambda_2\}$, respectively, which are called "active multipliers". In particular, the subfigure on the top left corner shows the continuous relaxation domain of (5.11), which contains a unique integer point $(0, 1) \in \mathbb{Z}^2_+$. And the four sides of this domain

are actually given by the aggregated inequalities with attached active multipliers, respectively. From Figure 5.2, we have the following observations:

Figure 5.2: Conic inequalities based on various 0-1 multipliers



(i) Consider the subfigure with active multipliers $\{\lambda_1, \mu_1\}$. The aggregated constraint is actually the polyhedral inequality (5.12a), whose corresponding conic inequality cuts off the top and down corners of the continuous relaxation domain as shown in the subfigure. Similarly, in the subfigure with active multipliers $\{\lambda_2, \mu_2\}$, the conic inequality derived from (5.12b) cuts off the left and right corners of the continuous relaxation domain.

(ii) In the four subfigures whose active multipliers are those indicated on the four sides of continuous relaxation domain, respectively, the conic inequality are skewed along the associated side of the continuous relaxation domain. This type of conic inequalities does not always provide a valid cut-off.

(iii) Given a continuous solution, for instance, $(x_1, x_2) = (0.1585, 0)$ on the left side of the con-

tinuous domain, the conic inequalities serve as conic cuts in six subfigures only, i.e., those with active multipliers $\{\mu_1\}, \{\mu_1, \mu_2\}, \{\lambda_1, \mu_1\}, \{\lambda_1, \mu_1, \mu_2\}, \{\lambda_2, \mu_1, \mu_2\}$ and $\{\lambda_1, \lambda_2, \mu_1, \mu_2\}$, respectively.

(a) Note that the realization of the l^1-norm vector is $(-0.2667, -0.7332)$, i.e., $t_1 = 0.2667$ and $t_2 = 0.7332$. Since $t_1 < t_2$, this point is closer to the corner associated with the first polyhedral inequality (5.12a). The conic inequality with active multipliers including λ_1 and μ_1 has high potential to cut off this continuous point, as verified in the corresponding subfigure.

(b) The continuous point stands on the left side of the continuous relaxation domain associated with multipliers including μ_1 and μ_2. Correspondingly, the subfigures in the right column all provide valid conic cuts.

The above observations imply that the performance of conic inequalities highly depends on the choice of multipliers. We can selectively choose multipliers to generate important conic inequalities for a given continuous solution, rather than considering all the possible multipliers. A separation heuristic of generating conic cuts from the conic inequalities shall be derived accordingly for numerical implementations in the next section.

5.4 Numerical Experiments and Results

In this section, we conduct numerical experiments to test the performance of the proposed comprehensive mixed-integer conic cuts and compare with that of conventional mixed-integer SOC cuts for solving (l^1-DQP). The impacts of conic cuts on the complexity of the branching tree and overall computational efficiency of finding an exact solution shall be examined.

The problem (l^1-DQP) in the form of a mixed-integer second-order cone program is given below.

$$
\begin{aligned}
\min \quad & v_0 \\
\text{s.t.} \quad & -v_j \leq \boldsymbol{B}_j \boldsymbol{x} - g_j \leq v_j, \ j = 1, \ldots, n, \\
& -t_i \leq \boldsymbol{A}_i \boldsymbol{x} - b_i \leq t_i, \ i = 1, \ldots, p, \\
& \boldsymbol{e}^T \boldsymbol{t} \leqslant r, \\
& \boldsymbol{x} \in \mathbb{Z}^n_+ \cap [\boldsymbol{l}, \boldsymbol{u}], \ (\boldsymbol{v}, v_0) \in \mathcal{L}_{n+1}, \ \boldsymbol{t} \in \mathbb{R}^p_+,
\end{aligned}
$$

where $\boldsymbol{e} = (1, 1, \ldots, 1)^T \in \mathbb{R}^p$ and $\mathcal{L}_{n+1} = \{(\boldsymbol{v}, v_0) \in \mathbb{R}^n \times \mathbb{R} : \|\boldsymbol{v}\|_2 \leq v_0\}$ is the $(n+1)$-dimensional second-order cone. Without loss of generality, we may assume that $l_1 = l_2 = \ldots, l_n = 0$ and $u_1 = u_2 = \ldots = u_n$, i.e., $\boldsymbol{x} \in [0, u]^n$ for some $u \in \mathbb{R}_+$. A set of (l^1-DQP) instances

with various values of $n, p \in \{40, 80\}$ and $u \in \{1, 5, 10\}$ are tested. For each test problem, the matrix $\boldsymbol{B}$ is randomly generated from the uniform distribution over $[0, 1]^{n\times n}$ and $\boldsymbol{g} = \boldsymbol{B}\boldsymbol{z}_0$ with $\boldsymbol{z}_0$ randomly generated from the uniform distribution over $[0, 2u]^n$. The matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$ are from the uniform distribution over $[0, 1]^{p\times n}$ and $[0, n]^p$, respectively. Meanwhile, to guarantee the feasibility of (l^1-DQP), r is given by $r = (1 - s)r_1 + sr_2$, where $r_1 = \min_{\mathbb{Z}^n_+\cap[0,u]^n} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$, $r_2 = \min\{\|\boldsymbol{b}\|_1, \|\boldsymbol{A}(u, \ldots, u)^T - \boldsymbol{b}\|_1\}$ and $s \in (0, 1)$. We later find that the choice of $s \in (0, 1)$ does not make a difference regarding to our conclusion of numerical results, hence $s = 0.8$ is adopted here. Accordingly, two hundred instances are randomly generated for each combination of n, p and u.

All experiments are run on a personal computer with 3.20 GHz Intel Core i7-5600U CPU and 8Gb of RAM. We implement the program in MATLAB R2015a using Gurobi [35] MISOCP solver with a feasibility/optimality tolerance of 1e-6. Gurobi solver is run with default options except that the cut generation option is turned off. In the experiments, we use a separation heuristic to generate conic cuts at the root node, from the SOC conic inequalities and comprehensive conic inequalities, respectively. The separation heuristic is a partial conic aggregation procedure that we propose in the next subsection.

5.4.1 Partial conic aggregation procedure

Recall the two types of mixed-integer conic cuts introduced in Section 5.3:

- SOC cuts: from $|\boldsymbol{B}_j\boldsymbol{x} - g_j| \le v_j$, $j = 1, \ldots, n$;
- Comprehensive conic cuts: from $|\boldsymbol{B}_j\boldsymbol{x} - g_j| \le v_j$, $|\boldsymbol{A}_i\boldsymbol{x} - b_i| \le t_i$, $j = 1, \ldots, n$, $i = 1, \ldots, p$.

In generating each type of conic cuts for the continuous relaxation at the root node, a direct way is using each single polyhedral constraint to construct a corresponding mixed-integer conic rounding inequality in the form of (5.6) or (5.8), and check for violation for each integer variable x_i with a fractional value. This *pure procedure* provides n^2 and n^2+np inequalities as candidates for SOC and Comprehensive conic cuts, respectively. To improve the quality of conic cuts, we may follow [7] to apply conic aggregation on pairs of polyhedral constraints using multipliers $\boldsymbol{\lambda}, \boldsymbol{\mu} \in \{0, 1\}^2$. In this way, the total numbers of conic inequalities as candidates of SOC and FOC cuts increase to $8n^2(n-1)$ and $8n^2(n-1)+8np(p-1)$, respectively. Compared with the *pure procedure*, this *complete conic aggregation procedure* will possibly aggravate the computational burden with much more inequalities to check and also much more cuts added in the continuous relaxation.

To address this numerical issue, from our discussion in Section 5.3.3, we may selectively choose polyhedral constraints for aggregation and use proper aggregation multipliers to separate

a given continuous solution. We first propose the following *partial conic aggregation procedure* to generate SOC cuts with a given continuous solution $\bar{\boldsymbol{x}} \in \mathbb{R}^n$:

Step 1: Let the index set $I = \{i \in \{1, \ldots, n\} |\, x_i \notin \mathbb{Z}\}$. Calculate the norm-vector $\bar{\boldsymbol{v}} = \boldsymbol{B}\bar{\boldsymbol{x}} - \boldsymbol{g} \in \mathbb{R}^n$ and sort $|\bar{v}_j|, j = 1, \ldots, n$ ascendingly. Find an index set $J \subset \{1, \ldots, n\}$ such that $|\bar{v}_j|$, $j \in J$ are the first quarter among the arranged values of $\{|\bar{v}_j|, j = 1, \ldots, n\}$.

Step 2: For each $i \in I$, generate conic cuts following the procedures as below.

(2.1) For each $j \in J$, generate the conic rounding inequality (5.6) from $|\boldsymbol{B}_j x - g_j| \leq v_j$ with $\alpha = B_{ij}$.

(2.2) Conduct conic aggregation on each pair of constraints among $|\boldsymbol{B}_j x - g_j| \leq v_j$, $j \in J$, using multipliers that are associated with $\bar{v}$. For instance, given $j, j' \in J$ with $j \neq j'$, if $\bar{v}_j > 0$ and $\bar{v}_{j'} < 0$, we select λ_j and $\mu_{j'}$ as active multipliers, i.e., $\boldsymbol{\lambda} = (\lambda_j, \lambda_{j'}) \in \{0, 1\}^2$ and $\boldsymbol{\mu} = (\mu_j, \mu_{j'}) \in \{0, 1\}^2$ with $\lambda_j = \mu_{j'} = 1$, to aggregate the two constraints of $|\boldsymbol{B}_j x - g_j| \leq v_j$ and $|\boldsymbol{B}_{j'} x - g_{j'}| \leq v_{j'}$.

(2.3) For each aggregated polyhedral constraint with index pair (j, j') and multipliers $\boldsymbol{\lambda}, \boldsymbol{\mu} \in \{0, 1\}^2$, generate the corresponding conic rounding inequality with $\alpha = \bar{B}_i$, where $\bar{\boldsymbol{B}} = \frac{1}{2}(\boldsymbol{\mu} + \boldsymbol{\lambda})^T [\boldsymbol{B}_j;\ \boldsymbol{B}_{j'}]$ is an n-dimensional row vector.

(2.4) Among all of the conic rounding inequalities generated from (2.1)-(2.3), pick up those that are violated by the given continuous solution $\bar{\boldsymbol{x}}$. If the number of violated inequalities is larger than five, then we choose those with the largest five violated values.

To generate the comprehensive conic cut, we slightly adjust the above partial conic aggregation procedure. At Step 1, calculate the norm-vector $\bar{\boldsymbol{t}} = \boldsymbol{A}\bar{\boldsymbol{x}} - \boldsymbol{b} \in \mathbb{R}^p$ and index set $K \subset \{1, \ldots, p\}$ such that $|\bar{t}_k|$, $k \in K$ are the smallest quarter among $\{|\bar{t}_k|, k = 1, \ldots, p\}$. Update $J = J \cup K$, $\boldsymbol{B} = [\boldsymbol{B}; \boldsymbol{A}]$, $\boldsymbol{g} = [\boldsymbol{g}; \boldsymbol{b}]$, $\bar{\boldsymbol{v}} = [\bar{\boldsymbol{v}}; \bar{\boldsymbol{t}}]$ and then go to Step 2.

This *partial conic aggregation procedure* selects representative conic inequalities among those generated from *complete conic aggregation strategy*. So the cost of generating conic cuts and the computational burden of solving the tightened problem will be relieved. In our numerical experiments, the SOC cuts and comprehensive conic cuts are added at the root node using this *partial conic aggregation procedure* as the separation heuristic, respectively.

5.4.2 Numerical results

For each problem size of $n, p \in \{40, 80\}$ and $u \in \{1, 5, 10\}$, the 200 randomly generated (l^1-DQP) instances shall be solved in three different ways: without adding any cut (Original in short), adding conventional SOC cuts (SOC-cut in short) and adding new comprehensive conic cuts

(Comp.-cut in short), respectively. We shall analyze the computational results from the following three aspects: (i) the possibility that conic cuts can be separated from the SOC inequalities and comprehensive conic inequalities, respectively, (ii) the computational cost of generating each type of conic cuts, and (iii) if SOC cuts or comprehensive conic cuts are generated, the reductions of the number of nodes explored in the searching tree and total CPU time of branch-and-bound algorithm, respectively.

Binary-valued instances

We first test the instance set with integer variables taking 0-1 binary values, i.e., $u = 1$. Among the 200 randomly generated instances of problem size $n = p = 40$, there are 134 instances (type-I) in which both the SOC rounding inequalities and comprehensive conic inequalities provide conic cuts at the root node, while the comprehensive conic cuts can be obtained in additional 15 instances (type-II). For the remaining 51 instances (type-III), no SOC cut or comprehensive conic cut is generated at the root node.

Table 5.1 presents the average results of using three different ways to solve (l^1-DQP), respectively. In particular, the CPU time of solving (l^1-DQP) with conic cuts added shown in Table 5.1 is the summation of cut generation time and branch-and-bound CPU time. One observation

Table 5.1: Computational results (average), $n = 40, m = 40, u = 1$

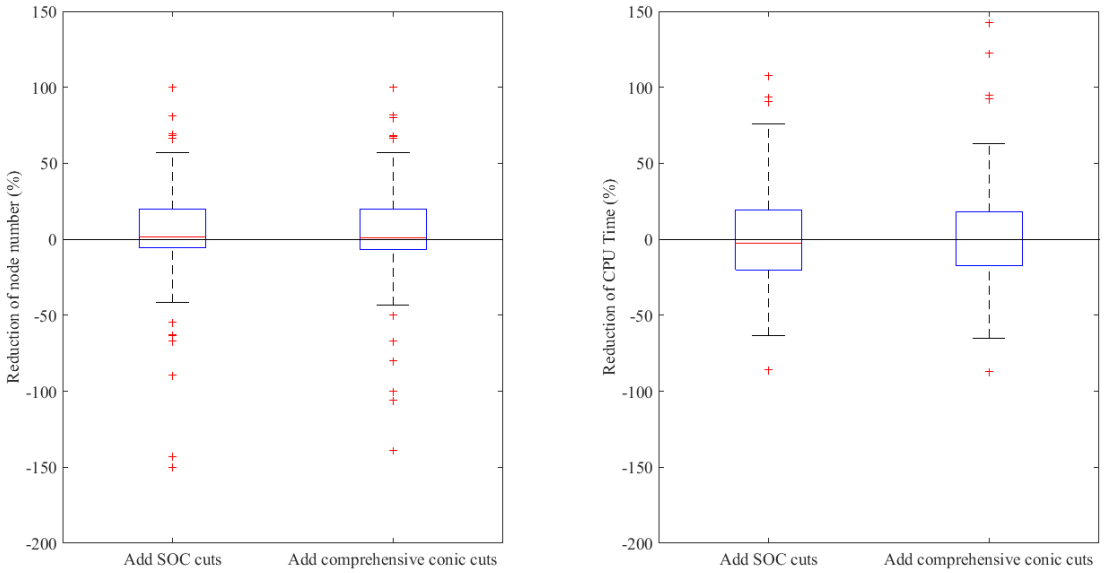
Instances	Results	Original	SOC-cuts	Comp.-cuts
Type-I	# of cuts	0	8	9
(134)	# of nodes	300	281	282
	CPU time	0.50s	(0.31+0.41)s	(1.17+0.44)s
Type-II	# of cuts	0	0	3
(15)	# of nodes	6	6	5
	CPU time	0.15s	(0.07+0.15)s	(0.38+0.14)s
Type-III	# of cuts	0	0	0
(51)	# of nodes	875	875	875
	CPU time	1.13s	(0.20+1.13)s	(0.83+1.13)s

from Table 5.1 is that adding SOC cuts and comprehensive conic cuts could reduce the average number of nodes explored in the searching tree, respectively. In the meanwhile, the cut generation procedure (using our proposed separation heuristic) takes the same order of CPU time as that of solving (l^1-DQP) by Gurobi solver without adding any cut, though the CPU time of solving the resulting branch-and-bound scheme is reduced. To explore the impact of conic cuts

on the branch-and-bound algorithm, we shall focus on the branch-and-bound CPU time rather than the total time including generating conic cuts. In the later numerical experiments, we will see that the branch-and-bound CPU time dominates the cut-generation time in larger-size instances. It is worth improving the separation heuristic such that the extra computational cost of generating cuts may be reduced, yet it is beyond the scope of this study.

Besides the average results, for more detailed information, we draw the box plot of reductions of node number and branch-and-bound CPU time from the 134 type-I instances, respectively, in the next figure. Figure 5.3 shows that by adding either type of conic cut, the number of

Figure 5.3: Impact of adding conic cuts, $n = 40, m = 40, u = 1$



nodes explored in the searching tree and the branch-and-bound CPU time are indeed reduced in almost half of the type-I instances. The performance of comprehensive conic cuts in type-II instances is similar to that in type-I instance, so we do not show the corresponding box plot here.

From the above discussions, we know that for (l^1-DQP) with binary variables, (i) it is very likely to generate conventional SOC cuts and new comprehensive conic cuts at the root node, (ii) the performance of comprehensive conic cuts is compatible to that of SOC cuts for improving the computational efficiency of solving (l^1-DQP), and (iii) comprehensive conic cuts exist in some instances where the conventional SOC cuts cannot be successfully generated.

As the problem size (n and p) increases from 40 to 80, the average branch-and-bound CPU time of solving binary-valued (l^1-DQP) instances with or without adding conic cuts is still around 1 second. Since no other new observation is obtained, we shall move on to the numerical experiments that test (l^1-DQP) instances with general integer-valued variables.

General integer-valued instances

To further investigate and compare the performances of SOC cuts and comprehensive conic cuts for solving general integer-valued (l^1-DQP), we consider the instances of problem size $n, p \in \{40, 80\}$ and $u \in \{5, 10\}$. Recall that in the binary-valued instances ($u = 1$), there are three outcomes with respect to whether or not SOC cuts and comprehensive conic cuts are successfully generated: type I (both), type II (comprehensive conic cuts only) and type III (neither). Table 5.2 lists the percentage of each type appeared in the 200 randomly generated instances of each problem size tested. A direct observation is that SOC cuts have never been generated at the root node, while comprehensive conic cuts exist in about 30% of the instances with $u = 5$ and 13% of the instances with $u = 10$.

Table 5.2: Outcome percentage of the 200 instances

u	n	p	Type-I	Type-II	Type-III
5	40	40	0%	40%	60%
5	40	80	0%	31%	69%
5	80	40	0%	30%	70%
5	80	80	0%	22%	78%
10	40	40	0%	13%	87%
10	40	80	0%	12%	88%
10	80	40	0%	14%	86%
10	80	80	0%	14%	86%

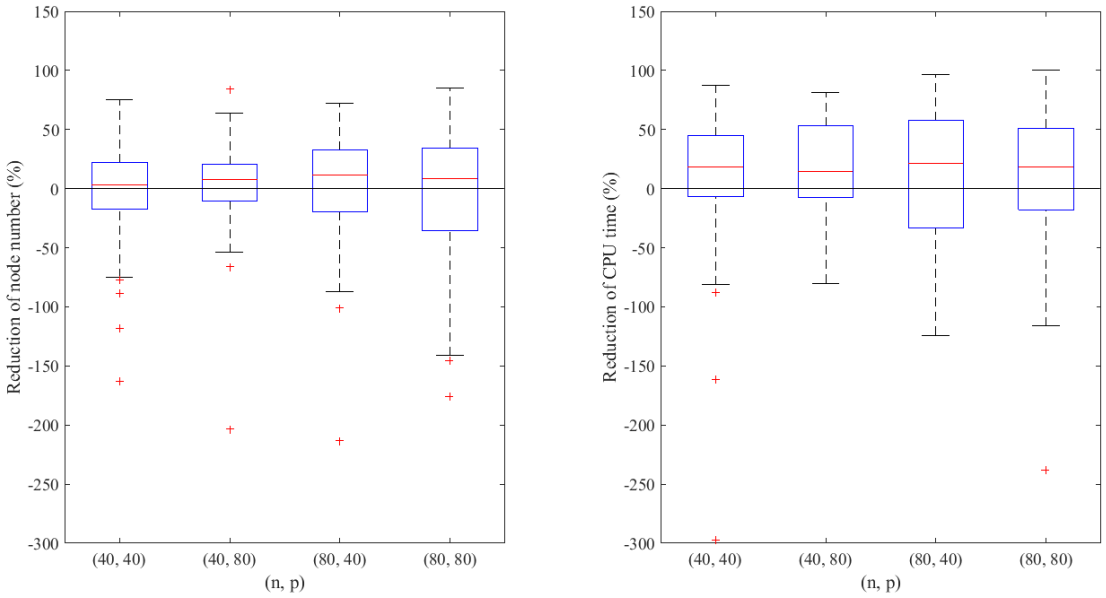
Since the conventional SOC cut generation method does not work here, we continue to explore the performance of proposed comprehensive conic cuts in the remainder of this part. Table 5.3 shows the results in the average level of solving type-II (l^1-DQP) instances with and without adding comprehensive conic cuts, respectively. We find that adding comprehensive conic cuts is capable of reducing the number of nodes explored in the searching tree and further leads to a shorter CPU time of branch-and-bound algorithm. Moreover, as the problem size increases, it takes longer CPU time to solve (l^1-DQP) without adding any cut, while the reduction of CPU time by adding comprehensive cuts is more significant. It is notable that in the general integer-

Table 5.3: Computational results of Type-II instances (average)

u	n	p	Original		Comp.-cuts	
			# of nodes	CPU time	# of nodes	CPU time
5	40	40	40350	53.94s	41154	(1.23+51.78)s
5	40	80	69202	81.40s	68143	(2.93+79.18)s
5	80	40	120486	396.33s	103282	(5.01+304.72)s
5	80	80	418870	1518.85s	322900	(10.75+961.65)s
10	40	40	27410	27.37s	24147	(1.01+22.81)s
10	40	80	97754	192.24s	90914	(2.49+161.89)s
10	80	40	153308	576.43s	118370	(3.73+365.65)s
10	80	80	296968	3238.91s	254169	(9.62+986.29)s

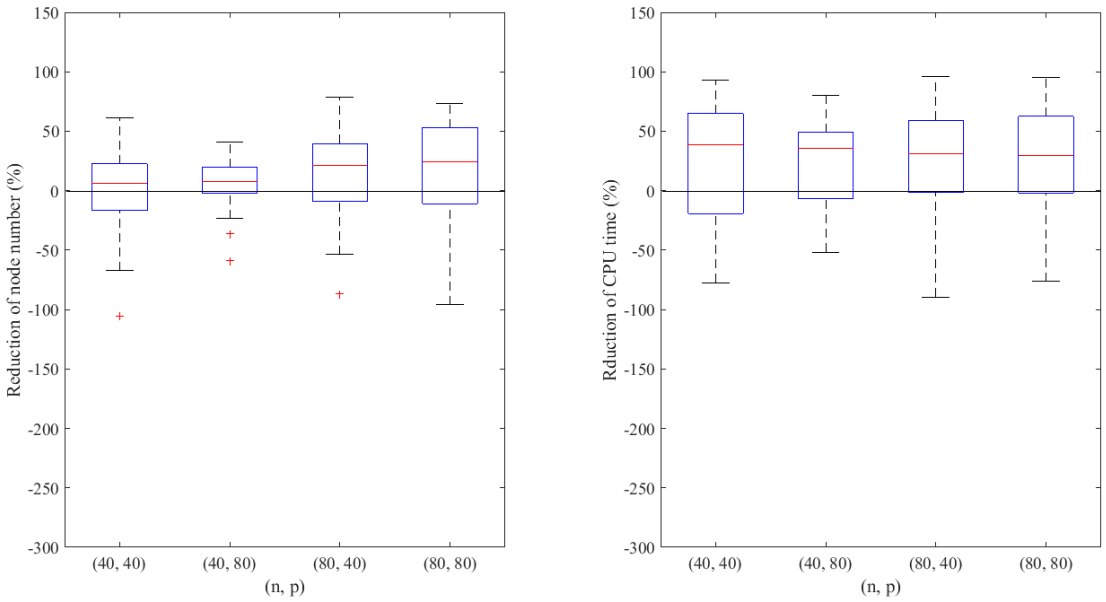
valued instances, the cut-generation computational time actually takes a quite small portion of the total CPU time, especially when the problem size is large.

Figure 5.4: Impact of adding comprehensive conic cuts, $u = 5$



For more specific results, we present the box plot of node number reduction and branch-and-bound CPU time reduction in type-II instances of each problem size with $u = 5$ and $u = 10$, in Figure 5.4 and Figure 5.5, respectively. In the case of $u = 5$, in more than half

Figure 5.5: Impact of adding comprehensive conic cuts, $u = 10$



of the type-II instances, the number of nodes explored decreases and the branch-and-bound algorithm with comprehensive conic cuts added is more computationally efficient. It is notable that the average reduction levels of node number and CPU time are higher than those in the binary-valued instances, though the variations are augmented. When it comes to the instances with $u = 10$, i.e. each variable may take more integer values, the difficulty of solving (l^1-DQP) actually increases. From Figure 5.5, we see that when comprehensive conic cuts are generated at the root node, there is a higher possibility of about 0.75 that adding these cuts could lead to a reduction of branch-and-bound CPU time. And the average reduction level is at 40% comparing with 20% in the case of $u = 5$. The comprehensive conic cuts perform better in improving the computational efficiency of branch-and-bound algorithm in the case of $u = 10$ than $u = 5$.

Note that there still exist type-III instances in which we fail to generate SOC cuts and comprehensive conic cuts. The average computational results in type-III instances are presented in Table 5.4. Comparing the results in Table 5.3 and Table 5.4, we find that type-III instances are much easier to solve than type-II instances, especially when the problem size is large. On the other hand, the computational cost of generating cuts is consistent for these two types of instances. Therefore, implementing the cut-generation procedure has the potential to relieve the computational burden of solving difficult (l^1-DQP) instances, although it may increase the computational time of solving easy instances.

From the above discussions, we know that for general integer-valued (l^1-DQP), conventional

Table 5.4: Computational results of Type-III instances (average)

u	n	p	Original		Comp.-cuts
			# of nodes	CPU time	Cut-generation time
5	40	40	46796	68.96s	1.63s
5	40	80	51243	67.54s	4.37s
5	80	40	31891	42.68s	7.28s
5	80	80	28928	30.18s	15.87s
10	40	40	17418	25.23s	1.79s
10	40	80	37971	56.84s	4.30s
10	80	40	32231	38.86s	6.33s
10	80	80	17501	17.01s	15.61s

SOC inequalities fail to provide conic cuts, yet our proposed comprehensive conic cuts still exist in some instances. Though the possibility of obtaining comprehensive conic cuts may decrease as the problem size u increases, adding the successfully generated cuts is more likely to improve the computational efficiency of the branch-and-bound algorithm for finding an exact solution.

5.5 Summary

In this chapter, we have extended the study to l^1-norm constrained convex quadratic programs with integer variables (l^1-DQP), a special linearly constrained DQP problem. The combination of the l^1-norm feature and discrete nature in quadratic optimization provides new insights to the study of (l^1-DQP). By exploiting the structure of the first-order cone embedded in the l^1-norm constraint, we have developed new ways to generate and manage the conic cuts induced by the integer requirements of the problem. Effective conic cuts can be incorporated into a branch-and-bound scheme for solving (l^1-DQP). Preliminary numerical results have indicated the new conic cuts exist in some instances where conventional SOC cuts fail to be generated. Moreover, adding the new conic cuts is of a high possibility to reduce the complexity of branching trees and improve the overall computational efficiency of the branch-and-bound scheme.

This study has initiated the research of exploring mixed-integer conic cuts that are specific to the structure of the original DQP problem. In order to utilize the new conic cuts in practical applications, several issues need to be further studied in the future. First, theoretical analysis on the effectiveness of using conic cuts is necessary for a better understanding of the new method of generating conic cuts. Second, it may be worth exploring more effective separation heuristics to relieve the computational burden of cut-generation. Finally, examining some explicit conditions under which the new conic cuts can be successfully generated may provide more insights for practical implementations.

Chapter 6

Conclusions

In this chapter, we summarize the research results of this dissertation, highlight its contributions and suggest some future research directions. Section 6.1 first summarizes the results obtained in Chapters 3-5 and then highlights the contributions of this dissertation. Directions for future research are discussed in Section 6.2.

6.1 Summary and Contributions

DQP is an important mathematical optimization problem in both practice and theory. We have seen extensive DQP models in real-life applications in engineering and system sciences. However, the combination of the quadratic feature and discrete nature makes the problem difficult to solve. Since DQP is NP-hard in general, one may obtain approximate solutions of DQP from polynomial-time solvable relaxation problems or find exact solutions to DQP via efficient branch-and-bound algorithms. In this dissertation, we have utilized up-to-date continuous optimization techniques and exploited the discrete structure embedded to study more effective solution methods for DQP.

In Chapter 3, we have proposed an RLT-based linear conic relaxation for finding approximate solutions to DQP. By analyzing the feasible domain of the proposed relaxation problem, we derived a new rank-two condition under which we may explicitly derive optimal solutions to DQP. The new condition is stronger than the known rank-one condition of general linear conic relaxations for binary quadratic programs. Numerical results have shown that the proposed relaxation can provide tight and robust lower bounds for the original DQP problem. The linear conic relaxation approach have been further extended to linearly and quadratically constrained DQP problems. For the linear conic relaxation of a linearly constrained DQP problem, the special rank-two property still holds. This work has shown that conic optimization techniques could provide new insights into developing more efficient approximate solution methods for

DQP.

In Chapter 4, we have developed a Super Logarithmic Method to derive effective 0-1 mixed-integer linear reformulations of DQP for finding exact optimal solutions. The Super Logarithmic Method was first proposed for linearizing a generalized signomial program with discrete variables (DSP). Compared with the state-of-the-art linearization method, the proposed method requires half an order fewer inequality constraints in the reformulation. Numerical results have supported the conclusion that solving the reformulation provided by the Super Logarithmic Method is more effective. Later, we customized the Super Logarithmic Method to treat DQP. By exploring the network structure embedded in DQP, we have further enhanced the reformulation of DQP provided by the Super Logarithmic Method. In this new linearization method, we have exploited the discrete structure of the product of discrete variables and provided a more effective reformulation that may significantly improve the computational efficiency of finding exact solutions.

In Chapter 5, we have extended our study to a special linearly constrained DQP problem, called l^1-norm constrained convex quadratic programming with integer variables (l^1-DQP). We explored an advanced solution method that incorporates mixed-integer cuts into a branch-and-bound scheme for solving l^1-DQP. Exploiting the first-order conic structure embedded in the l^1-norm constraint, we have developed new conic cuts of l^1-DQP that are expected to be more effective than the conventional second-order conic (SOC) cuts. Numerical results have shown the promising performance of proposed conic cuts in improving the computational efficiency of the branch-and-bound scheme for solving l^1-DQP. This work has provided insights for exploiting conic structures of nonlinear discrete optimization problems to generate effective conic cuts in a branch-and-bound scheme.

DQP as a special subclass of nonlinear discrete programs is known to be NP-hard. The research in this dissertation has developed two effective solution methods for finding approximate and exact solutions to DQP, respectively. For a special linearly constrained convex DQP problem, we have further explored a branch-and-bound based solution method that utilizes more comprehensive conic cuts from the embedded conic structure. The results have demonstrated that the advanced linear conic optimization technique has the potential of treating difficult nonlinear discrete programs by exploiting conic structures embedded in the problem. In the meanwhile, we have shown that the discrete nature of DQP could be handled in three different ways: (i) In the linear conic relaxation approach, the discrete requirement of variables is transformed into special structured continuous constraints, which is the key to the special rank-two condition of the proposed relaxation problem. (ii) The new linearization method successfully decomposes the discrete structure of the product of discrete variables for developing more effective reformulations. (iii) The discrete nature can also be combined with conic structures in the problem to derive effective conic cuts for a branch-and-bound scheme. Exploiting the

discrete structure of DQP has allowed us to develop more efficient algorithms for computing approximate and exact solutions.

As we have discussed in Chapter 3, the linear conic relaxation approach of DQP can be extended to approximate DQP with linear and quadratic constraints. Also, the linearization method proposed in Chapter 4 is directly applicable to quadratically constrained DQP problems, in which the key remains to be the linearization of quadratic terms in discrete variables. And the conic cut generation method investigated in Chapter 5 can be naturally extended to (l^1-DQP) with additional convex quadratic constraints. Namely, the DQP solution methods in this dissertation are applicable to treat more general quadratically constrained quadratic programing problems with discrete variables.

6.2 Future Research

For the solution methods of DQP that we have proposed in Chapters 3-5, there are several remaining issues worthy of further studies in the future.

(1.1) The linear conic relaxation approach in Chapter 3 has shown the advantages in generating high-quality lower bounds of DQP, yet the computational efficiency of solving the relaxation problem needs to be further improved. Motivated by the recent progress in solving doubly nonnegative programming (DNN) problems [25, 53], we intend to develop efficient algorithms that are specialized to the structure of linear conic relaxation.

(1.2) Note that the linearization method proposed in Chapter 4 is particularly effective for solving DQP with special network-structured coefficients. It will be meaningful to explore real-life applications that can be modeled as special-structured DQP problems and customize the proposed linearization method to solve the practical DQP problem.

(1.3) We have seen the performance of proposed conic cuts in solving l^1-DQP by a branch-and-bound scheme. Yet theoretical analysis of the conic cuts' effectiveness needs to be further studied. We would like to investigate the properties of proposed conic cuts and extend the conic cut generation method to general l^p-norm constrained DQP problems in the future.

At last, the research in this dissertation shall be further extended to study other related nonlinear discrete optimization problems. The linear conic approach for continuous polynomial programming has arisen in recent literature. To study polynomial discrete optimization problems, we may adopt the linear conic relaxation approach to study approximate solution methods. And the new linearization method can also be applied to generate effective MILP reformulations of polynomial discrete optimization problems for fast computation of exact solutions.

REFERENCES

[1] Adams, W.P., R.J. Forreste and F.W. Glover (2004). Comparisons and enhancement strategies for linearizing mixed 0-1 quadratic programs. *Discrete Optimization*, 1(2), 99-120.

[2] Adams, W.P. and S.M. Henry (2012). Base-2 expansions for linearizing products of functions of discrete variables. *Operations Research*, 60(6), 1477-1490.

[3] Akan, M., O. Alagoz, B. Ata, F.S. Erenay and A. Said (2012). A broader view of designing the liver allocation system. *Operations Research*, 60(4), 757-770.

[4] Anstreicher, K.M. (2009). Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization*, 43(2-3), 471-484.

[5] Arora, J.S. (2004). *Introduction to Optimum Design*, Academic Press.

[6] Arora, J.S., M.W. Huang and C.C. Hsieh (1994). Methods for optimization of nonlinear problems with discrete variables: A review. *Structural Optimization*, 8(2-3), 69-85.

[7] Atamtürk, A. and V. Narayanan (2010). Conic mixed-integer rounding cuts. *Mathematical Programming* 122(1), 1-20.

[8] Belotti, P., J.C. Góez, I. Pólik, T.K. Ralphs and T. Terlaky (2015). A conic representation of the convex hull of disjunctive sets and conic cuts for integer second-order cone optimization. In *Numerical Analysis and Optimization*, Al-Baali, M., Grandinetti, L. and Purnama, A. (eds.), Springer, Switzerland, 1-35.

[9] Bertsimas, D. and J.N. Tsitsiklis (1997). *Introduction to Linear Optimization*, Athena Scientific Belmont, MA.

[10] Benson, H.Y. and U. Saglam (2013). Mixed-integer second-order cone programming: A survey. In *Tutorials in Operations Research: Theory Driven by Influential Applications*, Topaloglu, H., Smith, J.C. and Greenberg, H.J. (eds.), INFORMS, Catonsville, MD, 13-36.

[11] Biegler, L.T. and I.E. Grossmann (2004). Retrospective on optimization. *Computers and Chemical Engineering*, 28(8), 1169-1192.

[12] Billionnet, A., S. Elloumi and A. Lambert (2008). Linear reformulations of integer quadratic programs. In *Modeling, Computation and Optimization in Information Systems and Management Sciences*, H.A. Le Thi, P. Bouvry and P.D. Tao (Eds), Springer Berlin Heidelberg, 43-51.

[13] Billionnet, A., S. Elloumi and A. Lambert (2012). Extending the QCR method to general mixed integer programs. *Mathematical Programming*, 131(1-2), 381-401.

[14] Billionnet, A., S. Elloumi and A. Lambert (2013). An efficient compact quadratic convex reformulation for general integer quadratic programs. *Computational Optimization and Applications*, 54(1), 141-162.

[15] Björk, K., P.O. Lindberg and T. Westerlund (2003). Some convexifications in global optimization of problems containing signomial terms. *Computers and Chemical Engineering*, 27(5), 669-679.

[16] Bomze, I.M. and E. de Klerk (2002). Solving standard quadratic optimization problem via linear, semidefinite and copositive programming. *Journal of Global Optimization*, 24(2), 163-185.

[17] Boyd, S., S.J. Kim, L. Vandenberghe and A. Hassibi (2007). A tutorial on geometric programming. *Optimization and Engineering*, 8(1), 67-127.

[18] Buchheim, C., A. Caprara and A. Lodi (2012). An effective branch-and-bound algorithm for convex quadratic integer programming. *Mathematical Programming*, 135(1-2), 369-395.

[19] Buchheim, C., M. Santis, L. Palagi and M. Piacentini (2013). An exact algorithm for nonconvex quadratic integer minimization using ellipsoidal relaxations. *SIAM Journal on Optimization*, 23(3), 1867-1889.

[20] Buchheim, C. and A. Wiegele (2013). Semidefinite relaxations for non-convex quadratic mixed-integer programming. *Mathematical Programming*, 141(1-2), 435-452.

[21] Burer, S. (2009). On the copositive representation of binary and continuous nonconvex quadratic programs. *Mathematical Programming*, 120(2), 479-495.

[22] Burer, S. and A.N. Letchford (2012). Non-convex mixed-integer nonlinear programming: A survey. *Surveys in Operations Research and Management Science*, 17(2), 97-106.

[23] Callegari, S., F. Bizzarri, R. Rovatti and G. Setti (2010). On the approximate solution of a class of large discrete quadratic programming problems by modulation: The case of circulant quadratic forms. *Signal Processing, IEEE Transactions on*, 58(12), 6126-6139.

[24] Çezik, M.T. and I. Garud (2005). Cuts for mixed 0-1 conic programming. *Mathematical Programming* 104(1), 179-202.

[25] Chen, J. and S. Burer (2012). Globally solving nonconvex quadratic programming problems via completely positive programming. *Mathematical Programming Computation* 4(1), 33-52.

[26] Deng, Z., S.-C. Fang, Q. Jin and C. Lu (2012). Conic approximation to nonconvex quadratic programming with convex quadratic constraints. *Journal of Global Optimization*, 61(3), 459-478.

[27] Duffin, R.J., E.L. Peterson and C. Zener (1967). *Geometric Programming: Theory and Application*, New York: Wiley.

[28] Ernst, A., H. Jiang and M. Krishnamoorthy (2006). Exact solutions to task allocation problems. *Management Science*, 52(10), 1634-1646.

[29] Fang, S.-C. and S. Puthenpura (1993). *Linear Optimization and Extensions: Theory and Algorithms*, Englewood Cliffs, NJ: Prentice-Hall.

[30] Fang, S.-C. and W. Xing (2013). *Linear Conic Optimization*, Science Press, Beijing, China.

[31] Floudas, C.A. and C.E. Gounaris (2009). A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1), 3-38.

[32] Glineur, F., and T. Terlaky (2004). Conic formulation for l^p-norm optimization. *Journal of Optimization theory and Applications*, 122(2), 285-307.

[33] Glover, F. (1975). Surrogate constraint duality in mathematical programming. *Operations Research*, 23(3), 434-451.

[34] Guo, X., Z. Deng, S.-C. Fang, and W. Xing (2014). Quadratic optimization over one first-order cone. *Journal of Industrial and Management Optimization* 10(3), 945-963.

[35] Optimization, G. (2012). Gurobi Optimizer Reference Manual. http://www.gurobi.com.

[36] Hemmecke, R, M. Köppe, J. Lee and R. Weismantel (2009). Nonlinear Integer Programming. In *50 Years of Integer Programming 1958-2008*, M. Jünger, T.M. Liebling, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi and L.A. Wolsey (Eds), Springer, Berlin, 561-618.

[37] Hiller, F.S. and G.J. Liberman (1990). *Introduction to Operations Research*, McGraw-Hill.

[38] Horn, R.A. and C.R. Johnson (1990). *Matrix Analysis*, Cambridge, UK: Cambridge University Press.

[39] CPLEX, I.I. (2009). V12. 1: User's Manual for CPLEX. *International Business Machines Corporation*, 46(53), 157. http://www.ilog.com.

[40] Jeroslow, R.G. (1990). Representability in mixed integer programming, I: Characterization results. *Discrete Applied Mathematics*, 17(3), 223-243.

[41] Kettani, O. and M. Oral (1990). Equivalent formulations of nonlinear integer problems for efficient optimization. *Management Science*, 36(1), 115-119.

[42] Kong, N., A.J. Schaefer, B. Hunsaker and M.S. Roberts (2010). Maximizing the efficiency of the US liver allocation system through region design. *Management Science*, 56(12), 2111-2122.

[43] Lawler, E.L. and D.E. Wood (1966). Branch-and-bound methods: A survey. *Operations Research*, 14(4), 699-719.

[44] Li, H.-L. and C.-T. Chang (1998). An approximate approach of global optimization for polynomial programming problems. *European Journal of Operational Research*, 107(3), 625-632.

[45] Li, H.-L., S.-C. Fang, Y.-H. Huang and T. Nie (2016). An enhanced logarithmic method for signomial programming with discrete variables. *European Journal on Operational Research* 255(3), 922-934.

[46] Li, H.-L., Y.-H. Huang and S.-C. Fang (2013). A logarithmic method for reducing binary variables and inequality constraints in solving task assignment problems. *INFORMS Journal on Computing*, 25(4), 643-653.

[47] Li, H.-L., Y.-H. Huang and S.-C. Fang (2016). Linear reformulation of polynomial discrete programming for fast computation. To appear in *INFORMS Journal on Computing*.

[48] Li, H.-L., H.-C. Lu, C.-H. Huang and N.-Z. Hu (2009). A superior representation method for piecewise linear functions. *INFORMS Journal on Computing*, 21(2), 314-321.

[49] Li, H.-L. and H.-C. Lu (2010). Global optimization for generalized geometric programs with mixed free-sign variables. *Operations Research*, 57(3), 701-713.

[50] Lin, C.-Y. and P. Hajela (1992). Genetic algorithms in optimization problems with discrete and integer design variables. *Engineering Optimization*, 19(4), 309-327.

[51] Lin, M.-H., J.-F. Tsai and C.-S. Yu (2012). A review of deterministic optimization methods in engineering and management. *Mathematical Problems in Engineering*, 2012, Article ID 756023, 15 pages. http://dx.doi.org/10.1155/2012/756023.

[52] Lu, C., S.-C. Fang, Q. Jin, Z. Wang and W. Xing (2010). KKT solution and conic relaxation for solving quadratically constrained quadratic programming problems. *SIAM Journal on Optimization*, 21(4), 1475-1490.

[53] Lu, C., W. Xing and S.-C. Fang (2015). Primal alternating direction method for doubly nonnegative programming with applications to solving mixed integer quadratic programs. Working paper.

[54] Luo, Z.-Q., J.F. Sturm and S. Zhang (2004). Multivariate nonnegative quadratic mappings. *SIAM Journal on Optimization*, 14(4), 1140-1162.

[55] Luo, Z.-Q., W.K. Ma, A. So, Y. Ye, and S. Zhang (2010). Semidefinite relaxation of quadratic optimization problems. *IEEE Signal Processing Magazine*, 27(3), 20-34.

[56] Malik, U., I.M. Jaimoukha, G.D. Halikias and S.K. Gungah (2006). On the gap between the quadratic integer programming problem and its semidefinite relaxation. *Mathematical Programming*, 107(3), 505-515.

[57] Micciancio, D and S. Goldwasser (2002). *Complexity of Lattice Problems: A cryptographic Perspective*, Springer, Berlin.

[58] Murty, K.G. and S.N. Kabadi (1987). Some NP-Complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2), 117-129.

[59] Nesterov, Y. E. and A.S. Nemirovskii (1994). *Interior-point Polynomial Algorithms in Convex Programming*, Philadelphia, PA: Society for Industrial and Applied Mathematics.

[60] Nie, T., S.-C. Fang, Z. Deng, and J.E. Lavery (2016). On linear conic relaxation of discrete quadratic programs. *Optimization Methods and Software* 31(4), 1-18.

[61] Nie, T., S.-C. Fang, Q. An and Zhibin Deng (2016). Mixed-integer conic cuts of l^1-norm constrained convex quadratic integer programming. Working paper.

[62] Pardalos, P.M. and S.A. Vavasis (1991). Quadratic programming with one negative eigenvalue is NP-Hard. *Journal of Global Optimization*, 1(1), 15-22.

[63] Phuong, N.T.H., H. Tuy and F. Al-Khayyal (2006). Optimization of a quadratic function with a circulant matrix. *Computational Optimization and Applications*, 35(2), 135-159.

[64] Quan, J., Z. Wu and G. Li (2011). Global optimality conditions for some classes of polynomial integer programming problems. *Journal of Industrial and Management Optimization*, 7(1), 67.

[65] Rendl, F., R. Giovanni and W. Angelika (2010). Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Mathematical Programming*, 121(2), 307-335.

[66] Rendl, F. and H. Wolkowicz (1997). A semidefinite framework for trust region subproblems with applications to large scale minimization. *Mathematical Programming*, 77(1), 273-299.

[67] Rockafellar, R.T. (1972). *Convex Analysis (2nd Edition)*, Princeton, NJ: Princeton University Press.

[68] Sahinidis, N.V. (1996). BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2), 201-205.

[69] Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design*, 112(2), 223-229.

[70] Schrijver, A. (1998). *Theory of Linear and Integer Programming*, Jonh Wiley & Sons.

[71] Shen, P.P. and X.D. Bai (2013). Global Optimization for generalized geometric programming problems with discrete variables. *Optimization*, 62(7), 895-917.

[72] Sherali, H.D. (2007). RLT: A unified approach for discrete and continuous nonconvex optimization. *Annals of Operations Research*, 149(1), 185-193.

[73] Sherali, H.D. and W.P. Adams (1990). A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3), 411-430.

[74] Sherali, H.D. and W.P. Adams (1994). A hierarchy of relaxations and convex hull characterizations for mixed-integer zero-one programming problems. *Discrete Applied Mathematics*, 52(1), 83-106.

[75] Sherali, H.D. and W.P. Adams (1999). *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, Kluwer Academic Publishers, Dordrecht, Boston, London.

[76] Sherali, H.D. and A.R. Alameddine (1992). A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2(4), 379-410.

[77] Sherali, H.D. and J.C. Smith (2007). An improved linearization strategy for zero-one quadratic programming problems. *Optimization Letters*, 1(1), 35-47.

[78] Sherali, H.D. and C.H. Tuncbilek (1992). A global optimization algorithm for polynomial programming problems using a reformulation-linearization technique. *Journal of Global Optimization*, 2(1), 101-112.

[79] Sherali, H.D. and C.H. Tuncbilek (1995). A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Journal of Global Optimization*, 7(1), 1-31.

[80] Sherali, H.D. and H. Wang (2001). Global optimization of nonconvex factorable programming problems. *Mathematical Programming*, 89(3), 459-478.

[81] Shor, N.Z. (1987). Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25(6), 1-11.

[82] Sturm, J.F. (1999). Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11(1-4), 625-653. Version 1.3 available at http://sedumi.ie.lehigh.edu/.

[83] Sturm, J.F. and S. Zhang (2003). On cones of nonnegative quadratic functions. *Mathematics of Operations Research*, 28(2), 246-267.

[84] Sun, X., C. Liu, D. Li and J. Gao (2012). On duality gap in binary quadratic programming. *Journal of Global Optimization*, 53(2), 255-269.

[85] Terlaky, T. (1985). On l^p programming. *European Journal of Operational Research*, 22(1), 70-100.

[86] Tawarmalani, M., K. Kannan and P. De (2009). Allocating objects in a network of caches: Centralized and decentralized analyses. *Management Science*, 55(1), 132-147.

[87] Tsai, J.-F. and M.-H. Lin (2006). An optimization approach for solving signomial discrete programming problems with free variables. *Computers and Chemical Engineering*, 30(8), 1256-1263.

[88] Tsai, J.-F. and M.-H. Lin (2011). Finding multiple optimal solutions of signomial discrete programming problems with free variables. *Optimization Engineering*, 12(3), 425-443.

[89] Vielma, J.P., S. Ahmed and G. Nemhauser (2010). Mixed-integer models for nonseparable piecewise linear optimization: Unifying framework and extensions. *Operations Research*, 58(2), 303-315.

[90] Vielma, J.P., Ahmed S. and G. Nemhauser (2010). A note on "A superior representation method for piecewise linear functions". *INFORMS Journal on Computing*, 22(3), 493-497.

[91] Wang, Z., S.-C. Fang., D.Y. Gao, and W. Xing (2008). Global extremal conditions for multi-integer quadratic programming. *Journal of Industrial and Management Optimization*, 4(2), 213.

[92] Wu, S., L.M. Hitt, P. Chen and G. Anandalingam (2008). Customized bundle pricing for information goods: A nonlinear mixed-integer programming approach. *Management Science*, 54(3), 608-622.

[93] Zheng, X., X. Sun and D. Li (2011) Nonconvex quadratically constrained quadratic programming: best D.C. decomposition and their SDP representations. *Journal of Global Optimization*, 50(4), 695-712.