New Conic Optimization Techniques for Solving Binary Polynomial Programming Problems

by

Bissan Ghaddar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Management Sciences

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Polynomial programming, a class of non-linear programming where the objective and the constraints are multivariate polynomials, has attracted the attention of many researchers in the past decade. Polynomial programming is a powerful modeling tool that captures various optimization models. Due to the wide range of applications, a research topic of high interest is the development of computationally efficient algorithms for solving polynomial programs. Even though some solution methodologies are already available and have been studied in the literature, these approaches are often either problem specific or are inapplicable for large-scale polynomial programs. Most of the available methods are based on using hierarchies of convex relaxations to solve polynomial programs; these schemes grow exponentially in size becoming rapidly computationally expensive. The present work proposes methods and implementations that are capable of solving polynomial programs of large sizes. First we propose a general framework to construct conic relaxations for binary polynomial programs, this framework allows us to re-derive previous relaxation schemes and provide new ones. In particular, three new relaxations for binary quadratic polynomial programs are presented. The first two relaxations, based on second-order cone and semidefinite programming, represent a significant improvement over previous practical relaxations for several classes of non-convex binary quadratic polynomial problems. The third relaxation is based purely on second-order cone programming, it outperforms the semidefinite-based relaxations that are proposed in the literature in terms of computational efficiency while being comparable in terms of bounds. To strengthen the relaxations further, a dynamic inequality generation scheme to generate valid polynomial inequalities for general polynomial programs is presented. When used iteratively, this scheme improves the bounds without incurring an exponential growth in the size of the relaxation. The scheme can be used on any initial relaxation of the polynomial program whether it is second-order cone based or semidefinite based relaxations. The proposed scheme is specialized for binary polynomial programs and is in principle scalable to large general combinatorial optimization problems. In the case of binary polynomial programs, the proposed scheme converges to the global optimal solution under mild assumptions on the initial approximation of the binary polynomial program. Finally, for binary polynomial programs the proposed relaxations are integrated with the dynamic scheme in a branch-and-bound algorithm to find global optimal solutions.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Professor Miguel Anjos and Professor Juan Vera. Thank you for your support and advice throughout my graduate studies.

I would like to thank the members of my examination committee: Professor Etienne de Klerk, Professor Stephen Vavasis, Professor Samir Elhedhli, and Professor David Fuller. I am very grateful for their comments and helpful suggestions on my thesis. Having the opportunity to visit Professor Michael Jünger and PD Dr. Frauke Liers at the University of Köln in Germany for three months in 2010 was a truly amazing experience. I would like to sincerely thank Frauke and Angelika for their collaboration on solving partition problems. I am also grateful to the Department of Management Sciences for assisting me in many different ways. The financial support of the Ontario Graduate Scholarship, the NSERC Alexander Graham Bell Canada Graduate Scholarship, and the NSERC Canada Graduate Scholarship - Michael Smith Foreign Study Supplement were very appreciated.

Thanks to my friends and colleagues at Waterloo for all the great times we shared and for providing the friendly environment which I had the pleasure to work in. I would like also to thank Loulou, Huss, Sam, and Mac for the great friendship, all the emotional support, entertainment, and caring they provided.

I want to thank my beloved Joe Naoum-Sawaya. The love, joy, courage, and motivation you bring me is beyond description. My time in Waterloo and everywhere else was magical with you by my side. This thesis couldn't become a reality without your support.

Lastly and most importantly, I am indebted to my parents Taha and Fatima Ghaddar and my two wonderful sisters Ninar and Ruha for providing me with a loving and supportive environment and for encouraging me all the way. I can not thank you enough. It is to you that I dedicate this thesis.

Table of Contents

List of Tables

List of Figures

Notation

$\mathbb{R}^n_+$	the non-negative orthant cone
$\mathcal{L}^n$	the second-order cone
$\mathcal{S}^n_+$	the semidefinite cone
$\deg(\cdot)$	the degree of a polynomial
$\mathcal{M}_d(x)$	the set of monomials of x of degree at most d
$\mathbf{R}[x]$	the set of polynomials in n variables with real coefficients
$\mathbf{R}_d[x]$	the set of polynomials in n variables with real coefficients of degree at most d
$\Psi[x]$	the cone of real polynomials that are sum-of-squares
$\Psi_d[x]$	the cone of real polynomials of degree at most d that are sum-of-squares
$\mathcal{B}$	$\{x \in \mathbb{R}^n : \|x\|^2 = n\}$, the sphere of radius $\sqrt{n}$
$\mathcal{P}_d(S)$	the cone of polynomials of degree at most d that are non-negative over S
LP	linear program
SOC	second order cone program
SDP	positive semidefinite program
SOS	sum-of-squares
PP	polynomial program
BPP	binary polynomial program
BQPP	binary quadratic polynomial program

Chapter 1

Introduction

A polynomial programming problem has the form:

$$\begin{aligned} \textbf{(PP-P)}\ z_{PP} = \ & \max\ f(x) \\ & \text{s.t. } g_i(x) \geq 0 \quad 1 \leq i \leq m, \end{aligned}$$

where $f(x)$ and $g_i(x)$ are real-valued multivariate polynomials for $1 \leq i \leq m$. Equality constraints of the form $h_j(x) = 0$ can be included as they can be expressed as the inequality constraints $h_j(x) \geq 0$ and $h_j(x) \leq 0$.

Solving polynomial programming problems is an area being actively studied. The importance of polynomial programming (PP) is that it captures an important class of non-linear programming problems and has a wide range of practical applications in the context of control, process engineering, facility location, economics and equilibrium, and finance. PP generalizes several special cases that have been thoroughly studied in optimization, including mixed binary linear programming, convex/non-convex quadratic programming, and complementarity programming. It is well known that solving polynomial programs is an $\mathcal{NP}$-hard problem.

Several authors have proposed methods for constructing semidefinite relaxations of (PP-P), based on results about moment sequences [53] and representations of nonnegative polynomials as sum-of-squares (SOS). The key ingredient is to re-cast the feasibility of a finite

system of polynomial equalities and inequalities in terms of an alternative polynomial involving squares of (unknown) polynomials. Shor introduced the idea that given an unconstrained polynomial problem, one can compute the minimum value λ such that $\lambda - f(x)$ is a sum-of-squares to obtain an upper bound on z (the global optimum of f) [91]. λ can be computed in polynomial time using semidefinite programming (SDP) [96]. This idea was further developed by Parrilo [75] and Parrilo and Sturmfels [78] to the non-restricted case using sum-of-squares decomposition. Lasserre [53] proposed a general solution approach for polynomial optimization problems via semidefinite programming using methods based on moment theory.

Since the seminal work of Lasserre and Parrilo, intense research activity has been carried out on solving polynomial programming using sum-of-squares representations and the related theory of moments. This research includes the recent work of de Klerk and Pasechnik [20] where they approximated the copositive cone via a hierarchy of linear or semidefinite programs of increasing size using the idea of sum-of-squares and polynomials with non-negative coefficients. Lasserre [53, 54] presented a hierarchy of semidefinite programming approximations to polynomial programs which converges under mild assumptions to the optimal solution of the original polynomial program. Laurent [57] provided a comparison of the Sherali-Adams, Lovász-Schrijver, and Lasserre relaxations for binary programs. Laurent [58] also used real algebraic geometric tools to find solution schemes for polynomial programs on finite varieties. Nie, Demmel, and Sturmfels [74] proposed a method for finding the global minimum of a multivariate polynomial via sum-of-squares relaxation over its gradient variety. Nie [72] also presented regularization type methods which would solve significantly larger problems than interior-point type methods used to solve sum-of-squares and Lasserre's relaxations. Parrilo [76, 77] combined ideas from algebraic geometry and convex optimization to solve a wide range of computational problems appearing in real algebraic geometry. Peña, Vera, and Zuluaga [80, 104] presented solution schemes exploiting the equality constraints. In addition, the idea of approximating a set of non-negative polynomials is also present in the work of several authors such as the early work of Nesterov [70], Shor [91], and the $\mathcal{S}$-Lemma of Yakubovich (see [83]) among others. Specifically for binary optimization, the specialization of Lasserre's construction to binary PPs (BPP) was shown to converge in a finite number of steps in [54] and the relationship of the Lasserre hierarchy to other well-known hierarchies was studied in works such as [55, 57].

The main idea for solving polynomial programs is the application of representation theorems to characterize the set of polynomials that are non-negative on a given domain. Most PP approaches can be seen as the construction of SOS certificates of non-negativity for a suitable representation of the PP problem. Using the complexity (degree) of the certificates as a parameter, produces a hierarchy of approximations. While the resulting hierarchies yield very tight approximations of the original PP problem, from a computational perspective, these hierarchies all suffer from a common limitation, namely the explosion in size of the semidefinite relaxations involved. One way to overcome this difficulty is to exploit the structure of the problem. This can be done either by taking advantage of symmetry as in Bai, de Klerk, Pasechnik, and Sotirov [5], de Klerk [16], de Klerk, Pasechnik, and Schrijver [19], de Klerk and Sotirov [21], and Gatermann and Parrilo [29], or by exploiting sparsity to reduce the size of the semidefinite relaxation as in Kojima, Kim, and Waki [48], Waki, Kim, Kojima, and Muramatsu [98, 99], and several others [45, 46, 47, 49, 73]. However, in the absence of any favorable structure, the practical application of the SOS approach is severely limited. The motivation for this research is to devise new techniques to take advantage of the strength of the PP approach without relying on the presence of exploitable structure in the problem or suffering from the computational burden resulting from the exponential growth in the hierarchy. To achieve this objective, it is imperative to avoid the growth of the complexity of the non-negativity certificates involved.

In this thesis we develop algorithms to solve PPs in general and BPPs in particular. For this purpose we put together results about new relaxations of PPs and a dynamic inequality generation scheme which are then applied in the context of a branch-and-bound approach.

The first contribution of this thesis, presented in Chapter 3, is to use a characterization of non-negative linear polynomials over the ball to propose second-order cone (SOC) relaxations for binary polynomial programs. We use the polynomial programming framework to present a new second-order and semidefinite-based construction where we are able to theoretically show that the resulting relaxations provide bounds stronger than other computationally practical semidefinite-based relaxations proposed in the literature for binary quadratic polynomial programs. Additionally, our proposed framework enables us to isolate expensive components of existing relaxations, namely the semidefinite terms. By removing the semidefinite terms, we obtain relaxations based purely on second-order

cones. The computational experiments confirm our theoretical results, we obtain that the SOC-SDP-based relaxations give the best bounds. Our experiments also show that the purely SOC-based relaxations produce bounds that are competitive with the existing SDP bounds but computationally much more efficient.

The second contribution of this thesis, presented in Chapter 4, is to improve PP relaxations without incurring exponential growth in the size of the relaxation. The key ingredient is a dynamic generation scheme for general polynomial programs where instead of growing the degree of the certificates (which results in an exponential growth of the size of the relaxation), we fix the degree of the polynomials that we use and increase the set of polynomial inequalities describing the feasible region of the PP problem. These valid inequalities are then used to construct new certificates that provide better approximations. We obtain valid inequalities by means of a dynamic inequality generation scheme (DIGS) that makes use of information from the objective function to dynamically generate polynomial inequalities that are valid on the feasible region of the PP problem. The result is an iterative scheme that provides a sequence of improving approximations without growing the degree of the certificates involved. Depending on the original problem and the type of relaxation used, DIGS solves a sequence of linear, second-order cone, or semidefinite problems. Convergence to the global optimal solution is proven for a family of problems including binary quadratic programming with linear constraints. The computational results highlight the advantages of DIGS with respect to Lasserre's approach [54] as well as to the lift-and-project method of Balas, Ceria, and Cornuéjols [6].

The third contribution, presented in Chapter 5, is to develop a branch-and-dig algorithm, that is, a branch-and-bound algorithm that combines the proposed relaxations with DIGS to provide globally optimal solutions for binary polynomial programming problems. Finally, in Chapter 6 we present a computational tool that can be applied to solve a broad class of polynomial programs. The potential impact of the methodology presented in this thesis is significant, since it provides a means to tightly approximate PPs that, unlike previously proposed hierarchies of SDP relaxations, is in principle scalable to large general combinatorial optimization problems.

Chapter 2

Background

We present conic programming formulations reviewing some important properties, and special cases of conic programming. The importance of conic programming is that it allows to reveal rich structure which usually is possessed by a convex program and to exploit this structure in order to solve the program efficiently.

In Section 2.1, we focus primarily, on linear, second-order and semidefinite programming. Then in Section 2.2, we present polynomial programming problems. We also present Lasserre's hierarchy of improving relaxations for PP using the SOS certificates approach.

2.1 Conic Programming

Most of the definitions in this section have been taken from [59]. A set C is convex if

$$y_1,\ y_2 \in C \ \Rightarrow\ \lambda y_1 + (1-\lambda) y_2 \in C \,\forall\, 0 \leq \lambda \leq 1.$$

The dual of a set C is denoted by C^* and is defined by

$$C^* = \{a \in \mathbb{R}^n : y \in C,\ \langle a, y\rangle \geq 0\}.$$

A set $\mathcal{C} \subseteq \mathbb{R}^n$ is a said to be a cone if

$$\lambda \geq 0,\ y \in \mathcal{C} \Rightarrow \lambda y \in \mathcal{C}.$$

A cone $\mathcal{C}$ is pointed if $\mathcal{C} \cap (-\mathcal{C}) = \{0\}$, and solid if the interior of $\mathcal{C}$ is not empty. The dual cone of $\mathcal{C}$, $\mathcal{C}^*$, is always a convex cone and in case $\mathcal{C}^{**} = \mathcal{C}$ then $\mathcal{C}$ is said to be self-dual. A cone is said to be homogeneous if for any two points x, y in the interior of $\mathcal{C}$ there exist a linear transformation F such that $F(x) = y$ and $F(\mathcal{C}) = \mathcal{C}$. A cone is symmetric if it is self-dual and homogeneous.

Conic optimization unifies a wide variety of optimization problems. A general conic optimization problem is of the form:

$$\begin{aligned} \textbf{(CP-P)} \quad & \min\ c^T x \\ & \text{s.t. } Ax = b \\ & \quad x \in \mathcal{C} \end{aligned} \tag{2.1}$$

where $\mathcal{C} \subseteq \mathbb{R}^n$ is a closed convex pointed cone, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$. Hence, we optimize a linear function subject to a system of linear equations, and the condition of the decision variable x lying in $\mathcal{C}$.

The dual problem can be derived by introducing a Lagrangian multiplier $y \in \mathbb{R}^m$ for constraint (2.1). Define the Lagrangian function $L(x, y) = c^T x + y^T(b - Ax)$. Then,

$$\begin{aligned} \min_{x \in \mathcal{C}} \{c^T x : Ax = b\} &= \min_{x \in \mathcal{C}} \max_{y \in \mathbb{R}^m} c^T x + y^T(b - Ax) \\ &\geq \max_{y \in \mathbb{R}^m} \min_{x \in \mathcal{C}} c^T x + y^T(b - Ax) \\ &= \max_{y \in \mathbb{R}^m} \left(b^T y + \min_{x \in \mathcal{C}} (c - A^T y)^T x \right) \\ &= \max_{y \in \mathbb{R}^m} \{b^T y : c - A^T y \in \mathcal{C}^*\} \end{aligned}$$

where $\mathcal{C}^*$ is the dual cone of $\mathcal{C}$ and is defined by

$$\mathcal{C}^* = \{u \in \mathbb{R}^n : u^T x \geq 0, \forall x \in \mathcal{C}\}.$$

Hence, we define the dual of (CP-P) as follows:

$$\begin{aligned} \textbf{(CP-D)} \quad & \max\ b^T y \\ & \text{s.t. } c - A^T y \in \mathcal{C}^* \end{aligned} \tag{2.2}$$

$x \in \mathcal{C}$ is said to be feasible for (CP-P) if $Ax = b$. Similarly, $y \in \mathbb{R}^m$ is said to be feasible for (CP-D) if $c - A^T y \in \mathcal{C}^*$. If x is a primal feasible solution and y is a dual feasible solution, then their duality gap is defined as the difference between the objective values of (CP-P) and (CP-D):

$$c^T x - b^T y.$$

By construction, (CP-P) and (CP-D) satisfy weak duality:

Theorem 2.1.1. ***Conic-Weak Duality Theorem:*** *Let $x \in \mathcal{C}$ such that $Ax = b$, and $y \in \mathbb{R}^m$ such that $A^T y - c \in \mathcal{C}^*$, then the duality gap satisfies*

$$c^T x - b^T y \geq 0.$$

When $c^T x - b^T y = 0$, we say that strong duality holds. In the case where (CP-P) satisfies the Slater's constraint qualification, i.e., there exists an x in the interior of $\mathcal{C}$ such that $Ax = b$, we have strong duality. The dual problem (CP-D) satisfies Slater constraint qualification if there exists a y such that with $c - A^T y = z$ is in the interior of $\mathcal{C}^*$.

Theorem 2.1.2. *[23] Let $p^* = \inf\{c^T x : Ax = b,\ x \in \mathcal{C}\}$ and $d^* = \sup\{b^T y : z = c - A^T y,\ z \in \mathcal{C}^*\}$, then we have the following:*

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

For the conic relaxations of the polynomial programs that we consider, we have in general that both the primal and the dual problems satisfy the Slater constraint qualification and hence strong duality holds and both optima are attained.

Examples of pointed convex cones:

- the non-negative orthant cone, $\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geq 0\}$.
- the second-order cone, $\mathcal{L}^n = \{x \in \mathbb{R}^{n+1} : x_0 \geq \sqrt{\sum_{i=1}^n x_i^2}\},\ x_0 \geq 0$.
- the semidefinite cone, $\mathcal{S}^n_+ = \{X \in \mathbb{R}^{n\times n} : X^T = X, a^T X a \geq 0\, \forall a \in \mathbb{R}^n\}$.
- the Cartesian product of the non-negative orthant, second-order cones, and positive semidefinite cones.

Each of these cones is a closed convex pointed cone with nonempty interior. Moreover, each of these cones is self-dual, i.e., $\mathcal{C}^* = \mathcal{C}$. When considering these cones, the general conic programming formulation (CP-P) reduces to linear programming, second-order cone programming, and semidefinite programming formulations. In these particular cases, conic programming problems can be solved efficiently by interior-point methods. In the last decade, several efficient algorithms and software packages were developed to solve linear, second-order cone, and semidefinite optimization problems see e.g., Vandenberghe and Boyd [96], Lewis and Overton [62], Lobo, Vandenberghe, Boyd, and Lebret [63], Ben Tal and Nemirovski [8], Alizadeh and Goldfarb [2], Wolkowicz, Saigal, and Vandenberghe [100], and Todd [95].

Other examples of convex cones that we are interested in are:

- $\Psi_d[x] = \left\{p(x) \in \mathbf{R}_d[x] : p(x) = \sum_{i=1}^N q_i^2(x),\ q_i(x) \in \mathbf{R}_{\lfloor \frac{d}{2} \rfloor}[x]\right\}$ where $N = \binom{n+d}{d}$. Notice that in particular $\Psi_d[x] = \Psi_{d-1}[x]$ for every odd degree d.
- $\mathcal{P}_d(S) = \{p(x) \in \mathbf{R}_d[x] : p(x) \geq 0\, \forall x \in S\}$, for any $S \subseteq \mathbb{R}^n$.
- $\mathcal{P}^+_d(S) = \{p(x) \in \mathbf{R}_d[x] : p(x) > 0\, \forall x \in S\}$, for any $S \subseteq \mathbb{R}^n$.

$\Psi_d[x] \cong \mathcal{S}^N_+$ so we know how to optimize over it. On the other hand, the decision problem for $\mathcal{P}_d(S)$ and $\mathcal{P}^+_d(S)$ is $\mathcal{NP}$-hard. These cones are convex but not self-dual and thus not symmetric. It is unknown how to efficiently optimize over these cones.

2.1.1 Linear Programming

Linear programming (LP) aims to optimize a linear objective function subject to linear equality and inequality constraints. LP is equivalent to the particular case of conic programming (CP-P) where $\mathcal{C} = \mathbb{R}^n_+$. LP can be formulated as:

$$\begin{aligned} \textbf{(LP-P)} \ \min \ & c^T x \\ \text{s.t.} \ & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.3}$$

where $x \in \mathbb{R}^n$ is a vector of variables $(x_1, \cdots, x_n)$ and $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$ are given. And $x \geq 0$ is a notation for $x \in \mathbb{R}^n_+$, i.e., that all coordinates, $x_1, \cdots, x_n$, of the vector x are non-negative.

We refer to (LP-P) as the primal problem. Since, $(\mathbb{R}^n_+)^* = \mathbb{R}^n_+$, the dual problem, (LP-D), can be expressed as follows:

$$\begin{aligned} \textbf{(LP-D)} \ \max \ & b^T y \\ \text{s.t.} \ & c - A^T y \geq 0 \end{aligned} \tag{2.4}$$

Using Theorem 2.1.1, if x is feasible to (LP-P) and y is feasible to (LP-D), weak duality holds:

$$c^T x - b^T y \geq 0.$$

For LP, strong duality is stronger than Theorem 2.1.2.

Proposition 2.1.3. *If either the primal or the dual is feasible, then the optima of the primal and the dual are attained and are equal to each other. In particular, using strong duality if x^* is optimal to (LP-P) and y^* is optimal to (LP-D), then we obtain*

$$c^T x^* = b^T y^*.$$

2.1.2 Second-Order Cone Programming

Second-order cone programming (SOC) is a convex optimization problem having linear constraints and second-order cone constraints. An SOC primal problem is of the form [63]:

$$\begin{aligned} \textbf{(SOC-P)} \quad & \min\ c^T x \\ & \text{s.t. } Ax = b \\ & x \in \mathcal{L}^n, \end{aligned} \tag{2.5}$$

where the problem parameters are $c \in \mathbb{R}^{n+1}$, $A \in \mathbb{R}^{m\times(n+1)}$, and $b \in \mathbb{R}^m$. Since, $(\mathcal{L}^n)^* = \mathcal{L}^n$, the dual problem, (SOC-D), can be expressed as:

$$\begin{aligned} \textbf{(SOC-D)} \quad & \max\ b^T y \\ & \text{s.t. } c - A^T y \in \mathcal{L}^n. \end{aligned} \tag{2.6}$$

SOC is stronger than LP in the sense that every LP problem can be expressed as an SOC problem. Several efficient primal-dual interior-point methods for SOC have been developed in the last decade [51].

2.1.3 Semidefinite Programming

Semidefinite programming (SDP) is now well recognized as a powerful tool for combinatorial optimization. Early research in this vein has yielded improved approximation algorithms and very tight bounds for some hard combinatorial optimization problems, see [59] and the references therein. As a consequence, interest in the application of semidefinite techniques to combinatorial optimization problems has continued unabated since the mid-1990s. This has included not only theoretical approximation guarantees but also the development and implementation of algorithms for efficiently solving semidefinite relaxations of combinatorial optimization problems. Noteworthy developments in this direction include the biqmac solver for max-cut [88], an SDP-based branch-and-bound solver for max-k-cut [31], extremely tight bounds for the quadratic assignment problem [89], and exact solutions for single-row layout [4] as well as related quadratic linear ordering problems

[41]. Grötschel, Lovász, and Schrijver [34] proved that a semidefinite optimization problem can be solved in polynomial time, hence making semidefinite relaxations an attractive tool for deriving bounds for combinatorial optimization problems.

Semidefinite programming is an extension of linear programming in which the variable is a symmetric matrix which is required to be positive semidefinite. SDP optimizes (minimizes or maximizes) a linear function of a symmetric matrix variable X subject to linear constraints on X and X being positive semidefinite. The set of positive semidefinite matrices is a closed convex cone, but it is not polyhedral. There are several equivalent conditions for a matrix to be positive semidefinite (PSD):

- The matrix $X \in \mathcal{S}^n$ is positive semidefinite $X \succeq 0$.
- $a^T X a \geq 0 \, \forall a \in \mathbb{R}^n$.
- All eigenvalues of X are non-negative.
- All $2^n - 1$ principal minors of X are non-negative.
- There exists a factorization of X of the form $X = V^T V$.

We consider SDP problems in standard form:

$$\begin{aligned} \textbf{(SDP-P)} \ \min \ & \langle C, X \rangle \\ \text{s.t.} \ & \langle A_i, X \rangle = b_i \qquad 1 \leq i \leq m \\ & X \succeq 0 \end{aligned} \tag{2.7}$$

where C and A_i are matrices in $\mathcal{S}^n$ and $b_i \in \mathbb{R}$. Since $\mathcal{S}^n_+$ is self dual, the conic-dual problem of (SDP-P) is as follows:

$$\begin{aligned} \textbf{(SDP-D)} \ \max \ & b^T y \\ \text{s.t.} \ & C - \sum_{i=1}^{m} y_i A_i \succeq 0. \end{aligned} \tag{2.8}$$

The duality theory for semidefinite programming is not as tight as that of linear programming since a gap between the optimal primal and dual objective function values is

possible. From Theorem 2.1.1, we have weak duality between the primal and the dual objective. However, unlike for linear programming, Vandenberghe and Boyd [96] exemplify that optimality does not imply a zero gap between the primal and the dual objective. Slater constraint qualification provides sufficient (but not necessary) conditions for strong duality, see Theorem 2.1.2.

Linear programming and second-order cone programming are a particular case of SDP problems. In particular, the linear programming problem: $\min \{c^T x : Ax = b,\ x \geq 0\}$ where $A \in \mathbb{R}^{m \times n}$, can be written as the SDP problem

$$\min \{c^T \operatorname{Diag}(x) : \langle A,\ \operatorname{Diag}(x) \rangle = b,\ \operatorname{Diag}(x) \succeq 0\},$$

where $\operatorname{Diag}(x)$ is a diagonal matrix with the vector x on the diagonal.

In addition, SOC is a particular case of SDP since $\|x\|_2 \leq t$ can be written as a positive semidefinite constraint as follows:

$$\|x\|_2 \leq t \quad \Leftrightarrow \quad \begin{pmatrix} tI & x \\ x^T & t \end{pmatrix} \succeq 0.$$

For several combinatorial problems, the LP relaxations are weak, even when using the reformulation-linearization methods, unless other effective constraints are added which usually are computationally expensive. The bounds provided by using SOC relaxations are not as weak as those from LP relaxations which make it attractive to solve problems that cannot be handled by LP [63]. Although in many cases the SDP relaxation provides a better bound than the LP and SOC relaxations, this comes at the expense of the computational time required to solve the SDP which becomes rapidly huge and expensive. Theoretical results and numerical experiments [43, 44] show that the computational cost of solving SOC is much less than that of solving SDP, and is similar to that of LP. Solving a large-scale SDP problem remains a difficult problem though in the past decade a lot of progress has been made in this direction [10, 35].

2.2 Polynomial Programming

Given an n-tuple $\alpha = (\alpha_1, \cdots, \alpha_n)$ where $\alpha_i \in \mathbb{N}$, the total degree of the monomial $x^\alpha := x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ is the non-negative integer $d = |\alpha| := \sum_i^n \alpha_i$. A polynomial is a finite linear combination of monomials

$$f(x) = \sum_\alpha f_\alpha x^\alpha = \sum_\alpha f_\alpha x_1^{\alpha_1} \cdots x_n^{\alpha_n} = \langle f, \mathcal{M}_d(x) \rangle ,$$

where the vector of coefficients $f \in \mathbb{R}^N$ with $N = \binom{n+d}{d}$ and $\mathcal{M}_d(x) = (x^\alpha)_{|\alpha| \leq d}$ is the vector of monomials of x of degree at most d. Recall that polynomial programs are of the form:

$$\begin{aligned} \textbf{(PP-P)}\ z_{PP} = \ & \max\ f(x) \\ & \text{s.t. } g_i(x) \geq 0 \quad 1 \leq i \leq m, \end{aligned} \tag{2.9}$$

where $f(x)$ and $g_i(x)$ are multivariate polynomials.

Linear, binary, and linear complementary problems are special cases of polynomial programming problems and hence various applications can be expressed in terms of polynomial problems (see Example 2.2.1).

Example 2.2.1.

$$\begin{aligned} &\min\ x_1^3 + x_1^2 x_2 x_3 \\ &s.t.\ x_1^2 + 5x_2x_3 + 1 \geq 0 && (2.10)\\ &\quad - x_1^2 - x_2^2 - x_3^2 + 1 \geq 0 && (2.11)\\ &\quad x_1^2 - 1 = 0 && (2.12)\\ &\quad x_2 x_3 = 0 && (2.13)\\ &\quad x_2 \geq 0 \quad x_3 \geq 0. && (2.14) \end{aligned}$$

where (2.12) is a binary constraint and (2.13) is a complementary constraint.

Consider λ to be the optimal value for (PP-P), then λ is the smallest value such that $\lambda - f(x) \geq 0$ for all $x \in S := \{x : g_i(x) \geq 0;\ 1 \leq i \leq m\}$. As a result, we can express problem (PP-P) as:

$$\begin{aligned} \textbf{(PP-D)}\ z_{PP} = \ & \min \lambda \\ & \text{s.t. } \lambda - f(x) \geq 0 \quad \forall x \in S. \end{aligned} \tag{2.15}$$

The main idea for solving polynomial programming problems is to relax (PP-D) to solve an easier problem based on the construction of non-negative certificates. For instance, one way to obtain computable relaxations is to represent non-negative polynomials as sum-of-squares polynomials. To obtain a tractable relaxation, one can use a sum-of-squares decomposition with restricted degree of the (unknown) polynomials which can be re-phrased in terms of a linear system of equations involving positive semidefinite matrices [91, 100, 104].

2.2.1 Sum-of-Squares

Consider a polynomial $p(x) \in \mathbf{R}_d[x]$ with n variables and degree d. We are interested in studying when $p(x) \geq 0 \qquad \forall x \in \mathbb{R}^n$ which can be written as the conic decision problem $p(x) \in \mathcal{P}_d(\mathbb{R}^n)$. A necessary condition is that the degree of p is even. A sufficient condition is the existence of a sum-of-squares decomposition, i.e., there exist polynomials $q_1(x), \cdots, q_N(x)$ such that $p(x) = \sum_{i=1}^{N} q_i(x)^2$, i.e., $p(x) \in \Psi_d[x]$. If $p(x)$ is a sum-of-squares polynomial then it is a non-negative polynomial for all values of x; however the converse does not hold. Hilbert [39] studied the sum-of-squares representations of homogeneous polynomials (forms) of even degree d with n variables. He showed that in the homogeneous case, quadratic forms, forms of two variables, and forms of degree four with three variables are non-negative over $\mathbb{R}^n$ if and only if they have SOS representations. Theorem 2.2.2 translates Hilbert's results to non-homogeneous polynomials as follows:

Theorem 2.2.2. *[39] $\Psi_d[x] = \mathcal{P}_d(\mathbb{R}^n)$ if and only if $n = 1$, $d = 2$, or $n = 2$ and $d = 4$.*

Hilbert showed that for $n = 3, d \geq 6$ and $n \geq 4, d \geq 4$, there exist non-negative polynomials with no SOS representation for general classes of polynomials. For instance, the Motzkin polynomial $x_1^4 x_2^2 + x_1^2 x_2^4 - 3x_1^2 x_2^2 + 1$ is non-negative but not an SOS [66].

An advantage of approximating the non-negativity constraint using sum-of-squares is that checking if a polynomial is sum-of-squares is equivalent to solving an SDP problem and hence it can be solved efficiently in polynomial time. The reason is the following theorem:

Theorem 2.2.3. *[91] A polynomial $p(x)$ of degree d is SOS if and only if $p(x) = \sigma(x)^T Q \sigma^T(x)$, where $Q \in \mathcal{S}_+^N$ and σ is the vector of monomials of degree at most $\frac{d}{2}$ in the x variables (of length N), i.e., $\sigma(x) = [x^\alpha]$ with $|\alpha| \leq \frac{d}{2}$ and $N = \binom{n+d/2}{d/2}$.*

Proof. Assume $p(x)$ is SOS. Then $p(x) = \sum_i q_i(x)^2$. Each polynomial $q_i(x)$ can be written as $q_i(x) = \delta_i^T \sigma(x)$, where $\delta_i^T \in \mathbb{R}^N$. Therefore,

$$p(x) = \sum_{i=1}^{k} q_i(x)^2 = \sum_{i=1}^{k} \left(\delta_i^T \sigma(x)\right)^2 = \sigma(x)^T \sum_{i=1}^{k} \left(\delta_i \delta_i^T\right) \sigma(x) = \sigma(x)^T Q \sigma(x)$$

where $Q = \sum_{i=1}^{k} \left(\delta_i \delta_i^T\right) \in \mathcal{S}_+^N$.

Now assume $p(x) = \sigma(x)^T Q \sigma(x)$ where $Q \in \mathcal{S}_+^N$. Then, we can factorize Q and obtain $Q = \sum_{i=1}^{k} \delta_i \delta_i^T$, where $\delta_i \in \mathbb{R}^N$. Hence,

$$p(x) = \sigma(x)^T Q \sigma(x) = \sigma(x)^T \sum_{i=1}^{k} \left(\delta_i \delta_i^T\right) \sigma(x) = \sum_{i=1}^{k} \left(\delta_i^T \sigma(x)\right)^2$$

and one can take $q_i(x) = \delta_i^T \sigma(x)$. □

In other words, every SOS polynomial can be written as a quadratic form in a set of monomials, σ of cardinality $\binom{n+d/2}{d/2}$ where d is the degree of p, with the corresponding matrix being positive semidefinite.

In the representation $p(x) = \sigma(x)^T Q \sigma^T(x)$, for the right and left-hand sides of the equation to be identical, all the coefficients of the corresponding monomials should be equal which is a linear system of equations in terms of the coefficients of p and the enteries of Q. Hence, Q is constrained by linear equalities and a positive semidefinite constraint. As a result, the sum-of-squares representation problem can be easily seen to be equivalent to an SDP

feasibility problem in the standard primal form:

$$\begin{aligned} \sum_{i,j:i+j=\alpha} Q_{ij} &= p_\alpha \qquad \forall |\alpha| \leq d \\ Q &\succeq 0. \end{aligned} \tag{2.16}$$

The size of the matrix Q in the corresponding SDP is $\binom{n+d/2}{d/2} \times \binom{n+d/2}{d/2}$. In addition, we have $\binom{n+d}{d}$ equality constraints. This problem is polynomial time solvable if d is fixed.

Example 2.2.4. *Consider the polynomial* $p(x_1, x_2) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2$*:*

$$p(x_1, x_2) = \begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} Q \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_2 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 4 & -1.5 \\ 0 & -1.5 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix}$$

Since Q is a positive semidefinite matrix, $p(x_1, x_2)$ has a sum-of-squares decomposition which immediately establishes global non-negativity, i.e., $p(x_1, x_2) \geq 0$ for all x_1, x_2.

2.2.2 Polynomial Programming Relaxations

For the unconstrained version of (PP-P):

$$z_{PP} = \max_x \ f(x) \qquad \equiv \qquad \min_\lambda \{\lambda : \lambda - f(x) \geq 0 \, \forall x \in \mathbb{R}^n\}.$$

Relaxing the condition $\lambda - f(x) \geq 0$ to $\lambda - f(x) \in \Psi_d[x] \subseteq \mathcal{P}_d(\mathbb{R}^n)$, we have the following:

$$\begin{aligned} z_{PP} \ = \ \min_\lambda \ & \lambda \\ \text{s.t. } & \lambda - f(x) \geq 0 \end{aligned} \qquad \leq \qquad \begin{aligned} \mu_{sos} \ = \ \min_{\lambda, Q} \ & \lambda \\ \text{s.t. } & \lambda - f(x) = \sigma(x)^T Q \sigma^T(x) \\ & Q \succeq 0. \end{aligned}$$

Therefore, μ_{sos} is an upper bound on z_{PP}.

For the constrained version of (PP-P),

$$z_{PP} = \max_{x \in S} f(x) \qquad \equiv \qquad \min_{\lambda} \{\lambda : \lambda - f(x) \geq 0 \, \forall x \in S\}.$$

The condition $\lambda - f(x) \in \mathcal{P}_d(S)$ is $\mathcal{NP}$-hard in general for interesting cases of S and d as discussed in Section 2.2.3. Relaxing this condition to $\lambda - f(x) \in \mathcal{K}$ for a suitable $\mathcal{K} \subseteq \mathcal{P}_d(S)$ and defining

$$\begin{aligned} \mu_{\mathcal{K}} \; = \; & \min_{\lambda} \lambda \\ & \text{s.t. } f(x) - \lambda \in \mathcal{K}, \end{aligned}$$

we have $\mu_{\mathcal{K}} \geq z_{PP}$. Finding a good approximation of $\mathcal{P}_d(S)$ is a key factor in obtaining a good bound on the problem. At the same time, we need a tractable approximation, i.e., one that uses LP, SOC, and SDP cones, and thus can be solved efficiently using interior-point methods.

For instance, using Putinar's representation theorem [87]:

Example 2.2.5. *Assume there exists a real-valued polynomial $u(x)$ with $u(x) \in \Psi[x] + \sum_{i=1}^{m} \Psi[x] g_i(x)$ such that $\{x \in \mathbb{R}^n : u(x) \geq 0\}$ is compact then*

$$\mathcal{P}(S) \supseteq \left\{ p(x) : p(x) = \sigma_0(x) + \sum_{i=1}^{m} \sigma_i(x) g_i(x), \; s(x) \in \Psi[x] \right\} \supseteq \mathcal{P}^+(S).$$

Define $g_0(x) = 1$ and $G = \{g_i(x) : i = 0, \ldots, m\}$. For $r > 0$ defining

$$\Gamma_G^r = \left(\Psi_r[x] + g_1(x) \Psi_{r - \deg(g_1)}[x] + \cdots + g_m(x) \Psi_{r - \deg(g_m)}[x] \right) \cap \mathbf{R}_d[x],$$

we have

$$\Gamma_G^2 \subseteq \Gamma_G^3 \subseteq \Gamma_G^4 \subseteq \ldots \subseteq \Gamma_G^r \subseteq \mathcal{P}_d(S) \text{ and } \mathcal{P}_d^+(S) \subseteq \bigcup_{r=0}^{\infty} \Gamma_G^r.$$

Remark: We use the notation $\Gamma_G^r \uparrow \mathcal{P}_d(S)$ of [104] as a short hand for the following two conditions $\Gamma_G^r \subseteq \Gamma_G^{r+1} \subseteq \mathcal{P}_d(S)$ and $\mathcal{P}_d^+(S) \subseteq \bigcup_{r=0}^{\infty} \Gamma_G^r$.

Notice that $\Gamma_G^r \uparrow \mathcal{P}_d(S)$ implies $\mu_{\Gamma_G^r} \uparrow z_{PP}$.

So, using Putinar's representation theorem, replacing the non-negativity condition by a tractable stronger condition in terms of sum-of-squares, one obtains a hierarchy of upper bounds $\mu_{\Gamma_G^r}$ on z_{PP}, each $\mu_{\Gamma_G^r}$ can be found by solving an SDP relaxation.

2.2.3 Special Cases of Polynomial Programs

In this section, we study special cases of (PP-P).

Unrestricted Polynomial Optimization

From Theorems 2.2.2 and 2.2.3, it follows that optimizing (unrestricted) polynomials of degree d over $\mathbb{R}^n$ can be expressed as an SDP for any d when $n = 1$, for any n when $d = 2$, and for $n = 2$ and $d = 4$ and thus is solvable in polynomial time.

On the other hand, optimizing degree four polynomials over $\mathbb{R}^n$ is $\mathcal{NP}-$hard. This can be shown by considering the problem of partitioning a sequence of n integers $a_1, \cdots, a_n$ in two sets of equal sum, which is known to be $\mathcal{NP}-$hard. The sequence can be partitioned if and only if the optimal solution of

$$\min_{x \in \mathbb{R}^n} \left(\sum_{i=1}^{n} a_i x_i \right)^2 + \sum_{i=1}^{n} (x_i^2 - 1)^2,$$

is zero.

Moreover, deciding whether $p(x) \in \mathcal{P}_4(\mathbb{R}^n)$ contains the problem of deciding whether a matrix is copositive, which is known to be $\mathcal{NP}-$hard [67]. A $n \times n$ matrix A is copositive if

$$\sum_{i,j} A_{ij} x_i^2 x_j^2 \in \mathcal{P}_4(\mathbb{R}^n).$$

Union and Intersection of Regions

Given $S, T \subseteq \mathbb{R}^n$, we study the relationship between optimizing over S, T, $S \cap T$, and $S \cup T$.

Lemma 2.2.6. *Consider the sets S and T, where $S \subseteq T$ then we have $\mathcal{P}_d(S) \supseteq \mathcal{P}_d(T)$.*

Proof. Consider $p(x)$ in $\mathcal{P}_d(T)$. Let $a \in S$, then $a \in T$ and $p(a) \geq 0$. So, $p(x) \in \mathcal{P}_d(S)$. □

Corollary 2.2.7. *Let $S \subseteq T$, then*

$$\begin{aligned} \mu_S \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(S) \end{aligned} \qquad \leq \qquad \begin{aligned} \mu_T \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(T) \end{aligned}$$

Lemma 2.2.8. *For any sets S and T*

$$\mathcal{P}_d(S \cup T) = \mathcal{P}_d(S) \cap \mathcal{P}_d(T).$$

Proof. Using $S, T \subseteq S \cup T$, from Lemma 2.2.6 $\mathcal{P}_d(S), \mathcal{P}_d(T) \supseteq \mathcal{P}_d(S \cup T)$. Hence, $\mathcal{P}_d(S) \cap \mathcal{P}_d(T) \supseteq \mathcal{P}_d(S \cup T)$.

Next consider a polynomial $p(x)$ in $\mathcal{P}_d(S) \cap \mathcal{P}_d(T)$. Let $a \in S \cup T$, then $p(a) \geq 0$, hence $p(x)$ is in $\mathcal{P}_d(S \cup T)$. □

Corollary 2.2.9. *For any sets S and T,*

$$\begin{aligned} \mu_{S\cup T} \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(S \cup T) \end{aligned} \qquad = \qquad \begin{aligned} \mu_{S\cup T} \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(S) \cap \mathcal{P}_d(T) \end{aligned}$$

Lemma 2.2.10. *For any sets S and T*

$$\begin{aligned} \mathcal{P}_d(S \cap T) &\supseteq \sum_{t=0}^{d} \mathcal{P}_t(S)\mathcal{P}_{d-t}(T) \\ &\supseteq \mathcal{P}_d(S) + \mathcal{P}_d(T). \end{aligned}$$

Proof. Consider $p(x) \in \sum_{t=0}^{d} \mathcal{P}_t(S)\mathcal{P}_{d-t}(T)$. There are $q_t(x)$ in $\mathcal{P}_t(S)$ and $r_t(x)$ in $\mathcal{P}_{d-t}(T)$ such that $p(x) = \sum_{t=0}^{d} q_t(x) r_t(x)$. Let $a \in S \cap T$, then $p(a) = \sum_{t=0}^{d} q_t(a) r_t(a) \geq 0$. Hence, $p(x) \in \sum_{t=0}^{d} \mathcal{P}_t(S)\mathcal{P}_{d-t}(T)$.

For the second line of the lemma, we use $\sum_{t=0}^{d} \mathcal{P}_t(S)\mathcal{P}_{d-t}(T) \supseteq \mathcal{P}_d(S)\mathcal{P}_0(T) + \mathcal{P}_0(S)\mathcal{P}_d(T) = \mathcal{P}_d(S)\mathbb{R}_+ + \mathbb{R}_+\mathcal{P}_d(T) = \mathcal{P}_d(S) + \mathcal{P}_d(T)$. □

Corollary 2.2.11. *For any sets S and T*

$$\begin{aligned} \mu_{S\cap T} \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(S\cap T) \end{aligned} \qquad \geq \qquad \begin{aligned} \mu_{S,T} \;=\; & \min_{\lambda} \lambda \\ & s.t.\ \lambda - f(x) \in \mathcal{P}_d(S) + \mathcal{P}_d(T) \end{aligned}$$

Lemma 2.2.12. *For any S, d, and $f \in \mathbf{R}_d[x]$*

$$\mathcal{P}_d\left(S \cap \{x : f(x) \geq 0\}\right) \supseteq \mathcal{P}_d(S) + f(x)\mathcal{P}_{d-\deg(f)}(S).$$

Optimizing over Subsets of Reals

To motivate the result of this section, we first consider the following example.

Example 2.2.13. *Is the polynomial $p(y) = 3y^2+2y-1$ non-negative over the interval $[4,5]$?*

We have $a = 4$, $b = 5$, and $d = 2$. Substituting $y = x + 4$, we obtain

$$\begin{aligned} p_1(x) := p(x+4) &= 3(x+4)^2 + 2x + 8 - 1 \\ &= 3x^2 + 24x + 48 + 2x + 7 \\ &= 3x^2 + 26x + 55. \end{aligned}$$

$p_1(x) = 3x^2 + 26x + 55$ and $p_1(x) \in \mathcal{P}_2([0,1]) \Leftrightarrow p(y) \in \mathcal{P}_2([4,5])$. Substituting $x = \frac{1}{z+1}$, in $p_1(x)$

$$\begin{aligned} p_2(z) := p_1\left(\frac{1}{z+1}\right) &= \frac{3}{(z+1)^2} + \frac{26}{z+1} + 55 \\ &= \frac{3 + 26(z+1) + 55(z+1)^2}{(z+1)^2} \\ &= \frac{3 + 26z + 26 + 55z^2 + 110z + 55}{(z+1)^2} \\ &= \frac{55z^2 + 136z + 84}{(z+1)^2}. \end{aligned}$$

And $p_2(z) \in \mathcal{P}_2(\mathbb{R}_+) \Leftrightarrow p_1(x) \in \mathcal{P}_2([0,1])$. *Substituting* $z = w^2$, *we obtain*

$$p_3(w) := p_2(w^2) = 55w^4 + 136w^2 + 84$$

and $p_3(w) \in \mathcal{P}_2(\mathbb{R}) \Leftrightarrow p_2(z) \in \mathcal{P}_2(\mathbb{R}_+)$. *Since* $p_3(w)$ *is an SOS, it is non-negative for all* $w \in \mathbb{R}$, *then* $p(y)$ *is non-negative for all* $y \in [4,5]$.

The next lemma follows from the results of [86, 17] on sos representations of polynomials that are nonnegative on an interval:

Lemma 2.2.14. *Problem (PP-P) can be expressed as an SDP for any* d *for*

1. $S = [0, \infty)$
2. $S = [a, b]$.

Proof.

1. $p(x) \in \mathcal{P}_d([0,\infty)) \Leftrightarrow p(x^2) \in \mathcal{P}_d(\mathbb{R})$
2. $p(y) \in \mathcal{P}_d([a,b]) \Leftrightarrow p((b-a)x + a) \in \mathcal{P}_d([0,1]) \Leftrightarrow p(\frac{b-a}{z+1} + a)(z+1)^d \in \mathcal{P}_d(\mathbb{R}_+)$

□

From Lemma 2.2.8 and $\mathcal{P}_d(S) = \mathcal{P}_d(\text{closure}(S))$, we have the following corollary.

Corollary 2.2.15. *When* $S \subseteq \mathbb{R}$ *is a finite union of intervals, (PP-P) can be casted as an SDP for any* d.

Optimizing a Linear Polynomial

We consider a linear objective function subject to a set of constraints.

Theorem 2.2.16.

$$\mathcal{P}_1(S) = \left\{ a^T x + b : \begin{bmatrix} a \\ b \end{bmatrix} \in \tilde{S}^* \right\},$$

where $\tilde{S} = \left\{ \begin{bmatrix} ts \\ t \end{bmatrix}^T : s \in S,\ \forall t \geq 0 \right\}$ *and* $\tilde{S}^*$ *is the dual cone of* $\tilde{S}$.

Proof.

$$\begin{aligned}
\left\{ \begin{bmatrix} a \\ b \end{bmatrix} : \begin{bmatrix} a \\ b \end{bmatrix}^T \begin{bmatrix} s \\ 1 \end{bmatrix} \in \mathcal{P}_1(S) \right\} &= \left\{ \begin{bmatrix} a \\ b \end{bmatrix} : \begin{bmatrix} a \\ b \end{bmatrix}^T \begin{bmatrix} s \\ 1 \end{bmatrix} \geq 0 \, \forall s \in S \right\} \\
&= \left\{ \begin{bmatrix} a \\ b \end{bmatrix} : \begin{bmatrix} a \\ b \end{bmatrix}^T \begin{bmatrix} ts \\ t \end{bmatrix} \geq 0 \, \forall s \in S,\ t \geq 0 \right\} \\
&= \left\{ \begin{bmatrix} a \\ b \end{bmatrix} : \begin{bmatrix} a \\ b \end{bmatrix}^T y \geq 0 \, \forall y \in \tilde{S} \right\}, \\
&= \tilde{S}^*,
\end{aligned}$$

□

Hence, the complexity of $\mathcal{P}_1(S)$ depends on the complexity $\tilde{S}^*$.

Example 2.2.17. *Define* $\mathcal{B} := \{x : \|x\|^2 = n\}$, *convex*$(\tilde{\mathcal{B}}) = \mathcal{L}^n$ *and thus* $\tilde{\mathcal{B}}^* = \mathcal{L}^{n*} = \mathcal{L}^n$, *where* $\mathcal{L}^n$ *is the second-order cone, we have:*

$$\mathcal{P}_1(\mathcal{B}) = \left\{ a^T x + b : \begin{bmatrix} a \\ b \end{bmatrix} \in \mathcal{L}^n \right\}.$$

Example 2.2.18. *Consider a polyhedron* H

$$\mathcal{P}_1(H) = \left\{ a^T x + b : \begin{bmatrix} a \\ b \end{bmatrix} \in \tilde{H}^* \right\},$$

and the set $\tilde{H}^*$ *is a polyhedral cone. In this example, (PP-D) is the LP-dual of (PP-P).*

Optimizing over the Ball

Consider the polynomial $f(x)$ of degree one, then $f(x) \geq 0$ for $x \in \mathcal{B}$ can be represented using second-order cone due to the following lemma

Lemma 2.2.19. *$f(x) \in \mathcal{P}_1(\mathcal{B})$ implies that $f(x) = f^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix}$ with $f \in \mathcal{L}^n$.*

Example 2.2.20. *Consider the polynomial $f(x_1, x_2) = 4 + x_1 + x_2$:*

$$f(x_1, x_2) = f^T \begin{pmatrix} \sqrt{2} \\ x_1 \\ x_2 \end{pmatrix}$$

where $f = \begin{pmatrix} 2\sqrt{2} & 1 & 1 \end{pmatrix}^T \in \mathcal{L}^2$ hence $f(x_1, x_2) \in \mathcal{P}_1(\mathcal{B})$.

By the $\mathcal{S}-$Lemma of Yakubovich (see [85]) $p(x) \in \mathcal{P}_2(\mathcal{B})$ may be rewritten as a semidefinite constraint.

Lemma 2.2.21.

$$p(x) \in \mathcal{P}_2(\mathcal{B}) \Leftrightarrow p(x) = \begin{pmatrix} 1 \\ x \end{pmatrix}^T M \begin{pmatrix} 1 \\ x \end{pmatrix} + h(n - \sum_i x_i^2),$$

where $M \succeq 0$ and $h \geq 0$.

Nesterov [71] showed that optimizing a polynomial of degree $d = 3$ on the ball is an $\mathcal{NP}$-hard problem, using a reduction from the maximum stable set problem. de Klerk [15] showed that for any graph $G = (V, E)$ with stability number $\alpha(G)$,

$$\sqrt{1 - \frac{1}{\alpha(G)}} = 3\sqrt{3} \max_{\|x\|^2 + \|y\|^2 = 1} \sum_{\substack{i<j \\ (i,j) \notin E}} y_{ij} x_i x_j.$$

Hence for $d \geq 3$ and $S = \mathcal{B}$, (PP-P) is $\mathcal{NP}$-hard problem.

Quadratic Optimization

Assume

$$S = \{x \in \mathbb{R}^n : q_i(x) \geq 0,\ i = 1, \cdots, m\},$$

where $q_i(x) = x^T Q_i x + 2b_i^T x + c_i$, hence quadratic inequalities (and equalities).

Solving a quadratic problem is known to be $\mathcal{NP}$-hard in general as seen in [28]. On the other hand, solving a convex quadratic problem can be done in polynomial time. Solving an indefinite quadratic objective subject to a single quadratic constraint (inequality or equality) or a concave quadratic inequality constraint and a linear inequality constraint can be solved in polynomial time by first solving a specific form of SDP relaxation, followed by a matrix decomposition procedure as shown in Sturm and Zhang [94] and Yakubovich's $\mathcal{S}$-lemma (see [83, 85]).

Example 2.2.22. *[94]* $\mathcal{P}_2(\mathcal{B} \cap \{x : a^T x \geq b\}) = \mathcal{P}_2(\mathcal{B}) + (a^T x - b)\mathcal{P}_1(\mathcal{B})$

In Ye and Zhang [101], more special cases of quadratic optimization problems are solved in polynomial time by showing that a certain SDP relaxation is exact.

We consider the following general quadratic optimization problem:

$$\begin{aligned} \textbf{(QP)}\ \ & \min\ q_0(x) \\ & \text{s.t. } q_i(x) \geq 0, \qquad i = 1, \cdots, m \end{aligned}$$

A semidefinite programming relaxation of (QP) is

$$\begin{aligned} \textbf{(SDP-P}_\mathbf{Q}\textbf{)}\ \ & \min \left\langle \begin{pmatrix} c_0 & b_0^T \\ b_0 & Q_0 \end{pmatrix}, X \right\rangle \\ & \text{s.t. } \left\langle \begin{pmatrix} c_i & b_i^T \\ b_i & Q_i \end{pmatrix}, X \right\rangle \geq 0 \qquad i = 1, \cdots, m \\ & X_{00} = 1, \quad X \succeq 0 \end{aligned}$$

The SDP problem (SDP-P_Q) has a dual, which is given by

$$\begin{aligned}\textbf{(SDP-D}_\mathbf{Q}\textbf{)}\ \max\, & y_0 \\ \text{s.t. } & \begin{pmatrix} c & b^T \\ b & Q \end{pmatrix} - y_0 \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - \sum_i^m y_i \begin{pmatrix} c_i & b_i^T \\ b_i & Q_i \end{pmatrix} \succeq 0 \\ & y_i \geq 0 \qquad i = 1, \cdots, m\end{aligned}$$

Assuming (QP) satisfies the Slater condition, it follows that (SDP-P_Q) satisfies the Slater condition too. Additionally, (SDP-D_Q) satisfies the Slater condition in the following cases as shown in [101].

Proposition 2.2.23. *The problem (SDP-D_Q) satisfies the Slater regularity condition, either when at least one of the m constraints is ellipsoidal or when the objective function is strictly convex.*

Theorem 2.2.24. *[101] Suppose that (SDP-P_Q) and (SDP-D_Q) both satisfy the Slater condition and $m = 2$. Furthermore, suppose that (SDP-P_Q) has at least one non-binding constraint at optimality. Then, (QP) can be solved in polynomial time.*

Another polynomially solvable special case of (QP) is when $m = 2$ and the quadratic functions $q(x)$, $q_1(x)$, and $q_2(x)$ are homogeneous, i.e., there are no linear terms [101].

Optimizing over Polyhedral Cones

We assume that the domain S is a polyhedral cone

$$S = \{x : a_i x - b_i \geq 0\, i = 1, \cdots, m\}.$$

We saw in Example 2.2.18 that in this case $\mathcal{P}_1(S)$ is a polyhedral cone. The optimization of degree 1 polynomials over S corresponds to LP-programming which is known to be polynomial-time solvable. The degree two case is $\mathcal{NP}$-hard, actually Bellare and Rogaway [7] proved that, unless $\mathcal{P} = \mathcal{NP}$, if $\epsilon \in (0, \frac{1}{3})$ there is no polynomial time-approximation algorithm for the problem of minimizing a polynomial of degree $d \geq 2$ over the set $S = \{x \in [0,1]^n : Ax \leq b\}$.

Optimizing over the Simplex

Consider the problem of minimizing a polynomial $p(x)$ of degree d on the standard simplex

$$\Delta := \left\{ x \in \mathbb{R}^n_+ : \sum_{i=1}^{n} x_i = 1 \right\}.$$

The problem

$$\begin{aligned} &\min_x p(x) \\ &\text{s.t. } x \in \Delta \end{aligned} \tag{2.17}$$

is $\mathcal{NP}-$hard even for degree $d = 2$ as it contains the maximum stable set problem. Let G be a graph with adjacency matrix A and I is the identity matrix, then the maximum size $\alpha(G)$ of a stable set in G can be expressed as [18]

$$\frac{1}{\alpha(G)} = \min_{x \in \Delta} x^T (I + A) x.$$

Some easy cases:
When $p(x)$ is concave then the global minimum of p is attained at one of the n extreme points of Δ. When p is convex of degree 2 the problem becomes a convex quadratic problem that may be solved in polynomial time.

Bomze and de Klerk [9] showed that problem (2.17) with $d = 2$ allows a polynomial time approximation scheme (PTAS). This result was extended to polynomials of fixed degree d by de Klerk, Laurent, and Parrilo [18]. On the other hand, this problem does not have a fully polynomial time approximation scheme (FPTAS) [15].

Lemma 2.2.25. *[18] For a fixed degree d, there exists a PTAS for maximizing polynomials of degree d over the unit simplex.*

Using

$$p(x) \in \mathcal{P}_d(\Delta) \Leftrightarrow p(x^2) \in \mathcal{P}_d(\mathcal{B})$$

we obtain the following corollary:

Corollary 2.2.26. *There exists a PTAS for maximizing polynomials with even exponents and fixed degree on the ball.*

Optimizing over the Hypercube

Consider the problem of minimizing a polynomial $p(x)$ of degree d on the hypercube. The problem

$$\begin{aligned} &\min p(x) \\ &\text{s.t. } x \in [0,1]^n \end{aligned} \tag{2.18}$$

is $\mathcal{NP}-$hard even for degree $d = 2$ since it contains the max-cut problem with non-negative weights [15]. Moreover, the max-cut problem does not have a PTAS and hence for $d = 2$ and $S = \{x : x \in [0,1]^n\}$ the optimization problem does not have a PTAS. As a result, optimizing over the unit hypercube is much harder than optimizing over the simplex.

Optimizing over the Vertices of the Hypercube

If $S = \{x : x \in \{0,1\}^n\}$ and $d = 1$, the problem of optimizing a linear objective function over S can be solved in polynomial time since this can be seen as solving a linear objective over the convex hull of S which is a linear programming problem. However, if $S = \{x : A^T x \leq b,\ x \in \{0,1\}^n\}$ and $d = 1$, the problem corresponds to binary programming which is $\mathcal{NP}-$hard [28].

If $S = \{x : x \in \{0,1\}^n\}$ and $d = 2$ then (PP-P) is $\mathcal{NP}-$hard as it contains the max-cut problem as a special case. For the max-cut problem with non-negative weights there are approximation results due to Goemans and Williamson [32] and Nesterov [69]. For non-negative weights, the objective function of the max-cut problem is convex quadratic since the Laplacian matrix of a graph is always positive semidefinite, hence optimizing over $\{0,1\}^n$ and $[0,1]^n$ is the same for that case.

2.2.4 Approximation Hierarchies for Polynomial Programs

Lasserre [53] introduced semidefinite relaxations corresponding to liftings of the polynomial programs into higher dimensions. The construction is motivated by results related to representations of non-negative polynomials as sum-of-squares and the dual theory of moments (see Section 2.2.6). Lasserre shows that the global maximum of $f(x)$ over a set S defined by polynomial inequalities reduces to solving a sequence of sum-of-squares type representations of polynomials that are non-negative on S. The convergence of Lasserre's method is based on the assumption that $G = \{g_1(x), \ldots, g_m(x)\}$, the given description of S, allows the application of Putinar's Theorem [87] (see Example 2.2.5). In particular, it assumes S is compact.

Recall the approximation Γ_G^r of $\mathcal{P}_d(S)$:

$$\Gamma_G^r = \left(\Psi_r[x] + \sum_{i=1}^{m} g_i(x) \Psi_{r-\deg(g_i)}[x] \right) \cap \mathbf{R}_d[x]. \tag{2.19}$$

Using the approximation Γ_G^r, a relaxation on the original polynomial program (PP-P) is obtained,

$$\begin{aligned} \mu_{\Gamma_G^r} = \inf_{\lambda} \ & \lambda \\ \text{s.t. } & \lambda - f(x) \in \Gamma_G^r, \end{aligned}$$

that is,

$$\begin{aligned} \mu_{\Gamma_G^r} = \inf_{\lambda, \sigma_i(x)} \ & \lambda \\ \text{s.t. } & \lambda - f(x) = \sigma_0(x) + \sum_{i=1}^{m} g_i(x) \sigma_i(x) \\ & \sigma_0(x) \in \Psi_r[x] \\ & \sigma_i(x) \in \Psi_{r-\deg(g_i)}[x] \qquad\qquad i = 1, \ldots, m. \end{aligned} \tag{2.20}$$

The optimization problem (2.20) can be reformulated as a semidefinite problem as described in Section 2.2.6. For a problem with n variables and m inequality constraints, the size of the optimization problem (2.20) is as follows:

- One psd matrix of dimension $\binom{n+r}{r}$;
- m psd matrices, each of dimension $\binom{n+r-\deg(g_i)}{r-\deg(g_i)}$ for $i \in \{1, \ldots, m\}$;
- $\binom{n+r}{r}$ constraints.

By increasing the value of r for the approximation Γ_G^r, Lasserre builds up a sequence of convex semidefinite relaxations of increasing size. Under mild conditions the optimal values of these problems converge to the global optimal value of the original non-convex problem (PP-P) [53]. The next theorem states the result using our notation.

Theorem 2.2.27. *[53] Let $S = \{x \in \mathbb{R}^n : g_i(x) \geq 0,\ i = 1, \cdots, m\}$ and $\{g_i(x) : i = 1, \ldots, m\}$. Assume S is a compact semialgebraic set (not necessarily convex) and for some i, $\{x \in \mathbb{R}^n : g_i(x) \geq 0\}$ is compact, then*

$$\Gamma_G^r \uparrow \mathcal{P}_d(S),$$

and therefore

$$\mu_{\Gamma_G^r} \uparrow z_{PP}.$$

Hence, using Lasserre's approach for general polynomial programs one may approach the global optimal value as closely as desired by solving a sequence of semidefinite problems that grow in the size of the semidefinite matrices and in the number of constraints. The larger the degree r, the better the optimal value $\mu_{\Gamma_G^r}$.

2.2.5 Handling Equality Constraints

As opposed to Lasserre's sequence of relaxations of $\mathcal{P}_d(S)$ where equality constraints are treated as a set of two inequalities, we differentiate between equality and inequality constraints as proposed in [80]. Given $S = \{x : g_i(x) \geq 0,\ i = 1, \ldots, m_1,\ h_i(x) = 0,\ i =$

$1, \ldots, m_2\}$, let $G = \{g_i(x) : i = 1, \ldots, m_1,\ h_i(x) : i = 1, \ldots, m_2\}$. Consider the following approximation of $\mathcal{P}_d(S)$:

$$\mathbb{K}_G^r = \left(\Psi_r[x] + \sum_{i=1}^{m_1} g_i(x)\Psi_{r-\deg(g_i)}[x] + \sum_{i=1}^{m_2} h_i(x)\mathbf{R}_{r-\deg(h_i)}[x] \right) \cap \mathbf{R}_d[x]. \tag{2.21}$$

The corresponding optimization problem over S can be written as:

$$\begin{aligned} \inf_{\lambda,\sigma_i(x),\delta_i(x)} \ & \lambda \\ \text{s.t. } & \lambda - f(x) = \sigma_0(x) + \sum_{i=1}^{m_1} \sigma_i(x) g_i(x) + \sum_{i=1}^{m_2} \delta_i(x) h_i(x) \\ & \sigma_0(x) \in \Psi_r[x] \\ & \sigma_i(x) \in \Psi_{r-\deg(g_i)}[x] && i = 1, \ldots, m_1 \\ & \delta_i(x) \in \mathbf{R}_{r-\deg(h_i)}[x] && i = 1, \ldots, m_2. \end{aligned} \tag{2.22}$$

Similar to problem (2.20), problem (2.22) can also be reformulated as a semidefinite problem. The following theorem follows by applying Corollary 1 of [80] and Putinar's theorem [87]

Theorem 2.2.28. *The sequence of cones $\mathbb{K}_G^r$ satisfies*

$$\mathbb{K}_G^r \uparrow \mathcal{P}_d(S).$$

Hence $\mu_{\mathbb{K}_G^r} \uparrow z_{PP}$.

Lemma 2.2.29. *Given $G = \{g_i(x) : i = 1, \ldots, m_1,\ h_i(x) : i = 1, \ldots, m_2\}$, let $\tilde{G} = \{g_i(x) : i = 1, \ldots, m_1, h_i(x), -h_i(x) : i = 1, \ldots, m_2\}$. Then $\mathit{\Gamma}_{\tilde{G}}^r \subsetneq \mathbb{K}_G^r$.*

Notice that from equations (2.19) and (2.21), $\mathbb{K}_G^r = \mathit{\Gamma}_{\tilde{G}}^r$ for even r, since $\mathbf{R}_r[x] = \Psi_r[x] - \Psi_r[x]$. However, when r is odd, $\Psi_r[x] = \Psi_{r-1}[x]$ so that $\mathbf{R}_r[x] \supsetneq \Psi_{r-1}[x] - \Psi_{r-1}[x]$, and thus $\mathit{\Gamma}_{\tilde{G}}^r \subset \mathbb{K}_G^r$.

2.2.6 Primal and Dual Perspective

Recall the polynomial programming problem and its reformulation:

$$\begin{array}{ll} \max_x & f(x) \\ \text{s.t.} & x \in S \end{array} \qquad \begin{array}{ll} \min_{\lambda\in\mathbb{R}} & \lambda \\ \text{s.t.} & \lambda - f(x) \in \mathcal{P}_d(S). \end{array} \tag{2.23}$$

We will abuse the notation and we identify $\mathbf{R}_d[x]$ with $\mathbb{R}^N$ where $N = \binom{n+d}{d}$, i.e. by identifying each polynomial $f(x) \in \mathbf{R}_d[x]$ with its vector of coefficients $f \in \mathbb{R}^N$. In this way $\mathcal{P}_d(S)$ is a cone in $\mathbb{R}^N$. We endow $\mathbb{R}^N$ with an inner product $\langle \cdot, \cdot \rangle$ such that for each $f(x) \in \mathbf{R}_d[x]$ and each $u \in \mathbb{R}^n$, $\langle f, \mathcal{M}_d(u) \rangle = f(u)$.

Then we define conic primal-dual pair that form the basis of the relaxations and algorithms that we develop in later chapters:

$$\begin{array}{ll} \min_{\lambda\in\mathbb{R}} & \lambda \\ \text{s.t.} & \lambda - f(x) \in \mathcal{P}_d(S) \end{array} \qquad \begin{array}{ll} \max_Y & \langle f, Y \rangle \\ \text{s.t.} & \langle 1, Y \rangle = 1 \\ & Y \in \mathcal{P}_d(S)^*, \end{array} \tag{2.24}$$

where $\mathcal{P}_d(S)^*$ is the dual cone defined in Section 2.1. $\mathcal{P}_d(S)^*$ is actually the cone generated by the convex hull of the set of monomials over S, as stated in the following lemma.

Lemma 2.2.30. *Let $S \subseteq \mathbb{R}^n$ be a compact set, then*

$$\{X \in \mathcal{P}_d(S)^* : \langle 1, X \rangle = 1\} = \mathit{conv}(\mathcal{M}_d(S)).$$

Proof. Let $\mathcal{M}_d(S) = \{\mathcal{M}_d(x) : x \in S\}$. Then

$$\begin{aligned} \mathcal{M}_d(S)^* &= \{p : \langle p, X \rangle \geq 0 \, \forall X \in \mathcal{M}_d(S)\} \\ &= \{p(x) : \langle p, \mathcal{M}_d(s) \rangle \geq 0 \, \forall s \in S\} \\ &= \{p(x) : p(s) \geq 0 \, \forall s \in S\} \\ &= \mathcal{P}_d(S). \end{aligned}$$

Therefore, $\mathcal{P}_d(S)^* = \mathcal{M}_d(S)^{**} = \text{closure}(\text{cone}(\text{conv}(\mathcal{M}_d(S))))$. Since S is compact, $\mathcal{M}_d(S)$ is compact. Also, for all $x \in S$, $\langle 1, M_d(x)\rangle = 1$ and thus

$$\{X \in \mathcal{P}_d(S)^* : \langle 1, X\rangle = 1\} = \{X \in \mathcal{M}_d(S)^{**} : \langle 1, X\rangle = 1\} = \text{conv}(\mathcal{M}_d(S)).$$

□

In the following two sections we show that Lasserre's hierarchy of semidefinite relaxations obtained using the idea of moments is analogous to our approach when the approximation of $\mathcal{P}_d(S)$ has Putinar's representation result (as given in (2.19)). In the next section, we show how to obtain (2.23) using the moment problem.

Moment Perspective

Problem (2.9) is of the form:

$$\begin{aligned} z_{PP} = \max_x\ & f(x) \\ \text{s.t. } & x \in S. \end{aligned} \tag{2.25}$$

For S a compact semialgebraic set, Lasserre [53] reformulates (2.25) as

$$\begin{aligned} z_{mom} = \max_{\mu \in \mathbf{M}(S)}\ & \int_S f d\mu \\ \text{s.t. } & \int_S d\mu = 1, \end{aligned} \tag{2.26}$$

where $\mathbf{M}(S)$ is the space of finite measures supported on S.

Theorem 2.2.31. *[53] Problems* (2.25) *and* (2.26) *are equivalent, that is* $z_{PP} = z_{mom}$.

Proof. We consider the case where z_{PP} is bounded. Since $f(x) \leq z_{PP}$ for all $x \in S$, then for any $\mu \in \mathbf{M}(S)$, $\int_S f d\mu \leq z_{PP}$ and thus $z_{mom} \leq z_{PP}$. Conversely, with every $x \in S$, we associate the Dirac measure at x in $\mathbf{M}(S)$ which is a feasible solution of problem (2.26) and hence $z_{mom} \geq z_{PP}$. This leads to the desired result $z_{PP} = z_{mom}$. □

In other words, one can formulate the general polynomial program as a moment problem. In contrast to problem (2.25), problem (2.26) is linear, convex, and a conic problem. However, (2.26) is infinite dimensional. The moment problem

$$\begin{aligned} \max_{\mu \in \mathbf{M}(S)} & \int_S f d\mu \\ \text{s.t.} & \int_S d\mu = 1. \end{aligned}$$

can be written as an infinite dimensional LP

$$\begin{aligned} \max_{\mu \in \mathbf{M}(S)} & \langle f, \mu \rangle \\ \text{s.t.} & \langle \mathbf{1}, \mu \rangle = 1, \end{aligned}$$

where $\mathbf{1}$ is the vector of all ones. The dual LP reads:

$$\begin{aligned} \min_{\lambda \in \mathbb{R}} & \lambda \\ \text{s.t.} & \lambda - f(x) \geq 0 \qquad \forall\, x \in S, \end{aligned}$$

and hence obtaining (2.23).

Semidefinite Relaxation

Let $y := (y_\alpha)_{\alpha \in \mathbb{N}^n}$ be an infinite sequence. We say, y has a representing measure $\mu \in \mathbf{M}(S)$, if

$$\exists \mu \in \mathbf{M}(S) \qquad \text{s.t.} \qquad y_\alpha = \int_S x^\alpha d\mu \qquad \forall \alpha \in \mathbb{N}^n.$$

Given y and $p(x) \in \mathbf{R}[x]$, let $L_y : \mathbf{R}[x] \to \mathbb{R}$ be a linear function where each monomial x^α in $p(x)$ is replaced by y_α:

$$L_y(p) = \sum_\alpha p_\alpha y_\alpha.$$

From Putinar's Theorem [87], if the set $\{x : g_i(x) \geq 0\}$ is compact for some $i = 1, \ldots, m$, then y has a representing measure μ on S if and only if

$$L_y(\sigma^2),\ L_y(\sigma^2 g_i) \geq 0, \quad \forall i = 1, \ldots, m \quad \forall \sigma(x) \in \mathbf{R}[x]. \tag{2.27}$$

We show that for a fixed degree r, checking condition (2.27) for all $\sigma(x) \in \mathbf{R}_r[x]$ reduces to solving a semidefinite problem. This allows us to obtain a SDP relaxation of (2.26).

One can introduce the moment matrix $M_r(y)$ with rows and columns indexed in the basis defined by v

$$v := [1, x_1, x_2, \cdots, x_n, x_1^2, x_1x_2, \cdots, x_1x_n, x_2^2, \cdots, x_n^2, x_1^3, \cdots, x_1^r, \cdots, x_n^r]^T,$$

then $M_r(y)$ is of dimension $N = \binom{n+d}{d}$ and is defined by:

$$M_r(y)(\alpha, \beta) := L_y(x^\alpha x^\beta) = L_y(x^{\alpha+\beta}) = y_{\alpha+\beta}, \quad \alpha, \beta \in \mathbb{N}^n, \quad |\ \alpha\ |, |\ \beta\ | \leq r.$$

For instance, for clarification, consider the two-dimensional case. The moment matrix $M_r(y)$ with $r = 2$ consists of block matrix $\{M_{i,j}(y)\}_{0 \leq i,j \leq r}$ and has the following form

$$M_2(y) = \begin{bmatrix} y_{00} & y_{10} & y_{01} \\ y_{10} & y_{20} & y_{11} \\ y_{01} & y_{11} & y_{02} \end{bmatrix}.$$

On the other hand, for a given $g(x) \in \mathbf{R}[x]$, define the localizing moment matrix:

$$M_r(gy)(\alpha, \beta) := L_y(g(x)x^{\alpha+\beta}) = \sum_\gamma g_\gamma y_{\alpha+\beta+\gamma}, \quad \alpha, \beta \in \mathbb{N}^n, \quad |\ \alpha\ |, |\ \beta\ | \leq r.$$

Lemma 2.2.32. *$L_y(s_0^2),\ L_y(s_i^2 g_i) \geq 0$ for all $s_0(x) \in \mathbf{R}_{\lfloor \frac{r}{2} \rfloor}[x]$ and $s_i(x) \in \mathbf{R}_{\lfloor \frac{r-\deg(g_i)}{2} \rfloor}[x] \Leftrightarrow M_r(y), M_{r-\deg(g_i)}(y) \succeq 0$ for all $i = 1, \ldots, m$.*

Proof. Let y be a finite sequence of moments, up to order r. Then y has a representing measure μ with support contained in S:

$$L_y(s_0^2) = \langle s_0, M_r(y)s_0\rangle = \int_S s_0^2 d\mu = \int_S s_0(x)^2\mu(dx) \geq 0, \quad \forall s_0(x) \in \mathbf{R}_{\lfloor \frac{r}{2} \rfloor}[x],$$

and

$$L_y(s_i^2 g_i) = \left\langle s_i, M_{r-\deg(g_i)}(g_i y)s_i\right\rangle = \int_S s_i^2 g_i d\mu = \int_S s_i(x)^2 g_i(x)\mu(dx) \geq 0, \quad \forall s_i(x) \in \mathbf{R}_{\lfloor \frac{r-\deg(g_i)}{2} \rfloor}[x],$$

so that $M_r(y), M_{r-\deg(g_i)}(g_i y) \succeq 0$ for all $i = 1, \ldots, m$. □

Using Lemma 2.2.32, an SDP relaxation for (2.26) can be formulated as:

$$\begin{aligned} \max_y \; & L_y(f) \\ \text{s.t.} \; & L_y(1) = 1 \\ & M_r(y) \succeq 0 \\ & M_{r-\deg(g_i)}(g_i y) \succeq 0 \qquad i = 1, \ldots, m. \end{aligned} \tag{2.28}$$

The SDP-dual of (2.28) can be reformulated as (2.20)

Chapter 3

New Conic Relaxations

In this chapter, we present new relaxations for BPP problems. In particular, we use characterizations of non-negative degree 1 and 2 polynomials over the ball to propose SDP/SOC relaxations of binary polynomial programs. We focus on BPPs that are of degree two and use the polynomial programming framework to re-derive, compare and strengthen existing relaxation schemes for binary quadratic polynomial programs (BQPP). We present a new second-order and semidefinite-based construction where we are able to theoretically show that the resulting relaxations provide bounds stronger than other computationally practical semidefinite-based relaxations proposed in the literature. In Section 3.1, we describe our solution methodology and present several relaxations for the binary quadratic polynomial problem including our three new proposed relaxations.

In Section 3.2, we apply our proposed relaxations to general quadratic constrained problems, quadratic assignment problems, quadratic linear constrained problems, and quadratic knapsack problems and theoretically compare them to other existing relaxations from the literature. In Section 3.3, we report computational results for BQPPs exploring the performance of several relaxations, comparing them to existing ones in terms of bounds and computational time on the four classes of problems mentioned above.

3.1 Binary Quadratic Polynomial Programming

Binary quadratic polynomial programming problem is a classical combinatorial problem. It is the problem of minimizing or maximizing a quadratic function of several binary variables, subject to quadratic and linear constraints. The problem can be formally expressed as:

$$
\begin{aligned}
\textbf{(BQPP)}\ z_{\text{BQPP}} = \ & \max\ x^T Q x + p^T x \\
& \text{s.t. } a_j^T x = b_j && j = 1, \cdots, t && (3.1)\\
& \quad c_j^T x \leq d_j && j = 1, \cdots, u && (3.2)\\
& \quad x^T F_j x + e_j^T x = k_j && j = 1, \cdots, v && (3.3)\\
& \quad x^T G_j x + h_j^T x \leq l_j && j = 1, \cdots, w && (3.4)\\
& \quad x_i \in \{-1, 1\} && i = 1, \cdots, n. && (3.5)
\end{aligned}
$$

Note that constraint (3.5) can be modified to allow some continuous variables. In this part, we focus on pure binary quadratic polynomial programs although the solution methodology can be applied to mixed-binary quadratic polynomial programs with bounded continuous variables. Furthermore, although one could consider an equivalent form of the problem with only inequality constraints, we treat equality and inequality constraints separately because this is beneficial from a computational perspective (see Section 2.2.5).

There are many well-known problems that can be naturally written as binary quadratic polynomial problems. For instance, folding of proteins in three-dimension by Phillips and Rosen [81], machine scheduling and unconstrained task allocation by Alidaee, Kochenberger, and Ahmadian [1], capital budgeting and financial analysis such as in Laughhunn [56], as well as other examples arising in physics and engineering applications such as the spin glass problem and circuit board layout design by Grötschel, Jünger, and Reinelt [33]. Furthermore, Boros and Hammer [11] and Boros and Prekopa [12] formulated many satisfiability problems as BQPPs. In addition, there are several applications related to combinatorial problems such as the single-row facility layout problem [3] and the quadratic assignment problem [60].

3.1.1 Polynomial Programming-Based Relaxations

Following Section 2.2, (BQPP) can be reformulated as

$$\begin{aligned} &\min_{\lambda} \lambda \\ &\text{s.t. } \lambda - q(x) \in \mathcal{P}_2(D \cap H), \end{aligned}$$

where $q(x) = \sum_{i,j} Q_{ij}x_ix_j + \sum_i p_ix_i$, $D = \{x : a_j^T x = b_j, c_j^T x \geq d_j, x^T F_j x + e_j^T x = k_j, x^T G_j x + h_j^T x \geq l_j\}$, and $H = \{-1, 1\}^n$. Since even checking if a polynomial is in $\mathcal{P}_2(H)$ is $\mathcal{NP}$-hard, tractable approximations of $\mathcal{P}_2(D \cap H)$ are needed. Using Section 2.2.5, a hierarchy of approximations to $\mathcal{P}_2(D \cap H)$ is obtained using the cone $\mathbb{K}_G^r \subseteq \mathcal{P}_2(D \cap H)$, where G is the set of polynomials defining D and H. The result is a hierarchy of relaxations:

$$\begin{aligned} \textbf{(BQPP}_{\mathbb{K}_G^r}\textbf{)} \quad \mu_{\text{BQPP}_{\mathbb{K}_G^r}} = &\min_{\lambda} \lambda \\ &\text{s.t. } \lambda - q(x) \in \mathbb{K}_G^r \end{aligned} \tag{3.6}$$

whose optimal value converges to the optimal value of (BQPP).

The size of the relaxations produced in (3.6) grows exponentially in r. For this reason, instead of looking at the hierarchy of relaxations, we will concentrate on the first and simplest relaxation where $r = 2$, i.e., $\mathbb{K}_G^2$. We study how to improve the approximation of $\mathcal{P}_2(D \cap H)$ using variations of the cone $\mathbb{K}_G^2$. The fundamental tool that we use to construct such inner approximations of $\mathcal{P}_2(D \cap H)$ is Lemma 2.2.19, a representation theorem for non-negative linear polynomials over $\mathcal{B}$ which results in second-order cone conditions. These yield stronger bounds than $\mathbb{K}_G^2$ with an insignificant impact on the computational time.

3.1.2 New Conic Relaxations of BQPP

In this section, we present three relaxations for the BQPP problem. Two of these relaxations are based on second-order cone and semidefinite programming and the final relaxation is solely based on second-order cone programming.

SOC-SDP-based Relaxations of BQPP

Recall the previous polynomial formulation of the binary quadratic polynomial problem. First, notice that $x \in H$ implies $\|x\|^2 = n$. Given $\mathcal{B} = \{x \in \mathbb{R}^n : \|x\|^2 = n\}$, we have $D \cap H \subseteq \mathcal{B}$ and by defining $\bar{\mathcal{K}}_2$ as

$$\begin{aligned}\bar{\mathcal{K}}_2 = \mathcal{P}_2(\mathcal{B}) + \sum_i (1+x_i)\mathcal{P}_1(\mathcal{B}) + \sum_i (1-x_i)\mathcal{P}_1(\mathcal{B}) + \sum_i (1-x_i^2)\mathbf{R}_0 + \sum_j (b_j - a_j^T x)\mathbf{R}_1[x] \\ + \sum_j (d_j - c_j^T x)\mathcal{P}_1(\mathcal{B}) + \sum_j (k_j - x^T F_j x - e_j^T x)\mathbf{R}_0 + \sum_j (l_j - x^T G_j x - h_j^T x)\mathbf{R}_0^+,\end{aligned}$$

we have $\bar{\mathcal{K}}_2 \subseteq \mathcal{P}_2(D \cap \mathcal{B} \cap H) = \mathcal{P}_2(D \cap H)$.

Using Lemmas 2.2.19 and 2.2.21 and the fact that the vector $[n \quad x^T]^T$ belongs to the second order cone, we can write the condition $\lambda - q(x) \in \bar{\mathcal{K}}_2$ as

$$\begin{aligned}\lambda - q(x) =& s(x) + \sum_i \alpha_i(x)(1+x_i) + \sum_i \beta_i(x)(1-x_i) + \sum_i \gamma_i(1-x_i^2) + \sum_j \delta_j(x)(b_j - a_j^T x) \\ &+ \sum_j \eta_j(x)(d_j - c_j^T x) + \sum_j \theta_j(k_j - x^T F_j x - e_j^T x) + \sum_j \xi_j(l_j - x^T G_j x - h_j^T x),\end{aligned}$$

with $s(x) = \begin{pmatrix}1 & x^T\end{pmatrix} S \begin{pmatrix}1 \\ x\end{pmatrix}$ where $S \in \mathcal{S}_+^{n+1}$ and $\alpha_i(x) = \alpha_i^T \begin{pmatrix}\sqrt{n} \\ x\end{pmatrix}$, $\beta_i(x) = \beta_i^T \begin{pmatrix}\sqrt{n} \\ x\end{pmatrix}$, and $\eta_j(x) = \eta_j^T \begin{pmatrix}\sqrt{n} \\ x\end{pmatrix}$ where $\alpha_i, \beta_i, \eta_j \in \mathcal{L}^n$, $\delta_j(x) \in \mathbf{R}_1[x]$, $\gamma_i, \theta_j \in \mathbb{R}$, and $\xi_j \in \mathbb{R}_+$.

We then obtain the following relaxation of (BQPP):

(BQPP$_{\mathbf{ss}}$) min λ

$$\begin{aligned}\text{s.t. } \lambda - q(x) = &\begin{pmatrix}1 & x^T\end{pmatrix} S \begin{pmatrix}1\\ x\end{pmatrix} + \sum_i (1+x_i)\alpha_i^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} + \sum_i (1-x_i)\beta_i^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} \\ &+ \sum_i \gamma_i(1-x_i^2) + \sum_j \delta_j(x)(b_j - a_j^T x) + \sum_j (d_j - c_j^T x)\eta_j^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} \\ &+ \sum_j \theta_j(k_j - x^T F_j x - e_j^T x) + \sum_j \xi_j(l_j - x^T G_j x - h_j^T x), \\ S \in \mathcal{S}_+^{n+1}, \qquad &\alpha_i, \beta_i, \eta_j \in \mathcal{L}^n, \qquad \delta_j(x) \in \mathbf{R}_1[x], \qquad \gamma_i, \theta_j \in \mathbb{R}, \qquad \xi_j \in \mathbb{R}_+.\end{aligned}$$

To strengthen this relaxation we can add valid inequalities to the original problem (BQPP) which is equivalent to adding more variables to the relaxation due to Lemma 2.2.12.

Notice that products of linear constraints, such as $(d_k - c_k^T x)(1 + x_i)$, $(d_k - c_k^T x)(1 - x_i)$, $(d_k - c_k^T x)(d_l - c_l^T x)$, $(1-x_j)(1-x_i)$, $(1+x_j)(1+x_i)$, and $(1-x_j)(1+x_i)$ are also considered as valid inequalities and can be added to (BQPP$_{\text{ss}}$) to further strengthen the relaxation.

Hence we obtain

$$\begin{aligned}
\textbf{(BQPP}_{\textbf{ss}^+}\textbf{)}\ \min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \begin{pmatrix} 1 & x^T \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_i (1+x_i)\alpha_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i (1-x_i)\beta_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_i \gamma_i(1-x_i^2) + \sum_j \delta_j(x)(b_j - a_j^T x) + \sum_j (d_j - c_j^T x)\eta_j^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_j \theta_j(k_j - x^T F_j x - e_j^T x) + \sum_j \xi_j(l_j - x^T G_j x - h_j^T x) \\
& + \sum_{i,k} \sigma_{ik}(d_k - c_k^T x)(1+x_i) + \sum_{i,k} \mu_{ik}(d_k - c_k^T x)(1-x_i) \\
& + \sum_{k \le l} \nu_{kl}(d_k - c_k^T x)(d_l - c_l^T x) + \sum_{i \le j} \tau_{ij}(1-x_i)(1-x_j) \\
& + \sum_{i \le j} \omega_{ij}(1+x_i)(1+x_j) + \sum_{i,j} \phi_{ij}(1-x_i)(1+x_j) \\
S \in \mathcal{S}_+^{n+1}, \quad & \alpha_i, \beta_i, \eta_j \in \mathcal{L}^n, \quad \delta_j(x) \in \mathbf{R}_1[x], \quad \gamma_i, \theta_j \in \mathbb{R}, \quad \xi_j, \sigma_{ik}, \mu_{ik}, \nu_{kl}, \tau_{ij}, \omega_{ij}, \phi_{ij} \in \mathbb{R}_+.
\end{aligned}$$

Pure SOC-based Relaxations of BQPP

The relaxation (BQPP_{ss}) can further be relaxed by removing the positive semidefinite variable leading to the following relaxation:

$$\begin{aligned}
\min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \sum_i (1+x_i)\alpha_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i (1-x_i)\beta_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i \gamma_i(1-x_i^2) \\
& + \sum_j \delta_j(x)(b_j - a_j^T x) + \sum_j (d_j - c_j^T x)\eta_j^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_j \theta_j(k_j - x^T F_j x - e_j^T x) \\
& + \sum_j \xi_j(l_j - x^T G_j x - h_j^T x), \\
\alpha_i, \beta_i, \eta_j \in \mathcal{L}^n, \quad & \delta_j(x) \in \mathbf{R}_1[x], \quad \gamma_i, \theta_j \in \mathbb{R}, \quad \xi_j \in \mathbb{R}_+.
\end{aligned}$$

One type of valid inequalities that we consider for BQPP is:

$$-1 \leq x_i x_j \leq 1. \tag{3.7}$$

These inequalities are redundant in the presence of the SDP and $x_i^2 = 1$ terms. However, once the SDP term is removed adding these constraints will strengthen the SOC relaxation. Note that there are $O(n^2)$ inequalities and they can be added iteratively to the SOC relaxation using a separation algorithm as described in Section 5.5. In this section, we include all these valid inequalities since we are comparing the quality of the bounds.

Hence, we obtain our proposed SOC-based relaxation:

$$\begin{aligned}
\textbf{(BQPP}_{\textbf{soc}}\textbf{)} \ \min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \sum_i (1+x_i)\alpha_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i (1-x_i)\beta_i^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i \gamma_i (1-x_i^2) \\
& + \sum_j \delta_j(x)(b_j - a_j^T x) + \sum_j (d_j - c_j^T x)\eta_j^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_j \theta_j (k_j - x^T F_j x - e_j^T x) + \sum_j \xi_j (l_j - x^T G_j x - h_j^T x) \\
& + \sum_{i<j} h_{ij}^+ (1 + x_i x_j) + \sum_{i<j} h_{ij}^- (1 - x_i x_j) \\
\alpha_i, \beta_i, \eta_j \in \mathcal{L}^n, \qquad & \delta_j(x) \in \mathbf{R}_1[x], \qquad \gamma_i, \theta_j \in \mathbb{R}, \qquad \xi_j, h_{ij}^+, h_{ij}^- \in \mathbb{R}_+.
\end{aligned}$$

By construction we have the following theorem relating the three presented relaxations:

Theorem 3.1.1. *Let $\mu_{BQPP_{SOC}}$, $\mu_{BQPP_{SS}}$, and $\mu_{BQPP_{SS^+}}$ be the optimal solution value of ($BQPP_{SOC}$), ($BQPP_{SS}$), and ($BQPP_{SS^+}$) respectively, then*

$$\mu_{BQPP_{SOC}} \geq \mu_{BQPP_{SS}} \geq \mu_{BQPP_{SS^+}} \geq z_{BQPP}.$$

3.2 Applications

In this section, we apply our proposed framework to the following classes of constrained BQPPs:

- General quadratic polynomial problems;
- Quadratic linear constrained problems;
- Quadratic assignment problem;
- Quadratic knapsack problems.

First, we start with the most general class of binary quadratic polynomial problems where we have quadratic and linear constraints. Then we consider the special case with only linear constraints and finally we consider problems with a single linear constraint. We re-derive existing relaxations that have been proposed in the literature for each of these problems and theoretically compare our proposed two SOC-SDP-based relaxations to them. We show theoretically that we obtain stronger relaxations based on applying the methodology of Section 3.1. In addition, in Section 3.3 we compare the relaxations computationally for each of these four classes of binary quadratic problems. Our computational results show that more time efficient relaxations are obtained if the SDP term is omitted.

3.2.1 General Quadratic Polynomial Problems

We consider the general binary quadratic problem (BQPP). Building on the ideas presented in Section 2.2.4, Lasserre [54] introduced SDP relaxations for binary polynomial programs

by approximating $\mathcal{P}_2(D \cap H)$ using the cone Γ_G^r, that is,

$$\begin{aligned}\Gamma_G^r = \Bigg(& \Psi_r[x] + \sum_i (1 - x_i^2)\Psi_{r-2}[x] + \sum_i (x_i^2 - 1)\Psi_{r-2}[x] + \sum_i (b_i - a_i^T x)\Psi_{r-2}[x] \\ & + \sum_i (a_i^T x - b_i)\Psi_{r-2}[x] + \sum_i (d_i - c_i^T x)\Psi_{r-2}[x] + \sum_i (k_i - x^T F_i x - e_i^T x)\Psi_{r-2}[x] \\ & + \sum_i (x^T F_i x + e_i^T x - k_i)\Psi_{r-2}[x] + \sum_i (l_i - x^T G_i x - h_i^T x)\Psi_{r-2}[x] \Bigg) \cap \mathbf{R}_2[x],\end{aligned}$$

for even $r \geq 2$. Taking $r = 2$, we obtain the Lasserre relaxation of order 1 (L1) for (BQPP):

$$\begin{aligned}\textbf{(BQPP}_{\textbf{L1}}\textbf{)} \ \min \ & \lambda \\ \text{s.t.} \ & \lambda - q(x) \in \Gamma_G^2.\end{aligned}$$

Theorem 3.2.1 shows that ($\text{BQPP}_{\text{SS}^+}$) provides the best bound for the BQPP problem while (BQPP_{SS}) has a better bound than Lasserre's relaxation of order 1.

Theorem 3.2.1. *Let $\mu_{BQPP_{L1}}$, $\mu_{BQPP_{SS}}$, and $\mu_{BQPP_{SS^+}}$ be the optimal solution value of ($BQPP_{L1}$), ($BQPP_{SS}$), and ($BQPP_{SS^+}$) respectively, then*

$$\mu_{BQPP_{L1}} \geq \mu_{BQPP_{SS}} \geq \mu_{BQPP_{SS^+}} \geq z_{BQPP}.$$

Proof. Define

$$\begin{aligned}
\mathcal{H}_1 =& \Psi_2[x] + \sum_i (1 - x_i^2)\mathbf{R}_0 + \sum_i (b_i - a_i^T x)\mathbf{R}_1[x] + \sum_i (d_i - c_i^T x)\mathcal{P}_1(\mathcal{B}) \\
&+ \sum_i (k_i - x^T F_i x - e_i^T x)\mathbf{R}_0 + \sum_i (l_i - x^T G_i x - h_i^T x)\mathbf{R}_0^+. \\
\mathcal{H}_2 =& \mathcal{H}_1 + (d_i - c_i^T x)\mathcal{P}_1(\mathcal{B}) + \sum_i (1 + x_i)\mathcal{P}_1(\mathcal{B}) + \sum_i (1 - x_i)\mathcal{P}_1(\mathcal{B}) = \bar{\mathcal{K}}_2. \\
\mathcal{H}_3 =& \mathcal{H}_2 + \sum_{i,k} (1 + x_i)(d_k - c_k^T x)\Psi_0[x] + \sum_{i,k} (1 - x_i)(d_k - c_k^T x)\Psi_0[x] \\
&+ \sum_{k \le l} (d_k - c_k^T x)(d_l - c_l^T x)\Psi_0[x] + \sum_{i \le j} (1 + x_i)(1 + x_j)\Psi_0[x] \\
&+ \sum_{i \le j} (1 - x_i)(1 - x_j)\Psi_0[x] + \sum_{i,j} (1 + x_i)(1 - x_j)\Psi_0[x].
\end{aligned}$$

We have

$$\Psi_1[x] = \mathbf{R}_0^+ \subseteq \mathcal{P}_1(\mathcal{B}) \Rightarrow \mathbb{K}_G^2 \subseteq \mathcal{H}_1.$$

In addition, from Lemma 2.2.29, by setting r to two we have $\mathit{\Gamma}_G^2 \subseteq \mathbb{K}_G^2$ and therefore,

$$\mathit{\Gamma}_G^2 \subseteq \mathbb{K}_G^2 \subseteq \mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \mathcal{H}_3.$$

□

We now compare the relaxations in terms of computational complexity. Table 3.1 summarizes the number of variables (and for SDPs, the dimension) for each of the resulting optimization problems. Recall that the (BQPP) problem has t linear equalities, u linear inequalities, v quadratic equalities, w quadratic inequalities, and n binary variables.

Table 3.1: Problem dimension for various BQPP relaxations.

Relaxation	SDP	SOC	Linear Non-negative	Linear Free
$(\text{BQPP}_{\text{SS}^+})$	1, $(n+1) \times (n+1)$	$(2n+u)$, $(n+1)$	$w + 2tn + \binom{t+1}{2} + 2\binom{n+1}{2} + n^2$	$n + (n+1)t + v$
$(\text{BQPP}_{\text{SS}})$	1, $(n+1) \times (n+1)$	$(2n+u)$, $(n+1)$	w	$n + (n+1)t + v$
$(\text{BQPP}_{\text{L1}})$	1, $(n+1) \times (n+1)$	-	$2n + 2t + 2v + u + w$	-
$(\text{BQPP}_{\text{SOC}})$	-	$(2n+u)$, $(n+1)$	$w + n(n-1)$	$n + (n+1)t + v$

While the complexity of ($\text{BQPP}_{\text{SS}^+}$), ($\text{BQPP}_{\text{SS}}$) and ($\text{BQPP}_{\text{L1}}$) is similar, we see that (BQPP_{SOC}) trades off an $(n+1) \times (n+1)$ SDP matrix variable for $n(n-1)$ linear non-negative variables. If one applies an interior-point method to solve the relaxations, it is not immediately clear that (BQPP_{SOC}) is computationally cheaper. To see this, recall that the most time-consuming step for an interior-point algorithm is solving the Schur complement equation at each iteration. For this task, sparsity of the constraint matrices is key, and in this regard the trade-off is to the advantage of (BQPP_{SOC}). This is because in semidefinite programming, the Schur complement matrix is typically dense even when the constraint matrix is sparse, and hence computing the Cholesky factorization remains expensive. In contrast, for linear programming, a sparse constraint matrix results in a sparse Schur complement matrix, and this sparsity property can be exploited to speed up the computation of the Cholesky factorization [26]. To illustrate that sparsity is present in our relaxations, we present in Figure 3.1 the non-zero elements of the constraint matrices of (BQPP_{SOC}) and (BQPP_{SS}) for an instance of the quadratic knapsack problem with $n = 10$.

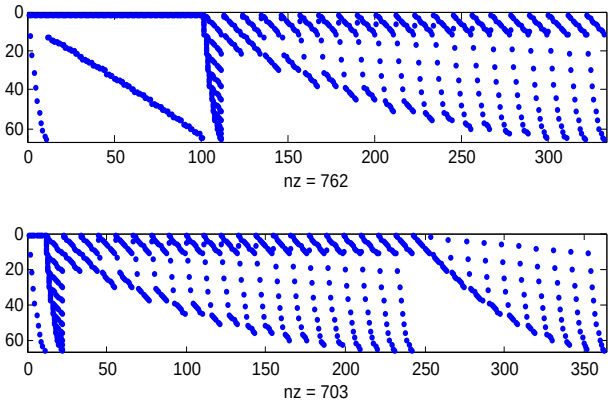

Figure 3.1: Sparsity of the constraint matrices for BQPP_{SOC} (top) and BQPP_{SS} (bottom).

3.2.2 Quadratic Assignment Problem

We consider the quadratic assignment problem (QAP) which is a particular case of BQPP. The quadratic assignment problem, a well-known $\mathcal{NP}$-hard problem, has attracted a lot of attention in literature in terms of theory, applications, and solution methodologies. The QAP was first introduced by Koopmans and Beckmann [50] as a mathematical model related to facility layout problem where you need to assign departments to locations having material flow between departments.

Several applications of the QAP come from assigning facilities to locations with spatial interactions. For example, hospital layout by Elshafei [24] and the assignment of buildings in a University campus by Dickey and Hopkins [22]. In addition, QAP has applications such as designing the keyboards and control panels for typewriter by Pollatschek, Gershoni, and Radday [84], minimizing the number of connections between components in a backboard wiring by Steinberg [92], statistical analysis by Hubert [40], machine scheduling by Geoffrion and Graves [30], and several others (see survey [64]).

Given n facilities, n locations, the flow f_{ik} between every pair facilities, the distance d_{jl} between every pair of locations, and a setup c_{ij} cost for allocating facilities to locations. The objective is to find an assignment of all facilities to all locations, such that the total cost of the assignment is minimized. Considering the binary variable x_{ij} to be

$$x_{ij} = \begin{cases} 1 & \text{if facility } i \text{ is assigned to location } j \\ 0 & \text{otherwise.} \end{cases}$$

The QAP problem can be formulated as

$$\begin{aligned} \textbf{(QAP-P)} \min \quad & \sum_{i \neq k, j \neq l} f_{ik} d_{jl} x_{ij} x_{kl} + 2 \sum_{i,j} c_{ij} x_{ij} & \\ \text{s.t.} \quad & \sum_{i} x_{ij} = 1 & 1 \leq j \leq n \\ & \sum_{j} x_{ij} = 1 & 1 \leq i \leq n \\ & x_{ij} \in \{0, 1\} & 1 \leq i, j \leq n. \end{aligned}$$

It is known that solving QAP problems are among the most difficult discrete optimization problems. In practice, QAP problems with $n \geq 30$ are still considered very hard to solve to optimality. Using a similar approach to Section 3.1.2, we obtain three different relaxations for (QAP-P). Considering the change of variables from $\{0,1\}$ to $\{-1,1\}$, the first relaxation is based on semidefinite programming and second-order cone as shown below:

$$\begin{aligned}
\textbf{(QAP}_{\textbf{ss}}\textbf{)}\ \max\ & \lambda \\
\text{s.t. } q(x) - \lambda = & \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_i r_i(x)(1 - \sum_j x_{ij}) + \sum_j t_j(x)(1 - \sum_i x_{ij}) \\
& + \sum_{i,j \neq l} u_{ijl}(x_{ij}x_{il}) + \sum_{j,i \neq k} v_{ikj}(x_{ij}x_{kj}) + \sum_{i,j} x_{ij} {f_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} \\
& + \sum_{i,j} (1 - x_{ij}) {g_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} + \sum_{i,j} c_{ij}(x_{ij} - x_{ij}^2) \\
& c_{ij}, u_{ijl}, v_{ikj} \in \mathbb{R}, \quad r_i(x), t_j(x) \in \mathbf{R}_1[x] \qquad f_{ij}, g_{ij} \in \mathcal{L}^{n^2}, \qquad S \in \mathcal{S}_+^{n^2+1},
\end{aligned}$$

where $\bar{x} = 2x - 1$ and $q(x) = \sum_{i \neq k, j \neq l} f_{ik} d_{jl} x_{ij} x_{kl} + 2 \sum_{i,j} c_{ij} x_{ij}$. Note that the equality constraints of the form $x_{ij}x_{il} = 0$ and $x_{ij}x_{kj} = 0$ are added to the above formulation for reasons that will become apparent in the next section. The second relaxation is (QAP_{ss^+}) with additional constraints involving products of inequality constraints:

$$\begin{aligned}
\textbf{(QAP}_{\textbf{ss}^+}\textbf{)}\ \max\ & \lambda \\
\text{s.t. } q(x) - \lambda = & \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_i r_i(x)(1 - \sum_j x_{ij}) + \sum_j t_j(x)(1 - \sum_i x_{ij}) \\
& + \sum_{i,j \neq l} u_{ijl}(x_{ij}x_{il}) + \sum_{j,i \neq k} v_{ikj}(x_{ij}x_{kj}) + \sum_{i,j} x_{ij} {f_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} \\
& + \sum_{i,j} (1 - x_{ij}) {g_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} + \sum_{i,j} c_{ij}(x_{ij} - x_{ij}^2) + \sum_{i \leq k, j \leq l} \gamma_{ijkl} x_{ij} x_{kl} \\
& + \sum_{i \leq k, j \leq l} \delta_{ijkl}(1 - x_{ij})(1 - x_{kl}) + \sum_{i \leq k, j \leq l} \zeta_{ijkl}(1 - x_{ij})(x_{kl}) \\
& c_{ij}, u_{ijl}, v_{ikj} \in \mathbb{R}, r_i(x), t_j(x) \in \mathbf{R}_1[x],\ \gamma_{ijkl}, \delta_{ijkl}, \zeta_{ijkl} \in \mathbb{R}^+,\ f_{ij}, g_{ij} \in \mathcal{L}^{n^2},\ S \in \mathcal{S}_+^{n^2+1}.
\end{aligned}$$

The third relaxation is based on second-order cone programming and is given below:

$$\begin{aligned}
\textbf{(QAP}_{\textbf{soc}}\textbf{)}\ \max\ & \lambda \\
\text{s.t. } q(x) - \lambda = & \sum_i r_i(x)(1 - \sum_j x_{ij}) + \sum_j t_j(x)(1 - \sum_i x_{ij}) \\
& + \sum_{i,j \neq l} u_{ijl}(x_{ij}x_{il}) + \sum_{j,i \neq k} v_{ikj}(x_{ij}x_{kj}) + \sum_{i,j} x_{ij} {f_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} \\
& + \sum_{i,j} (1 - x_{ij}) {g_{ij}}^T \begin{pmatrix} \sqrt{n^2} \\ \bar{x} \end{pmatrix} + \sum_{i,j} c_{ij}(x_{ij} - x_{ij}^2) \\
& + \sum_{i \leq k, j \leq l} h^+_{ijkl} x_{ij}x_{kl} + \sum_{i \leq k, j \leq l} h^-_{ijkl}(1 - x_{ij}x_{kl}) \\
c_{ij}, u_{ijl}, v_{ikj} \in \mathbb{R}, \quad & r_i(x), t_j(x) \in \mathbf{R}_1[x], \quad h^+_{ijkl}, h^-_{ijkl} \in \mathbb{R}^+, \quad f_{ij}, g_{ij} \in \mathcal{L}^{n^2}.
\end{aligned}$$

Zhao-Karisch-Rendl-Wolkowicz QAP Relaxation

Semidefinite-based relaxations for the QAP have been known to provide strong bounds. Zhao, Karisch, Rendl, and Wolkowicz [102], Rendl and Sotirov [89], and de Klerk and Sotirov [21] lifted the problem from the vector space $\mathbb{R}^{n \times n}$ to the cone of positive semidefinite matrices of order $\mathcal{S}^{n^2+1}$ and formulated several semidefinite relaxations which give tight bounds for the QAP. The QAP formulation presented by Zhao et al. [102] is:

$$\begin{aligned}
\min_{X \in \Pi}\ & \text{trace } FXDX^T + 2CX^T \\
& Xe = e \\
& X^T e = e \\
& X \in \{0,1\}^{n \times n}.
\end{aligned}$$

where X is an $n \times n$ permutation matrix, F, D, and C are $n \times n$ matrices. Another formulation presented by Zhao et al. in [102]:

$$\begin{aligned}
(\mathbf{QAP_{ZKRW}}) \min_X \ & \text{trace } (FXDX^T + 2CX^T) & \\
& Xe = e & \\
& X^T e = e & \\
& XX^T = I & \\
& X^T X = I & \\
& X_{ij} \geq 0 & 1 \leq i, j \leq n. \\
& X_{ij}^2 - X_{ij} = 0 & 1 \leq i, j \leq n.
\end{aligned}$$

Zhao et al. relaxed the constraints in (QAP_{ZKRW}) via Lagrangian duality and showed that the dual of the Lagrangian dual results in an SDP relaxation for the QAP. Although there are a lot of redundant constraints in (QAP_{ZKRW}), when taking the SDP relaxation these constraints aren't redundant and are helpful in terms of tightening the bound. Recent work by Rendl and Sotirov [89] has considered the application of bundle methods to handle the constraints that appear in the SDP relaxations for QAP.

Let X be a permutation matrix, $x = \text{vec}(X)$ and $c = \text{vec}(C)$. Then the objective function for QAP is

$$\begin{aligned}
q(x) = & \text{ trace } (FXDX^T + 2CX^T) \\
& = x^T (D \otimes F)x + 2c^T x \\
& = \text{trace } xx^T (D \otimes F) + 2c^T x \\
& = \text{trace } QY,
\end{aligned}$$

where

$$Q := \begin{pmatrix} 0 & \text{vec}(C)^T \\ \text{vec}(C) & D \otimes F \end{pmatrix}$$

and

$$Y := \begin{pmatrix} 1 & x^T \\ x & xx^T \end{pmatrix}.$$

Zhao et al. SDP relaxation can be written as:

$$\begin{aligned}
\textbf{(QAP}_{\textbf{ZKRW-R}}\textbf{)}\ \min_{Y}\ & \langle Q, Y\rangle & \\
& Y_{ij,kj} = 0 & \forall i \neq k, j \\
& Y_{ij,il} = 0 & \forall i, j \neq l \\
& \sum_j Y_{ij,ij} = 1 & \forall i \\
& \sum_i Y_{ij,ij} = 1 & \forall j \\
& Y_{ij,ij} - Y_{00,ij} = 0 & \forall i, j \\
& Y \succeq 0. &
\end{aligned}$$

Comparing Relaxations for QAP

We compare Zhao et al. SDP relaxation to our three proposed relaxations. First we re-derive ($\text{QAP}_{\text{ZKRW-R}}$) using polynomial programming as described in Section 2.2

$$\begin{aligned}
\textbf{(QAP-D)}\ \max\ & \lambda \\
& \text{s.t. } q(x) - \lambda \in \mathcal{P}_2(\{0,1\}^{n\times n} \cap D),
\end{aligned}$$

where $D := \{x : \sum_i x_{ij} = 1\, \forall j, \sum_j x_{ij} = 1\, \forall i, x_{ij}x_{kj} = 0\, \forall i \neq k, j, x_{ij}x_{il} = 0\, \forall i, j \neq l\}$. This problem can be relaxed using

$$\begin{aligned}
\mathcal{P}_2(\{0,1\}^{n\times n} \cap D) \supseteq \Psi_2[x] + \sum_i (1 - \sum_j x_{ij})\mathbf{R}_0 + \sum_j (1 - \sum_i x_{ij})\mathbf{R}_0 + \sum_{i,j\neq l} (x_{ij}x_{il})\mathbf{R}_0 \\
+ \sum_{j,i\neq k} (x_{ij}x_{kj})\mathbf{R}_0 + \sum_{ij} x_{ij}(1 - x_{ij})\mathbf{R}_0,
\end{aligned}$$

obtaining

$$\begin{aligned}
&\max \lambda \\
&\text{s.t. } q(x) - \lambda = \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_i r_i (1 - \sum_j x_{ij}) + \sum_j t_j (1 - \sum_i x_{ij}) \\
&\qquad\qquad + \sum_{i,j \neq l} u_{ijl}(x_{ij}x_{il}) + \sum_{j,i \neq k} v_{ikj}(x_{ij}x_{kj}) + \sum_{i,j} c_{ij}(x_{ij} - x_{ij}^2),
\end{aligned}$$

where $S \in \mathcal{S}_+^{n^2+1}$ and $r_i, t_j, u_{ijl}, v_{ikj}, c_{ij} \in \mathbb{R}$.

By equating the coefficients of the monomials of the above problem, we rewrite it as

$$\begin{aligned}
&\textbf{(QAP}_{\textbf{ZKRW-D}}\textbf{)} \max \lambda \\
&\qquad \text{s.t. } \lambda + S_{00,00} + \sum_i r_i + \sum_j t_j = 0 \\
&\qquad\qquad r_i + t_j + c_{ij} + S_{00,ij} + S_{ij,00} = 0 && \forall i, j \\
&\qquad\qquad S_{ij,ij} - c_{ij} = Q_{ij,ij} && \forall i, j \\
&\qquad\qquad u_{ijl}\delta_{i=k,j\neq l} + v_{ikj}\delta_{i\neq k,j=l} + S_{ij,kl} + S_{kl,ij} = Q_{ij,kl} && \forall i \neq k, j \neq l \\
&\qquad\qquad S \succeq 0,
\end{aligned}$$

where $\delta_{i=k,j\neq l}$ equals 1 if $i = k$ and $j \neq l$ and 0 otherwise and $\delta_{i\neq k,j=l}$ equals 1 if $i \neq k$ and $j = l$ and 0 otherwise. The dual of the above problem is a reformulation of ($\text{QAP}_{\text{ZKRW-R}}$).

Theorem 3.2.2. *Let $\mu_{QAP_{ZKRW\text{-}D}}$ and $\mu_{QAP_{SS}}$ be the optimal solution values of ($QAP_{ZKRW\text{-}D}$) and (QAP_{SS}) respectively, then*

$$\mu_{QAP_{ZKRW\text{-}D}} = \mu_{QAP_{ZKRW\text{-}R}} \leq \mu_{QAP_{SS}} \leq z_{QAP}.$$

Proof. Define

$$\begin{aligned}\mathcal{H}_4 =&\Psi_2[x] + \sum_i (1 - \sum_j x_{ij})\mathbf{R}_0 + \sum_j (1 - \sum_i x_{ij})\mathbf{R}_0 + \sum_{i,j\neq l}(x_{ij}x_{il})\mathbf{R}_0 \\ &+ \sum_{j,i\neq k}(x_{ij}x_{kj})\mathbf{R}_0 + \sum_{ij} x_{ij}(1 - x_{ij})\mathbf{R}_0, \\ \mathcal{H}_5 =&\mathcal{H}_4 + \sum_{i,j}(x_{ij})\mathcal{P}_1(\mathcal{B}) + \sum_{ij}(1 - x_{ij})\mathcal{P}_1(\mathcal{B}).\end{aligned}$$

Hence,

$$\mathcal{H}_4 \subseteq \mathcal{H}_5.$$

$\mathcal{H}_4$ corresponds to the approximation of $\mathcal{P}_2(\{0,1\}^{n\times n} \cap D)$ that is equivalent to ($\text{QAP}_{\text{ZKRW-D}}$) and since $\mathbf{R}_0 \subseteq \mathbf{R}_1[x]$, then $\mathcal{H}_5$ corresponds to a weaker representation of (QAP_{SS}). □

Table 3.2 presents the number of variables for the relaxations (QAP_{SS^+}), (QAP_{SS}), ($\text{QAP}_{\text{ZKRW-D}}$), and ($\text{QAP}_{\text{SOC}}$). Notice that the first three relaxations have the same computational complexity. However, the (QAP_{SS^+}) relaxation provides the best bounds as shown in Theorem 3.2.2.

Table 3.2: Problem dimension for various QAP relaxations.

Relaxation	SDP	SOC	Linear Non-negative	Linear Free
(QAP_{SS^+})	1, $(n^2+1)\times(n^2+1)$	$2n^2$, (n^2+1)	-	$n^2 + 2n(n^2+1) + 2n^2(n-1)$
(QAP_{SS})	1, $(n^2+1)\times(n^2+1)$	$2n^2$, (n^2+1)	-	$n^2 + 2n(n^2+1) + 2n^2(n-1)$
($\text{QAP}_{\text{ZKRW-D}}$)	1, $(n^2+1)\times(n^2+1)$	-	-	$n^2 + 2n + 2n^2(n-1)$
(QAP_{SOC})	-	$2n^2$, (n^2+1)	$n^2(n^2-1)$	$n^2 + 2n(n^2+1) + 2n^2(n-1)$

3.2.3 Quadratic Linear Constrained Problems

Without loss of generality, we formulate the binary quadratic linear constrained problem as:

$$\begin{aligned} \textbf{(QLCP)} \max\ & x^T Q x + p^T x \\ \text{s.t. } & a_j^T x \le b_j \qquad \forall j \in \{1, \cdots, m\} \\ & x \in \{-1, 1\}^n. \end{aligned}$$

Specializing the results of Section 3.1.2 to (QLCP), we obtain the following relaxations:

$$\begin{aligned} \textbf{(QLCP}_{\textbf{ss}}\textbf{)} \min\ & \lambda \\ \text{s.t. } & \lambda - q(x) = \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_{j=1}^{m} (b_j - a_j^T x) d_j^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} (1 + x_i) {f_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\ & \qquad + \sum_{i=1}^{n} (1 - x_i) {g_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} c_i (1 - x_i^2) \\ & c_i \in \mathbb{R}, \qquad f_i, g_i, d_j \in \mathcal{L}^n, \qquad S \in \mathcal{S}_+^{n+1}; \end{aligned}$$

(QLCP$_{\mathbf{ss^+}}$) min λ

$$\begin{aligned}
\text{s.t. } \lambda - q(x) = &\begin{pmatrix}1 & x\end{pmatrix} S \begin{pmatrix}1\\ x\end{pmatrix} + \sum_{j=1}^{m}(b_j - a_j^T x) d_j^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} + \sum_{i=1}^{n}(1+x_i) {f_i}^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} \\
&+ \sum_{i=1}^{n}(1-x_i) {g_i}^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} + \sum_{i=1}^{n}\sum_{k=1}^{m} \alpha_{ik}(1+x_i)(b_k - a_k^T x) \\
&+ \sum_{i=1}^{n}\sum_{k=1}^{m} \beta_{ik}(1-x_i)(b_k - a_k^T x) + \sum_{i=1}^{n}\sum_{j=i}^{n} \gamma_{ij}(1+x_i)(1+x_j) \\
&+ \sum_{i=1}^{n}\sum_{j=i}^{n} \delta_{ij}(1-x_i)(1-x_j) + \sum_{i=1}^{n}\sum_{j=1}^{n} \zeta_{ij}(1+x_i)(1-x_j) \\
&+ \sum_{k=1}^{m}\sum_{l=k}^{m} \eta_{kl}(b_k - a_k^T x)(b_l - a_l^T x) + \sum_{i=1}^{n} c_i(1-x_i^2) \\
c_i \in \mathbb{R}, \qquad &\alpha_{ik}, \beta_{ik}, \gamma_{ij}, \delta_{ij}, \zeta_{ij}, \eta_{kl} \in \mathbb{R}^n_+, \qquad f_i, g_i, d_j \in \mathcal{L}^n, \qquad S \in \mathcal{S}^{n+1}_+;
\end{aligned}$$

(QLCP$_{\mathbf{soc}}$) min λ

$$\begin{aligned}
\text{s.t. } \lambda - q(x) = &\sum_{j=1}^{m}(b_j - a_j^T x) d_j^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} + \sum_{i=1}^{n}(1+x_i) {f_i}^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} \\
&+ \sum_{i=1}^{n}(1-x_i) {g_i}^T \begin{pmatrix}\sqrt{n}\\ x\end{pmatrix} + \sum_{i=1}^{n} c_i(1-x_i^2) \\
&+ \sum_{i<j} h_{ij}^+(1+x_i x_j) + \sum_{i<j}^{n} h_{ij}^-(1-x_i x_j) \\
c_i \in \mathbb{R}, \qquad &h_{ij}^+, h_{ij}^- \in \mathbb{R}^+ \qquad f_i, g_i, d_j \in \mathcal{L}^n.
\end{aligned}$$

The Relaxation of Burer and Lovász-Schrijver

Burer [13] presented an SDP-based relaxation for the QLCP where the variables are 0-1. We introduce the following relaxation that is at least as strong as the relaxation presented

by Burer [97]:

$$\textbf{(QLCP}_{\textbf{Burer'}}\textbf{)} \quad \min \lambda$$

$$\text{s.t. } \lambda - q(x) = \begin{pmatrix} 1 & x & s & t \end{pmatrix} (M+N) \begin{pmatrix} 1 \\ x \\ s \\ t \end{pmatrix} + \sum_{i=1}^{n} c_i x_i (1-x_i)$$

$$+ \sum_{i=1}^{n} (1 - x_i - s_i) l_i(x) + \sum_{j=1}^{m} (b_j - a_j^T x - t_j) k_j(x)$$

$$c_i \in \mathbb{R}, \qquad l_i, k_i \in \mathbf{R}_1[x,s,t], \qquad M \in \mathcal{S}_+^{2n+m+1}, \qquad N \in \mathbb{R}_+^{(2n+m+1)\times(2n+m+1)},$$

where m is the number of linear constraints. Further ($\text{QLCP}_{\text{Burer'}}$) is equivalent to:

$$\min \lambda$$

$$\text{s.t. } \lambda - q(x) = \begin{pmatrix} 1 & x & 1-x & b-a^T x \end{pmatrix} (M+N) \begin{pmatrix} 1 \\ x \\ 1-x \\ b-a^T x \end{pmatrix} + \sum_{i=1}^{n} c_i x_i (1-x_i)$$

$$c_i \in \mathbb{R}, \qquad M \in \mathcal{S}_+^{2n+m+1}, \qquad N \in \mathbb{R}_+^{(2n+m+1)\times(2n+m+1)},$$

which can be written as

$$\begin{aligned}
&\min \lambda \\
&\text{s.t. } \lambda - q(x) = \begin{pmatrix} 1 & x \end{pmatrix} M' \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_{i=1}^{n} c_i x_i (1 - x_i) \\
&\qquad + \sum_{i=1}^{n} \sum_{k=1}^{m} \alpha_{ik} x_i (b_k - a_k^T x) + \sum_{i=1}^{n} \sum_{k=1}^{m} \beta_{ik} (1 - x_i)(b_k - a_k^T x) \\
&\qquad + \sum_{i=1}^{n} \sum_{j=i}^{n} \gamma_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{j=i}^{n} \delta_{ij} (1 - x_i)(1 - x_j) \\
&\qquad + \sum_{i=1}^{n} \sum_{j=1}^{n} \zeta_{ij} x_i (1 - x_j) + \sum_{k=1}^{m} \sum_{l=k}^{m} \eta_{kl} (b_k - a_k^T x)(b_l - a_l^T x) \\
&c_i \in \mathbb{R}, \qquad \alpha_{ik}, \beta_{ik}, \gamma_{ij}, \delta_{ij}, \zeta_{ij}, \eta_{kl} \in \mathbb{R}_+, \qquad M' \in \mathcal{S}_+^{n+1}.
\end{aligned}$$

Notice that ($\text{QLCP}_{\text{Burer'}}$) reduces to the N^+ relaxation of Lovász and Schrijver [65] by setting the variables $\gamma_{ij}, \delta_{ij}, \zeta_{ij}$, and η_{kl} to zero. That is, N^+ is equivalent to the following relaxation:

$$\begin{aligned}
&\textbf{(QLCP}_{N^+}\textbf{)} \ \min \lambda \\
&\qquad \text{s.t. } \lambda - q(x) = \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + \sum_{i=1}^{n} c_i x_i (1 - x_i) + \sum_{i=1}^{n} \sum_{k=1}^{m} \alpha_{ik} x_i (b_k - a_k^T x) \\
&\qquad\qquad + \sum_{i=1}^{n} \sum_{k=1}^{m} \beta_{ik} (1 - x_i)(b_k - a_k^T x) \\
&\qquad c_i \in \mathbb{R}, \qquad \alpha_{ik}, \beta_{ik} \in \mathbb{R}_+, \qquad S \in \mathcal{S}_+^{n+1}.
\end{aligned}$$

Comparing Relaxations for QLCP

We can prove the following result:

Theorem 3.2.3. *Let $\mu_{QLCP_{N^+}}$, $\mu_{QLCP_{Burer'}}$, and $\mu_{QLCP_{SS^+}}$ be the optimal solution value of ($QLCP_{N^+}$),($QLCP_{Burer'}$), and ($QLCP_{SS^+}$) respectively, then*

$$\mu_{QLCP_{N^+}} \geq \mu_{QLCP_{Burer'}} \geq \mu_{QLCP_{SS^+}} \geq z_{QLCP}.$$

Proof. Define

$$\begin{aligned}
\mathcal{H}_6 =& \Psi_2[x] + \sum_{i,k}(1+x_i)(b_k - a_k^T x)\Psi_0[x] + \sum_{i,k}(1-x_i)(b_k - a_k^T x)\Psi_0[x] + \sum_i (1-x_i^2)\mathbf{R}_0 \\
\mathcal{H}_7 =& \mathcal{H}_6 + \sum_{i\leq j}(1+x_i)(1+x_j)\Psi_0[x] + \sum_{i\leq j}(1-x_i)(1-x_j)\Psi_0[x] + \sum_{i,j}(1+x_i)(1-x_j)\Psi_0[x] \\
&+ \sum_{k\leq l}(b_k - a_k^T x)(b_l - a_l^T x)\Psi_0[x] \\
\mathcal{H}_8 =& \mathcal{H}_7 + \sum_j (b_j - a_j^T x)\mathcal{P}_1(\mathcal{B}) + \sum_i (1+x_i)\mathcal{P}_1(\mathcal{B}) + \sum_i (1-x_i)\mathcal{P}_1(\mathcal{B}).
\end{aligned}$$

Hence,

$$\mathcal{H}_6 \subseteq \mathcal{H}_7 \subseteq \mathcal{H}_8.$$

After a simple change of variables from $\{-1,1\}$ to $\{0,1\}$, $\mathcal{H}_6$ and $\mathcal{H}_7$ correspond to the representations (QLCP_{N^+}) and ($\text{QLCP}_{\text{Burer'}}$) respectively, while $\mathcal{H}_8$ corresponds to the representation ($\text{QLCP}_{\text{SS}^+}$). □

Table 3.3 lists the number of variables required to formulate the various relaxations for the QLCP problem of Theorem 3.2.3 , in addition to (QLCP_{SS}) and (QLCP_{SOC}) where we have m linear constraints and n binary variables. While ($\text{QLCP}_{\text{Burer'}}$) and ($\text{QLCP}_{\text{SS}^+}$) have the same computational complexity, ($\text{QLCP}_{\text{SS}^+}$) provides the best bounds as shown in Theorem 3.2.3 and confirmed by the computational results of Section 3.3.

Remark 3.2.4. *We are unable to compare theoretically the bounds obtained by ($QLCP_{SS}$), ($QLCP_{N^+}$) and ($QLCP_{Burer'}$). However in our computational experiments in Section 3.3.3, ($QLCP_{SS}$) always provides a strictly better bound than ($QLCP_{N^+}$) while ($QLCP_{Burer'}$) provides a strictly better bound than ($QLCP_{SS}$).*

Table 3.3: Problem dimension for various QLCP relaxations.

Relaxation	SDP	SOC	Linear Non-negative	Linear Free
($\text{QLCP}_{\text{SS}^+}$)	1, $(n+1)\times(n+1)$	$(2n+m)$, $(n+1)$	$2nm+n^2+2\binom{n+1}{2}+\binom{m+1}{2}$	n
($\text{QLCP}_{\text{Burer'}}$)	1, $(n+1)\times(n+1)$	-	$2nm+n^2+2\binom{n+1}{2}+\binom{m+1}{2}$	n
(QLCP_{SS})	1, $(n+1)\times(n+1)$	$(2n+m)$, $(n+1)$	-	n
(QLCP_{N^+})	1, $(n+1)\times(n+1)$	-	$2nm$	n
(QLCP_{SOC})	-	$(2n+m)$, $(n+1)$	$n(n-1)$	n

3.2.4 Quadratic Knapsack Problem

We consider the quadratic knapsack problem (QKP) which is the particular case of QLCP where $m = 1$. The QKP was introduced by Gallo, Hammer, and Simeone [27] and is $\mathcal{NP}$-hard. The QKP can be interpreted as follows: we are given n items with a non-negative weight w_i assigned to item i, and a $(n+1)\times(n+1)$ symmetric matrix Q with real entries. The QKP is the problem of selecting a subset of items so as to maximize the overall profit such that the total weight of the selected items does not exceed a given capacity c. Introducing the binary variable x_i such that

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is selected} \\ -1 & \text{otherwise,} \end{cases}$$

the problem may be formulated as:

$$\begin{aligned} \textbf{(QKP-P)} \max\ & q(x) = \begin{pmatrix} 1 & x \end{pmatrix} Q \begin{pmatrix} 1 \\ x \end{pmatrix} \\ \text{s.t. } & w^T x \le c \\ & x \in \{-1,1\}^n. \end{aligned}$$

The QKP is a generalization of the linear knapsack problem (where the objective function is linear). As in the case of the linear knapsack problem, the QKP often appears as a sub-problem to other complex problems such as the graph partitioning problem described

in Johnson, Mehrotra, and Nemhauser [42]. Since the QKP is a constrained version of the binary quadratic problem, all valid inequalities for the unconstrained BQPP problem are also valid for the QKP and hence they can be used to tighten bounds for this problem. Using the same approach as in Section 3.1.2, we obtain the following relaxations of (QKP-P):

$$\begin{aligned}
\textbf{(QKP}_{\textbf{ss}}\textbf{)}\ \min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + (c - w^T x) d^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} (1 + x_i) {f_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_{i=1}^{n} (1 - x_i) {g_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} c_i (1 - x_i^2) \\
c_i \in \mathbb{R}, \qquad & f_i, g_i, d \in \mathcal{L}^n, \qquad S \in \mathcal{S}_+^{n+1};
\end{aligned}$$

$$\begin{aligned}
\textbf{(QKP}_{\textbf{ss}^+}\textbf{)}\ \min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \begin{pmatrix} 1 & x \end{pmatrix} S \begin{pmatrix} 1 \\ x \end{pmatrix} + (c - w^T x) d^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} (1 + x_i) {f_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_{i=1}^{n} (1 - x_i) {g_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n} \alpha_i (1 + x_i)(c - w^T x) \\
& + \sum_{i=1}^{n} \beta_i (1 - x_i)(c - w^T x) + \sum_{i \le j} \gamma_{ij} (1 + x_j)(1 + x_i) \\
& + \sum_{i \le j} \delta_{ij} (1 - x_j)(1 - x_i) + \sum_{i,j} \zeta_{ij} (1 - x_j)(1 + x_i) \\
& + \sum_{i=1}^{n} c_i (1 - x_i^2) \\
c_i \in \mathbb{R}, \qquad & \alpha_i, \beta_i, \gamma_{ij}, \delta_{ij}, \zeta_{ij} \in \mathbb{R}^+, \qquad f_i, g_i, d \in \mathcal{L}^n, \qquad S \in \mathcal{S}_+^{n+1};
\end{aligned}$$

$$\textbf{(QKP}_{\textbf{SOC}}\textbf{)} \ \min \ \lambda$$

$$\text{s.t. } \lambda - q(x) = (c - w^T x)d^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n}(1+x_i){f_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^{n}(1-x_i){g_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix}$$

$$+ \sum_{i=1}^{n} c_i(1-x_i^2) + \sum_{i<j} h_{ij}^{+}(1+x_i x_j) + \sum_{i<j} h_{ij}^{-}(1-x_i x_j)$$

$$c_i \in \mathbb{R}, \qquad h_{ij}^{+}, h_{ij}^{-} \in \mathbb{R}^{+} \qquad f_i, g_i, d \in \mathcal{L}^n.$$

Helmberg-Rendl-Weismantel QKP Relaxation

Helmberg et al. [36] presented four SDP-based relaxations for the QKP where the discrete set is $\{0,1\}^n$. These relaxations are obtained by considering the semidefinite matrix $X = xx^T$. In particular they studied the relaxation

$$\textbf{(QKP}_{\textbf{HRW4}}\textbf{)} \ \max \ \langle P, X\rangle + \text{cst}$$

$$\text{s.t. } \sum_j w_j X_{ij} - \bar{c} X_{ii} \le 0 \quad 1 \le i \le n$$

$$X - \text{Diag}(X)\text{Diag}(X)^T \succeq 0,$$

where $\bar{c} = \frac{1}{2}(\sum_i w_i - c)$, P is an $n \times n$ matrix with entries $P_{ij} = 4Q_{ij}$ (for $i \neq j$) and $P_{ii} = 4Q_{ii} - 4\sum_j Q_{ij} + 4Q_{0i}$, and $\text{cst} = Q_{00} - 2\sum_i Q_{0i} + \sum_{i,j} Q_{ij}$ are obtained by mapping the variables from $\{-1,1\}$ to $\{0,1\}$. Helmberg et al. [36] showed that the optimal objective value of (QKP_{HRW4}), $\mu_{\text{QKP}_{\text{HRW4}}}$, provides the best bound among the SDP relaxations they provided. Actually, (QKP_{HRW4}) provides the tightest previously known SDP relaxation for the QKP in the literature. We will be using this relaxation for comparison purposes in our computational results. In addition, Helmberg et al. [36] strengthen these proposed relaxations by using cutting planes that are valid for BQPP. To illustrate the quality of these SDP relaxations and of the cutting planes, Helmberg et al. [36] present computational results on instances with up to 61 items.

Comparing Relaxations for QKP

We compare (QKP_{HRW4}) and our proposed relaxation. First we re-derive (QKP_{HRW4}) in a different way by considering the problem

$$\begin{aligned}\textbf{(QKP-D)}\ \min\ & \lambda \\ \text{s.t. } & \lambda - p(x) \in \mathcal{P}_2(\{0,1\}^n \cap \{x : (\bar{c} - w^T x) \geq 0\}),\end{aligned}$$

where $p(x) = \sum_{i,j} P_{ij} x_i x_j + \text{cst}$. This problem can be relaxed using

$$\mathcal{P}_2(\{0,1\}^n \cap \{x : (\bar{c} - w^T x) \geq 0\}) \supseteq \Psi_2[x] + \sum_i x_i(\bar{c} - w^T x)\Psi_0[x] + \sum_i x_i(1 - x_i)\mathbf{R}_0,$$

obtaining

$$\begin{aligned}\min\ & \lambda \\ \text{s.t. } & \lambda - p(x) = \begin{pmatrix}1 & x\end{pmatrix} S \begin{pmatrix}1 \\ x\end{pmatrix} + \sum_i d_i x_i(\bar{c} - w^T x) + \sum_i c_i x_i(1 - x_i),\end{aligned}$$

where $S \in \mathcal{S}_+^{n+1}$, $d_i \in \mathbb{R}_+$, and $c_i \in \mathbb{R}$. By equating the coefficients of the monomials of the above problem, we rewrite it as

$$\begin{aligned}\textbf{(QKP}_{\textbf{HRW4-D}}\textbf{)}\ \min\ & \lambda \\ \text{s.t. } & \lambda - \text{cst} - S_{00} = 0 \\ & c_i + \bar{c} d_i + S_{i0} + S_{0i} = 0 \\ & \frac{d_i w_j + d_j w_i}{2} - S_{ij} + c_i \delta_{i=j} = P_{ij} \qquad 1 \leq i \leq j \leq n \\ & S \succeq 0, \quad d_i \geq 0.\end{aligned}$$

where $\delta_{i=j}$ equals 1 if $i = j$ and 0 otherwise. Taking the dual of ($\text{QKP}_{\text{HRW4-D}}$), we obtain

$$\begin{aligned}
\max\ & \left\langle \bar{P}, \bar{X} \right\rangle \\
\text{s.t. } & \bar{X}_{00} = 1 && (3.8)\\
& \bar{X}_{ii} - \bar{X}_{i0} = 0 \quad 1 \le i \le n && (3.9)\\
& \sum_{j=1}^{n} w_j \bar{X}_{ij} - \bar{c}\bar{X}_{ii} \le 0 \quad 1 \le i \le n && (3.10)\\
& \bar{X} \succeq 0, && (3.11)
\end{aligned}$$

where $\bar{P} = \begin{pmatrix} \text{cst} & 0 \\ 0 & P \end{pmatrix}$. Since $X - \text{Diag}(X)\text{Diag}(X)^T \succeq 0$ is equivalent to

$$\bar{X} = \begin{pmatrix} 1 & \text{Diag}(X)^T \\ \text{Diag}(X) & X \end{pmatrix} \succeq 0,$$

the above problem is a reformulation of (QKP_{HRW4}). Taking $X = I$, X is strictly feasible for (QKP_{HRW4}), therefore Slater's constraint qualification is satisfied for (QKP_{HRW4}). In addition, $X - \text{Diag}(X)\text{Diag}(X)^T \succeq 0$ implies $-\frac{1}{8} \le X_{ij} \le 1$ [36]. As a result, the objective $\langle P, X \rangle$ is bounded by $\sum_{i,j} |P_{ij}|$ and we have strong duality.

Theorem 3.2.5. *Let $\mu_{QKP_{HRW4\text{-}D}}$ and $\mu_{QKP_{SS^+}}$ be the optimal solution values of ($QKP_{HRW4\text{-}D}$) and (QKP_{SS^+}) respectively, then*

$$\mu_{QKP_{HRW4\text{-}D}} = \mu_{QKP_{HRW4}} \ge \mu_{QKP_{SS^+}} \ge z_{QKP}.$$

Proof. Define

$$\begin{aligned}
\mathcal{H}_9 =& \Psi_2[x] + \sum_i (1 + x_i)(c - w^T x)\Psi_0[x] + \sum_i (1 - x_i^2)\mathbf{R}_0 \\
\mathcal{H}_{10} =& \mathcal{H}_9 + \sum_i (1 - x_i)(c - w^T x)\Psi_0[x] + \sum_{i \le j} (1 + x_i)(1 + x_j)\Psi_0[x] + \sum_{i \le j} (1 - x_i)(1 - x_j)\Psi_0[x] \\
& + \sum_{i,j} (1 + x_i)(1 - x_j)\Psi_0[x] + (c - w^T x)\mathcal{P}_1(\mathcal{B}) + \sum_i (1 + x_i)\mathcal{P}_1(\mathcal{B}) + \sum_i (1 - x_i)\mathcal{P}_1(\mathcal{B}).
\end{aligned}$$

Hence,

$$\mathcal{H}_9 \subseteq \mathcal{H}_{10}.$$

After mapping the variables from $\{-1,1\}$ to $\{0,1\}$, $\mathcal{H}_9$ corresponds to the approximation of $\mathcal{P}_2(\{0,1\}^n \cap \{x : (\bar{c} - w^T x) \geq 0\})$ that is equivalent to $(\text{QKP}_{\text{HRW4-D}})$ and $\mathcal{H}_{10}$ corresponds to the representation $(\text{QKP}_{\text{SS}^+})$. □

Table 3.4 presents the number of variables for the relaxations $(\text{QKP}_{\text{HRW4-D}})$, $(\text{QKP}_{\text{SS}^+})$, (QKP_{SS}), and $(\text{QKP}_{\text{SOC}})$. Notice that the first two relaxations have the same computational complexity. However, the $(\text{QKP}_{\text{SS}^+})$ relaxation provides the best bounds as shown in Theorem 3.2.5.

Table 3.4: Problem dimension for various QKP relaxations.

Relaxation	SDP	SOC	Linear Non-negative	Linear Free
$(\text{QKP}_{\text{SS}^+})$	$1,\ (n+1)\times(n+1)$	$(2n+1)\ ,\ (n+1)$	$2n + n^2 + 2\binom{n+1}{2}$	n
(QKP_{SS})	$1,\ (n+1)\times(n+1)$	$(2n+1)\ ,\ (n+1)$	-	n
$(\text{QKP}_{\text{HRW4-D}})$	$1,\ (n+1)\times(n+1)$	-	n	n
$(\text{QKP}_{\text{SOC}})$	-	$(2n+1)\ ,\ (n+1)$	$n(n-1)$	n

Remark 3.2.6. *In some instances, even when using the relaxation (QKP_{SS}), we obtain a strictly better bound than (QKP_{HRW4}) as shown in Section 3.3.4. For those instances (QKP_{SS^+}) is also strictly better than (QKP_{HRW4}).*

3.3 Computational Results

In this section, we present computational results obtained by implementing the proposed relaxations of Section 3.1.2 to the four classes of BQPP problems considered in Section 3.2. We conduct comparisons based on computational time and on the quality of the bounds. The focus is on verifying the efficiency of the proposed SOC relaxations compared to the SOS/SDP-based relaxations. All relaxations were implemented with Matlab 7.9.0 using

APPS (see Chapter 6) for constructing the problems and SeDuMi solver version 1.3 [93] was used to solve the conic problems. The experiments were done on a 1200 MHz Sun Sparc machine and the reported computational time is in cpu seconds.

3.3.1 General BQPPs Computational Results

We compare our proposed relaxations with Lasserre's relaxation of order 1 for solving general binary quadratic problems. We compare the following four relaxations:

($\mathbf{BQPP_{SS^+}}$): the relaxation presented in Section 3.1.2;

($\mathbf{BQPP_{SS}}$): the SOC-SDP-based relaxation presented in Section 3.1.2;

($\mathbf{BQPP_{L1}}$): the relaxation presented in Section 3.2.1;

($\mathbf{BQPP_{SOC}}$): the SOC relaxation presented in Section 3.1.2.

We consider 100 randomly generated instances that vary in size, n, from 10 items up to 70 and density from 20% to 100%. Each instance has n quadratic equality constraints of the form $1-x_i^2=0$ to formulate the binary constraints. In addition each instance has an equal number m of linear inequality constraints and of quadratic inequality constraints, with m varying from 1 to $\frac{n}{2}$. We implemented ($\mathrm{BQPP_{L1}}$) using our code. In Table 3.5, we report the average gap and the average computational time of all four relaxations (the average is computed over 5 instances for each combination of n and m). The gap (in %) is calculated as follows:

$$gap = 100 \times \frac{ub_{\text{relaxation}} - ub_{\text{best}}}{ub_{\text{best}}}\%,$$

where the best upper bound is the one obtained by the ($\mathrm{BQPP_{SS^+}}$) relaxation.

The bound of ($\mathrm{BQPP_{SS^+}}$) is the strongest among the four relaxations, therefore we report the average gaps of ($\mathrm{BQPP_{SS}}$), ($\mathrm{BQPP_{L1}}$), and ($\mathrm{BQPP_{SOC}}$) relative to ($\mathrm{BQPP_{SS^+}}$). Observe that ($\mathrm{BQPP_{SS}}$) provides better gaps than ($\mathrm{BQPP_{L1}}$) and ($\mathrm{BQPP_{SOC}}$) for all instances. To facilitate the comparison of ($\mathrm{BQPP_{L1}}$) and ($\mathrm{BQPP_{SOC}}$), we indicate the lower gap between them in bold. Notice that ($\mathrm{BQPP_{SOC}}$) frequently has better gaps than ($\mathrm{BQPP_{L1}}$).

Table 3.5: Computational results for the BQPP instances. The avg. gaps are with respect to ($BQPP_{SS^+}$).

n	m	($BQPP_{SS^+}$)	($BQPP_{SS}$)		($BQPP_{L1}$)		($BQPP_{SOC}$)	
		T(sec)	Gap	T(sec)	Gap	T(sec)	Gap	T(sec)
10	1	2.03	0.85	1.98	12.50	1.49	**2.40**	1.47
	5	1.99	2.15	1.71	32.40	1.27	**2.63**	1.07
20	1	5.42	0.24	5.36	7.96	4.18	**1.78**	1.81
	5	10.60	2.64	6.37	20.37	5.56	**16.59**	2.12
	10	14.96	4.26	6.66	72.30	5.36	**8.08**	2.42
30	1	22.33	1.09	16.39	**2.36**	10.32	28.11	7.69
	5	35.95	2.84	19.17	**23.88**	14.30	26.21	9.11
	15	73.29	10.74	22.94	**32.92**	17.02	53.21	11.19
40	1	78.18	1.66	56.30	34.89	34.59	**29.97**	33.07
	5	122.37	2.33	67.54	36.38	44.66	**28.18**	37.47
	20	306.31	5.71	88.80	50.60	48.27	**38.60**	44.11
50	1	268.93	0.68	179.74	**5.12**	112.49	15.16	48.72
	5	397.34	3.44	193.86	**17.71**	122.32	39.05	117.75
	25	1245.49	12.27	258.77	94.54	142.29	**43.08**	190.33
60	1	970.00	3.15	626.87	**19.61**	375.24	65.83	94.16
	5	1169.37	3.69	663.09	40.75	397.93	**39.75**	183.34
	30	5637.18	9.42	850.83	58.95	473.50	**52.10**	650.46
70	1	2793.31	0.93	2515.31	**29.44**	1214.23	31.51	165.63
	5	3848.18	2.50	2532.18	53.64	1245.09	**26.98**	549.22
	35	15420.53	14.85	2429.09	47.51	1446.99	**46.99**	1818.69
Avg.	-	-	4.27	-	34.69	-	30.06	-

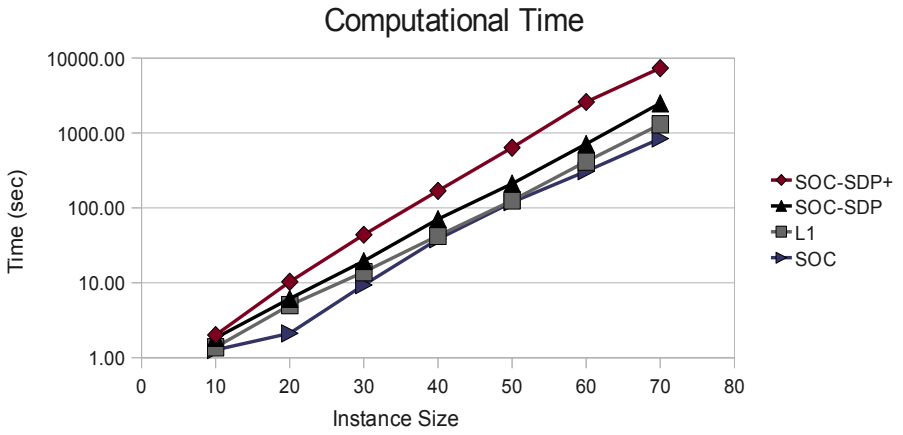

Figure 3.2: Computational time for BQPP (logarithmic scale).

In terms of computational cost, Table 3.5 and Figure 3.2 show that (BQPP_{SOC}) is the cheapest relaxation in most cases. When the number of linear constraints has a value of $\frac{n}{2}$, (BQPP_{L1}) is slightly cheaper, but for those cases the bounds provided by (BQPP_{L1}) are weaker than those provided by (BQPP_{SOC}). One can also observe that for $n \geq 50$, higher computational times correspond to better bounds in most cases. However, this is misleading, and is due to the averaging over 5 instances per line in the table. Looking at the detailed results for the 45 instances considered with $n \geq 50$, (BQPP_{L1}) is better than (BQPP_{SOC}) in terms of both time and bounds for 7 instances, and (BQPP_{SOC}) is better than (BQPP_{L1}) by both measures for 15 instances. Thus, higher times correspond to better bounds for only roughly half of the instances, and no clear conclusions can be drawn.

3.3.2 QAP Computational Results

We compare the performance of our proposed relaxations for the QAP with the relaxation of Zhao et al. [102] presented in Section 3.2.2. The data set used in the computational results is taken form the QAPLIB [14].

The presented computational results are based on the following four types of relaxations for the quadratic assignment problem:

Table 3.6: Computational results for QAP instances.

Instance	Optimal	(QAP_{SS^+})		(QAP_{SS})		($QAP_{ZKRW\text{-}R}$)		(QAP_{SOC})	
		Gap	T(sec)	Gap	T(sec)	Gap	T(sec)	Gap	T(sec)
Nug5	50	0.00	18.01	0.63	10.25	2.10	7.69	0.00	15.54
Nug6	86	0.00	282.55	14.70	22.60	14.99	23.30	0.00	43.12
Nug7	148	0.00	740.78	10.07	97.44	11.04	56.02	0.00	97.46
Nug8	214	0.23	3914.61	15.57	454.48	16.70	404.47	4.89	493.00
Nug12	578	1.73	134303.15	14.80	57644.38	15.75	53200.21	9.46	50651.62
Avg.	-	0.39	-	11.15	-	12.12	-	2.87	-

(QAP_{SS^+}): the relaxation presented in Section 3.2.2;

(QAP_{SS}): the SOC-SDP relaxation presented in Section 3.2.2;

($QAP_{ZKRW\text{-}D}$): the Zhao et al. SDP relaxation presented in Section 3.2.2;

(QAP_{SOC}): the SOC relaxation presented in Section 3.2.2.

Table 3.6 and Figure 3.3 reports results for 5 instances. The size for the instances varies from 25 to 144. For each instance, we report the percentage gap between the lower bound of the relaxation and the optimal value and the solution time in seconds.

From Table 3.6, when n gets large, (QAP_{SOC}) gets more computationally efficient compared to the other three relaxations. Further, (QAP_{SOC}) provides better lower bounds than ($QAP_{ZKRW\text{-}D}$) and (QAP_{SS}) due to the inequalities added for the pure SOC relaxation, these inequalities can be separated to make the SOC relaxation computationally cheaper. As shown in Theorem 3.2.2 and Table 3.6, (QAP_{SS^+}) and (QAP_{SS}) bounds are strictly tighter than the ones provided by ($QAP_{ZKRW\text{-}D}$), even though the bounds for the ($QAP_{ZKRW\text{-}D}$) relaxation are known to be strong [102]. Additionally, (QAP_{SOC}) provides much better bounds than ($QAP_{ZKRW\text{-}D}$) with an average gap of 2.87% compared to 12.12%.

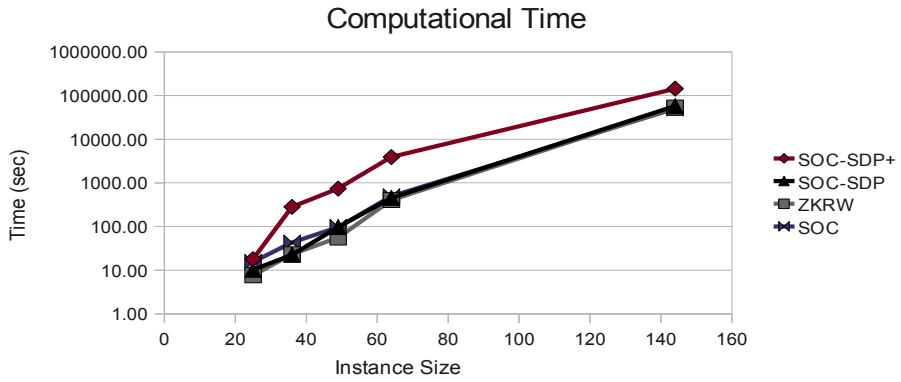

Figure 3.3: Computational time for QAP (logarithmic scale).

3.3.3 QLCP Computational Results

We compare our proposed relaxations of QLCP with the approach proposed by Burer [13] to solve binary quadratic polynomial problems with linear constraints. Table 3.7 reports the average gap (in %) between each relaxation's upper bound and the optimal objective value (known a priori), as well as the average computational time. We compare five relaxations:

($\mathbf{QLCP_{SS^+}}$): the strengthened SDP relaxation presented in Section 3.2.3;

($\mathbf{QLCP_{Burer'}}$): the relaxation presented in Section 3.2.3;

($\mathbf{QLCP_{SS}}$): the SOC-SDP relaxation presented in Section 3.2.3;

($\mathbf{QLCP_{N^+}}$): the Lovász-Schrijver relaxation presented in Section 3.2.3;

($\mathbf{QLCP_{SOC}}$): the SOC relaxation presented in Section 3.2.3.

We consider 732 instances that vary in size from 10 up to 50 items, and with density varying from 1% to 100%. The number of the linear constraints varies from 1 to 25. The data for the instances and their optimal objective values, as well as the upper bounds

and computational time of Burer's specialized implementation, labeled as T1 in Table 3.7, were all provided by Burer [13]. We also implemented Burer's relaxation using our code (as described in Section 3.2.3) and we report the average computational time we obtained for it as T2 in Table 3.7.

Table 3.7: Computational results for the QLCP instances.

n	m	($\text{QLCP}_{\text{SS}^+}$)		($\text{QLCP}_{\text{Burer'}}$)			(QLCP_{SS})		(QLCP_{N^+})		(QLCP_{SOC})	
		Gap	T(sec)	Gap	T1(sec)	T2(sec)	Gap	T(sec)	Gap	T(sec)	Gap	T(sec)
10	1	7.76	1.54	7.77	0.66	1.17	9.00	1.44	11.27	1.11	9.63	0.94
	5	11.28	2.82	11.30	0.72	2.18	16.60	1.77	21.59	1.63	18.73	1.15
20	1	3.72	6.90	3.75	1.24	5.95	4.94	5.19	7.89	4.40	7.84	1.80
	5	8.44	14.23	8.48	2.01	12.56	12.09	6.58	17.58	7.95	17.86	2.31
	10	10.62	20.10	10.64	2.22	17.53	15.70	7.11	21.86	11.63	23.19	2.45
30	1	1.74	20.52	1.80	2.17	17.64	2.32	13.66	3.40	12.00	5.94	8.50
	5	5.75	47.14	5.79	3.35	40.39	8.40	17.96	13.49	22.45	19.43	10.09
	15	10.09	76.75	10.14	4.76	64.28	15.41	20.52	22.66	34.63	29.21	11.84
40	1	1.26	72.38	1.31	3.76	63.50	1.84	41.09	2.91	34.29	8.86	29.63
	5	2.77	150.43	2.81	5.30	128.28	3.77	54.23	5.91	68.33	14.19	37.28
	20	9.94	297.52	10.01	10.21	245.19	16.03	66.26	26.89	120.24	33.11	42.54
50	1	1.07	222.98	1.11	4.96	200.60	1.31	134.42	2.07	104.82	4.79	32.99
	5	2.64	495.88	2.71	8.00	447.68	4.00	161.79	6.69	204.87	17.50	109.17
	25	9.57	1163.82	9.77	18.13	865.09	16.78	199.60	30.64	365.35	34.76	158.80
Avg.	-	6.19	-	6.24	-	-	9.16	-	13.92	-	17.50	-

From Table 3.7, we see that Burer's relaxation is the most efficient in terms of computational time but this is due to the fact that Burer's algorithm is specialized for solving problems of this form. However, in theory, it is an SDP-based relaxation and thus the computational time has a higher order of complexity than the SOC-based relaxation, (QLCP_{SOC}). This can be seen when comparing T2 with the computational time of the (QLCP_{SOC}) relaxation where the latter is on average 4 times more efficient for large n (see Figure 3.4). Among the four SDP-based relaxations, (QLCP_{SS}) is the most computationally efficient as seen from Figure 3.4.

As shown in Theorem 3.2.3 and Table 3.3, ($\text{QLCP}_{\text{SS}^+}$) provides the strongest bounds for

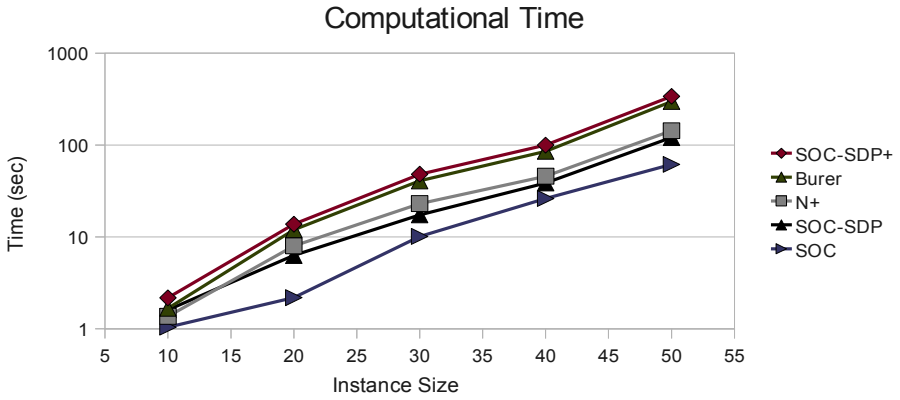

Figure 3.4: Computational time for QLCP (logarithmic scale).

the QLCP relaxation and has the same computational complexity as ($\text{QLCP}_{\text{Burer'}}$). On the other hand, both (QLCP_{N^+}) and (QLCP_{SS}) are semidefinite-based relaxations but with less computational complexity than ($\text{QLCP}_{\text{SS}^+}$) and ($\text{QLCP}_{\text{Burer'}}$). We notice that ($\text{QLCP}_{\text{SS}}$) provides better bounds than (QLCP_{N^+}) for all instances and is more computationally efficient. The average percentage gap for (QLCP_{SS}) is 9.16% while that of (QLCP_{N^+}) is 13.92%. In addition,(QLCP_{SOC}) provides comparable bounds with (QLCP_{N^+}) with an average percentage gap of 17.50% but is computationally the most efficient.

3.3.4 QKP Computational Results

We compare the performance of our proposed relaxations for the QKP with the relaxation of Helmberg et al. [36] presented in Section 3.2.4. We generated test instances using the approach proposed in [82]. The P_{ij} and w_j values are discrete taken from a uniform random distribution in $[1, 100]$ and $[1, 50]$ respectively. The capacity $\bar{c}$ is uniformly distributed in $[50, \sum_{j=1}^{n} w_j]$. The density ρ of the P matrix varies from 10 to 90 %.

The presented computational results are based on the following four types of relaxations for the quadratic knapsack problem:

($\mathbf{QKP_{SS^+}}$): the relaxation presented in Section 3.2.4;

($\mathbf{QKP_{SS}}$): the SOC-SDP relaxation presented in Section 3.2.4;

($\mathbf{QKP_{HRW4}}$): the Helmberg et al. SDP relaxation presented in Section 3.2.4;

($\mathbf{QKP_{SOC}}$): the SOC relaxation presented in Section 3.2.4.

Table 3.8 reports results for 45 instances. These instances vary in size and density. The size varies from 20 to 100 items and the density varies from 10 to 90% with a step size of 20%. For each instance, we report the upper bound and the solution time in seconds.

In terms of computational time, ($\mathrm{QKP_{SOC}}$) is the most computationally efficient for all instances. For example, for the largest instances ($n = 100$) the ($\mathrm{QKP_{SOC}}$) relaxation is on average 23 times faster than the ($\mathrm{QKP_{SS^+}}$), 19 times faster than the ($\mathrm{QKP_{SS}}$) relaxation, and 10 times faster than the ($\mathrm{QKP_{HRW4}}$) relaxation (see Figure 3.5).
Further for all the tested instances, the ($\mathrm{QKP_{SS^+}}$) and ($\mathrm{QKP_{SS}}$) bounds are strictly tighter than the ones provided by ($\mathrm{QKP_{HRW4}}$), even though the bounds for the ($\mathrm{QKP_{HRW4}}$) relaxation are known to be strong [36, 82]. In addition, we report the gap between the bounds of ($\mathrm{QKP_{SS}}$), ($\mathrm{QKP_{HRW4}}$), and ($\mathrm{QKP_{SOC}}$) and the bound of ($\mathrm{QKP_{SS^+}}$). Over all instances, the percentage gap of the ($\mathrm{QKP_{SOC}}$) relaxation with respect to the ($\mathrm{QKP_{HRW4}}$) relaxation ranges from -8% to around 31% with an average of 4.39%, where a negative sign implies that the ($\mathrm{QKP_{SOC}}$) relaxation is better. Notice that ($\mathrm{QKP_{SOC}}$) performs particularly well for instances with high density. In particular, ($\mathrm{QKP_{SOC}}$) obtains better bounds than ($\mathrm{QKP_{HRW4}}$) for all the instances with $d = 90\%$.

3.4 Concluding Remarks

We used polynomial programming approaches to produce tractable relaxations for general binary polynomial optimization problems. These approximations utilize second-order and semidefinite cones over which it is known how to optimize efficiently. We proposed a second-order cone relaxation for general BPP problems with constraints of degree $d-1$ and applied it to several binary quadratic polynomial instances. When compared to SDP-based

Table 3.8: Computational results for QKP instances. The gaps are with respect to (QKP_{SS^+}).

n	ρ	(QKP_{SS^+})		(QKP_{SS})			(QKP_{HRW4})			(QKP_{SOC})		
		UB	T(sec)	UB	Gap	T(sec)	UB	Gap	T(sec)	UB	Gap	T(sec)
20	10	809.00	8.66	811.22	0.27	6.19	814.84	0.72	4.10	811.74	0.34	4.54
20	30	2617.50	4.01	2619.34	0.07	5.68	2623.98	0.25	3.10	2619.48	0.08	1.97
20	50	1120.90	7.52	1137.25	1.46	6.09	1175.07	4.83	4.14	1262.98	12.68	1.42
20	70	2340.94	5.53	2356.25	0.65	5.51	2397.20	2.40	4.14	2540.40	8.52	1.69
20	90	6082.09	5.61	6083.70	0.03	5.72	6086.12	0.07	3.85	6083.80	0.03	1.75
30	10	1011.34	20.83	1022.20	1.07	18.24	1044.39	3.27	9.31	1129.01	11.63	6.91
30	30	3451.65	24.15	3470.97	0.56	16.37	3511.30	1.73	9.77	3939.00	14.12	6.01
30	50	8116.24	17.14	8125.16	0.11	19.48	8142.11	0.32	12.25	8127.76	0.14	9.83
30	70	8042.65	15.01	8047.03	0.05	18.20	8073.14	0.38	10.78	8108.38	0.82	6.94
30	90	5114.00	15.96	5127.57	0.27	15.78	5150.78	0.72	9.35	5136.34	0.44	8.81
40	10	3845.33	51.43	3853.49	0.21	55.43	3864.51	0.50	38.72	3875.12	0.77	32.86
40	30	11807.67	40.09	11809.44	0.02	59.27	11828.42	0.18	32.40	11811.54	0.03	34.71
40	50	4298.30	93.95	4309.76	0.27	69.20	4365.56	1.56	34.06	5161.31	20.08	26.29
40	70	17415.63	76.24	17424.10	0.05	60.41	17446.14	0.18	35.92	17447.01	0.18	31.18
40	90	25599.30	64.70	25612.48	0.05	59.17	25630.04	0.12	39.15	25615.00	0.06	36.12
50	10	2316.83	274.29	2353.89	1.60	158.24	2412.48	4.13	96.62	2846.05	22.84	44.32
50	30	11414.34	186.91	11433.16	0.16	188.84	11485.59	0.62	114.12	12050.94	5.58	64.22
50	50	23823.61	270.40	23846.12	0.09	181.09	23863.04	0.17	116.33	23850.99	0.11	27.62
50	70	32567.32	133.12	32571.10	0.01	213.29	32626.49	0.18	113.25	32575.12	0.02	26.29
50	90	17658.96	167.46	17671.03	0.07	168.55	17682.63	0.13	91.98	17672.78	0.08	23.05
60	10	7173.33	705.30	7188.68	0.21	673.37	7215.96	0.59	394.70	7410.08	3.30	138.25
60	30	26403.91	552.28	26496.51	0.35	644.20	26530.82	0.48	312.36	26502.66	0.37	79.46
60	50	13853.47	726.82	13871.42	0.13	682.12	13895.51	0.30	355.53	14396.64	3.92	78.18
60	70	56556.58	663.95	56561.20	0.01	797.11	56583.48	0.05	343.42	56561.20	0.01	58.54
60	90	62009.00	357.10	62009.00	0.00	478.40	62015.61	0.01	391.59	62009.00	0.00	38.21
70	10	3961.79	2969.84	4036.66	1.89	1689.45	4109.61	3.73	954.02	5104.22	28.84	230.87
70	30	20191.73	2698.05	20208.57	0.08	2262.87	20275.13	0.41	1237.78	21826.79	8.10	296.70
70	50	45493.48	2760.52	45507.07	0.03	2407.57	45573.21	0.18	1224.95	45752.77	0.57	154.61
70	70	1621.19	2900.58	1631.57	0.64	2308.23	1882.75	16.13	1081.38	1737.92	7.20	143.12
70	90	32850.56	1777.27	32857.31	0.02	2574.93	32913.98	0.19	1157.09	32876.13	0.08	102.06
80	10	13062.74	4407.65	13074.13	0.09	5008.22	13118.78	0.43	2584.26	13506.53	3.40	564.75
80	30	1480.00	3327.65	1480.00	0.00	4388.67	1537.29	3.87	2143.41	1532.02	3.51	264.94
80	50	23126.43	6694.43	23141.40	0.06	4650.24	23220.33	0.41	2494.70	25240.44	9.14	365.01
80	70	58613.63	5422.86	58621.35	0.01	5419.69	58649.30	0.06	2979.25	59322.02	1.21	270.39
80	90	112167.40	4178.58	112184.20	0.01	5052.10	112202.99	0.03	2958.44	112184.53	0.02	185.92
90	10	6189.28	15610.86	6311.21	1.97	7057.15	6447.72	4.18	4818.87	8500.89	37.35	517.90
90	30	30656.56	16455.66	30710.62	0.18	9398.88	30829.68	0.56	5587.52	36535.46	19.18	740.54
90	50	81336.10	10319.41	81344.17	0.01	12623.02	81393.43	0.07	6233.23	81385.48	0.06	426.81
90	70	8004.38	12082.24	8014.95	0.13	11942.53	8312.97	3.86	4292.94	8297.26	3.66	458.36
90	90	55262.87	11603.07	55285.71	0.04	8883.04	55305.54	0.08	5640.32	55291.14	0.05	295.34
100	10	23941.78	23975.63	23951.45	0.04	18831.11	23977.05	0.15	9883.59	24021.78	0.33	1867.44
100	30	40216.48	31499.01	40257.87	0.10	17673.77	40370.14	0.38	9832.83	45597.97	13.38	973.49
100	50	11707.00	27958.75	11737.62	0.26	18867.58	11879.03	1.47	8553.04	13937.02	19.05	1308.24
100	70	122205.33	20428.73	122215.14	0.01	24684.33	122305.61	0.08	9482.50	122476.61	0.22	431.02
100	90	63378.00	12182.11	63378.00	0.00	14881.31	63411.61	0.05	10280.16	63378.00	0.00	484.25
Avg.		-	-	-	0.30	-	-	1.34	-	-	5.81	-

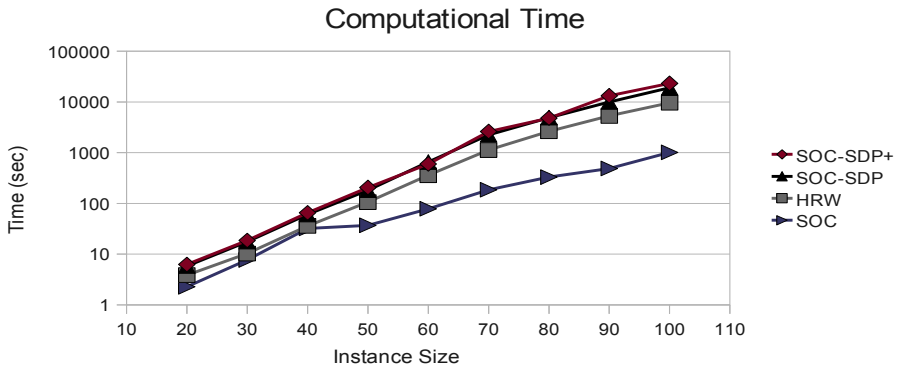

Figure 3.5: Computational time for QKP (logarithmic scale).

relaxations, these SOC-based relaxations of BQPP are significantly more computationally efficient with only a small degradation of bounds.

The main contribution of this section is the use of SOC for general binary polynomial programs. The SOC relaxations show strong potential, both in terms of bounds and of computational time, to be used in an exact algorithm scheme to find optimal solutions for large instances of such problems in a reasonable time. In the following sections, we use these SOC approximations in particular when adding valid inequalities to BPPs. Additionally, we investigate the use of SOC-based relaxations with additional valid inequalities in a branch-and-bound framework.

Chapter 4

Dynamic Inequality Generation Scheme

In this chapter, we propose a dynamic inequality generation scheme (DIGS) for general polynomial programs. The key idea of DIGS is to bound the complexity (degree) of the non-negative certificates, avoiding the exponential growth of relaxation size as in the hierarchies presented in Section 2.2.4. Instead, our approach makes use of information from the objective function to construct improved approximations of the polynomial program, by dynamically generating polynomial inequalities that are valid on the feasible region. Iteratively the new valid inequalities are used to construct new non-negative certificates, obtaining improving approximations to the polynomial program without growing the degree of the certificates involved. Depending on the original problem and the type of relaxation used, the iterative procedure solves a sequence of linear, second-order cone, or semidefinite problems. For illustrative purposes, we focus on semidefinite relaxations in this chapter and thus obtain a sequence of semidefinite problems. In the next chapter, we use SOC relaxations from Section 3.1.2 and obtain a sequence of second-order cone problems. The general scheme is presented in Section 4.1.

Further, we propose a dynamic inequality generation scheme specialized to polynomial programs where (some) all of the variables are binary. Our method for the binary case can be seen as a generalization, from binary linear programs to BPP, of the lift-and-

project methods of Balas, Ceria, and Cornuéjols [6], Sherali and Adams [90], and Lovász and Schrijver [65]. Convergence to the global optimal solution is proven for a family of problems including binary quadratic programming with linear constraints. The specialized scheme and the convergence results are presented in Section 4.2.

To evaluate the proposed approach, we apply it computationally to general polynomial programs and binary polynomial programs, comparing the results to Lasserre's approach [54, 53] for the general and binary cases and to the lift-and-project method of Balas, Ceria, and Cornuéjols [6] for the binary linear case. The computational results are presented in Section 4.3.

4.1 General Case

In this section, we propose a scheme to dynamically generate valid polynomial inequalities for general PPs. Given any hierarchy, instead of growing r, which increases the size of the problem exponentially, we fix r to a small value (mainly to d, the degree of (PP-P)) and improve the approximation $\mathbb{K}^r_G$ of $\mathcal{P}_d(S)$ by growing the set G, e.g., by adding valid polynomial inequalities to the description of S. Our approach makes use of information from the objective function to dynamically generate polynomial inequalities that are valid on the feasible region. These valid inequalities are then used to construct new non-negative certificates, obtaining better approximations to (PP-P). We can use any approximation $\mathcal{K}$ of $\mathcal{P}_d(S)$, however for ease of exposition we work with the hierarchy $\mathbb{K}^r_G$ presented in Section 2.2.5.

Notice that if $p(x) \in \mathbb{K}^r_G$, then $\mathbb{K}^r_G = \mathbb{K}^r_{G\cup\{p\}}$, and thus we do not obtain better non-negative certificates. On the other hand, if $p(x) \geq 0$ is a valid inequality outside $\mathbb{K}^r_G$ then the approximation $\mathbb{K}^r_{G\cup\{p\}}$ to $\mathcal{P}_d(S)$ is improved and thus we may obtain an improved bound for (PP-P).

Lemma 4.1.1. *Let $r \geq d$ and $p(x) \in \mathcal{P}_d(S) \setminus \mathbb{K}^r_G$.*

$$\mathbb{K}^r_G \subsetneq \mathbb{K}^r_{G\cup\{p\}} \subseteq \mathcal{P}_d(S) \text{ and thus } \mu^r_G \geq \mu^r_{G\cup\{p\}}.$$

Using Lemma 4.1.1, given a finite $G \subseteq \mathcal{P}_d(S)$ and $d \geq 1$, the following procedure is an idealized generic iterative scheme to find improving approximations to $\mathcal{P}_d(S)$:

Algorithm 1: Iterative Scheme for PP

Input: G

Output: A sequence $(G_i)_{\{i=0,1,\ldots\}}$ such that $\mathbb{K}^d_{G_i} \uparrow \mathcal{P}_d(S)$

1 **set**: $i = 0$, $G_0 = G$, and Stop=0
2 **while** *Stop=0* **do**
3 Given G_i, find $p_i(x) \in \mathcal{P}_d(S) \setminus \mathbb{K}^d_{G_i}$;
4 **if** $p_i(x)$ *does not exist* **then**
5 Stop=1;
6 **return** G_i;
7 **else**
8 **set**: $G_{i+1} := G_i \cup \{p_i\}$;
9 **set**: $i := i + 1$;
10 **end**
11 **end**

Remark: We can not solve Algorithm 1 due to Step 3 of the algorithm. In Section 4.1.1, we discuss how to tackle this problem.

Notice that $\mathbb{K}^d_{G_0} \subsetneq \mathbb{K}^d_{G_1} \subsetneq \cdots \subsetneq \mathbb{K}^d_{G_s} \subsetneq \mathbb{K}^d_{G_{s+1}} \cdots \subseteq \mathcal{P}_d(S)$. Defining the sequence of problems:

$$\begin{aligned} \textbf{(PP-M}_s\textbf{)}\ \mu_s = \inf_{\lambda}\ & \lambda \\ \text{s.t. } & \lambda - f(x) \in \mathbb{K}^d_{G_s}, \end{aligned} \tag{4.1}$$

from Lemma 4.1.1, it follows that:

Lemma 4.1.2.

$$\mu_0 \geq \cdots \geq \mu_s \geq \mu_{s+1} \geq \cdots \geq z_{PP}.$$

For any $s \geq 0$, the optimization problem (PP-M$_s$) can be written as:

$$\begin{aligned}
\mu_s = \inf_{\lambda,\sigma_i(x),\eta_i} \ & \lambda \\
\text{s.t. } & \lambda - f(x) = \sigma_0(x) + \sum_{i=1}^{m} \sigma_i(x) g_i(x) + \sum_{i=1}^{s} \eta_i p_i(x) \qquad (4.2) \\
& \sigma_0(x) \in \Psi_d[x] \\
& \sigma_i(x) \in \Psi_{d-\deg(g_i)}[x] \qquad i = 1, \ldots, m \\
& \eta_i \in \mathbb{R}^+ \qquad i = 1, \ldots, s.
\end{aligned}$$

For a problem with m inequality constraints and performing s iterations of the iterative scheme, the size of (4.2) is:

- One psd matrix of dimension $\binom{n+d}{d}$;
- m psd matrices, each of dimension $\binom{n+d-\deg(g_i)}{d-\deg(g_i)}$ for $i \in \{1, \ldots, m\}$;
- s non-negative variables;
- $\binom{n+d}{d}$ constraints.

By fixing the degree of the relaxation, we are able to handle larger problem sizes. Problem (2.20) presented in the Section 2.2.4, has $m+1$ psd matrices of size $O(n^r)$ and $O(n^r)$ constraints while (4.2) has $m+1$ psd matrices of size $O(n^d)$ plus $O(s)$ non-negative variables and $O(n^d)$ constraints. When $d \ll r$, the size of the positive semidefinite matrices and the number of constraints are significantly lower in (4.2) compared to (2.20). This reduction in the order of magnitude of the number of variables and the number of constraints is a key factor in reducing the total computational time required.

4.1.1 Dynamic Inequality Generation Scheme (DIGS)

Now we look closely at how to generate a valid inequality, that is at how to execute Step 3 of the generic iterative scheme given in Algorithm 1:

$$\texttt{GIVEN } G_i\texttt{, FIND } p(x) \in \mathcal{P}_d(S) \setminus \mathbb{K}^d_{G_i}.$$

We need to tackle two problems, first how to generate $p(x) \in \mathcal{P}_d(S)$ and second how to ensure $p(x) \notin \mathbb{K}_G^d$.

To tackle the first issue, we approximate $\mathcal{P}_d(S)$ using $\mathcal{K} \subseteq \mathcal{P}_d(S)$ such that $\mathcal{K} \backslash \mathbb{K}_G^d \neq \emptyset$. Notice that $\mathcal{K} = \mathbb{K}_G^r$ where $r \geq d+1$ would work. So one could use approximations of degree $d+1$ and have $p(x) \in \mathbb{K}_G^{d+1} \setminus \mathbb{K}_G^d$, however when d is even, this might result in a very slow improvement in the bound of (PP-P) as $\Psi_{d+1}[x] = \Psi_d[x]$. Thus, for even d, we use $\mathbb{K}^{d+2}$ and for odd d we can use $\mathbb{K}_G^{d+1}$. For the rest of the section, we consider $\mathbb{K}_G^{d+2}$ for ease of notation.

To solve the second issue, i.e, to ensure $p(x) \notin \mathbb{K}_G^d$, we use the optimal dual solution of (4.2), denoted by Y. In this way, the relaxation (4.1) and its SDP dual correspond to the conic primal-dual pair:

$$\begin{array}{ll} \inf_\lambda & \lambda \\ \text{s.t.} & \lambda - f(x) \in \mathbb{K}_G^d \end{array} \qquad\qquad \begin{array}{ll} \sup_Y & \langle f, Y \rangle \\ \text{s.t.} & \langle 1, Y \rangle = 1 \\ & Y \in (\mathbb{K}_G^d)^*. \end{array}$$

From the definition of dual cone $(\mathbb{K}_G^d)^*$, we have the following lemma,

Lemma 4.1.3. *Let Y be a dual solution of (4.1). For all $p(x) \in \mathbb{K}_G^d$, $\langle p, Y \rangle \geq 0$.*

Thus to generate $p(x) \in \mathcal{P}_d(S) \setminus \mathbb{K}_G^d$, we need to find $p(x) \in \mathbb{K}_G^{d+2}$ such that $\langle p, Y \rangle < 0$. This can be done by solving the following semidefinite problem. We refer to this problem as the polynomial generation subproblem:

$$\begin{array}{lll} \textbf{(PP-Sub)} & \inf\limits_p & \langle p, Y \rangle \\ & \text{s.t.} & p(x) \in \mathbb{K}_G^{d+2} \\ & & \| \, p \, \| \leq 1. \end{array} \tag{4.3}$$

The normalization constraint is added since $p(x)$ and $cp(x)$ are equivalent inequalities for any $c > 0$, i.e., the inequality can be scaled arbitrarily. Moreover, without the normalization constraint, (PP-Sub) is unbounded. There are several options for choosing the norm $\| \cdot \|$. In the following section, we discuss different ways to normalize $p(x)$.

Normalization

There are several options for choosing the norm $\| \, . \, \|$. The three normalizations proposed are aimed at guaranteeing the existence of a finite optimum for subproblem (4.3). Consider the three different norms for a given $p(x)$ denoted by $\| \, p \, \|_1, \| \, p \, \|_2$, and $\| \, p \, \|_\alpha$ defined respectively as follows:

1. The first norm is the ℓ_1 norm of $p(x)$, that is $\| \, p \, \|_1 = \sum_{|\alpha|=d} |p_\alpha|$.
2. The second norm is the ℓ_2 norm of $p(x)$, $\| \, p \, \|_2 = \sum_{|\alpha|=d} p_\alpha^2$. This gives a second-order cone condition on the coefficients of the polynomial $p(x)$.
3. The third norm is the polynomial norm. Recall the form of the polynomial $p(x)$:

$$p(x) = \sum_{|\alpha|=d} p_\alpha x^\alpha = \sum_{|\alpha|=d} c(\alpha) \bar{p}_\alpha x^\alpha,$$

where $c(\alpha) := \frac{d!}{\alpha_1! \ldots \alpha_n!}$. The polynomial norm of p is: $\| \, p \, \|_\alpha = \sum_{|\alpha|=d} \dfrac{|p_\alpha|}{c(\alpha)}$.

Normalization Approach 1: The first normalization approach takes one of the norms discussed above and sets it to 1. As a result we obtain the following subproblem:

$$\begin{aligned} &\min \ \langle p, Y \rangle \\ &\text{s.t. } p(x) \in \mathbb{K}_G^d \\ &\quad \| \, p \, \| \leq 1 \end{aligned}$$

Normalization Approach 2: The second normalization is the one that maximizes the norm between Y, the optimal dual solution of (4.2), and the set $\{\mathcal{M}_d(x) : p(x) = 0\}$. The polynomial $p(x)$ can be written in the form of $p = \begin{bmatrix} p_0 \\ p_\alpha \end{bmatrix}$ and $Y = \begin{bmatrix} Y_0 \\ Y_\alpha \end{bmatrix}$ where $Y_0 = 1$ and $|\alpha| > 0$. Let $\hat{Y} = \begin{bmatrix} 1 \\ \hat{Y}_\alpha \end{bmatrix}$ be the projection of Y on the hyperplane corresponding to

$\{\mathcal{M}_d(x) : p(x) = 0\}$ and consider $\hat{Y} = Y_\alpha + tp_\alpha$ where t is a direction, then

$$\| Y - \hat{Y} \| = \| tp_\alpha \| = | t | \| p_\alpha \|$$

Since $\hat{Y}$ is on the hyperplane defining $\{\mathcal{M}(x) : p(x) = 0\}$ then it satisfies the following

$$p_\alpha^T \hat{Y}_\alpha + p_0 = 0.$$

Substituting $\hat{Y}$ by $Y_\alpha + tp_\alpha$ we obtain:

$$p_\alpha^T Y_\alpha + tp_\alpha^T p_\alpha + p_0 = 0.$$

We maximize the following

$$\begin{aligned} \max \ & | t | \| p_\alpha \| \\ \text{s.t. } & p(x) \in \mathbb{K}_G^d \\ & p_\alpha^T Y_\alpha + tp_\alpha^T p_\alpha + p_0 = 0 \\ & \| p_\alpha \| \leq 1, \end{aligned}$$

which is equivalent to

$$\begin{aligned} \max \ & | t | \\ \text{s.t. } & p(x) \in \mathbb{K}_G^d \\ & t = -p_\alpha^T Y_\alpha - p_0 \\ & \| p_\alpha \| \leq 1. \end{aligned}$$

Maximizing the absolute value of t is equivalent to maximizing the absolute value of $p_\alpha^T Y_\alpha + p_0$ which is a negative value equivalent to $\langle p, Y \rangle$. Hence the problem can be written as

$$\begin{aligned} \min \ & \langle p, Y \rangle \\ \text{s.t. } & p(x) \in \mathbb{K}_G^d \\ & \| p_\alpha \| \leq 1 \end{aligned}$$

where $|\alpha| > 0$ and $\|\cdot\|$ is any of the norms previously discussed.

The normalization used in the following sections is the ℓ_2 norm combined with Approach 2, i.e., the one that maximizes the ℓ_2 distance between Y and the set $\{\mathcal{M}_d(x) : p(x) = 0\}$. For a polynomial $p(x)$, this normalization constraint is equivalent to $\sum_{|\alpha|>0} p_\alpha^2 \leq 1$.

DIGS Algorithm

Using the ingredients discussed in this section, the dynamic inequality generation scheme can be summarized as follows:

Algorithm 2: DIGS for General Polynomial Programs

Input: $G = \{g_i(x) : i = 1, \ldots, m\}$, $f(x)$

Output: μ

set: $i = 0$, $G_0 = G$, and Stop=0

while *Stop=0* **do**

Let Y^i be a optimal dual solution of:

$$\begin{aligned} \mu = \min_{\lambda} \ & \lambda \\ \text{s.t. } & \lambda - f(x) \in \mathbb{K}^d_{G_i} \end{aligned}$$

;

Let $p_i(x) \in \mathbf{R}_d[x]$ be a solution to:

$$\begin{aligned} \min_{p} \ & \langle p, Y^i \rangle \\ \text{s.t. } & p(x) \in \mathbb{K}^{d+2}_{G_i} \\ & \| \, p \, \| \leq 1 \end{aligned}$$

;

if $\langle p, Y^i \rangle > -10^{-3}$ **then**

Stop=1;

else

set: $G_{i+1} := G_i \cup \{p_i\}$;

set: $i := i + 1$;

end

end

To show how the method works, we use an illustrative example. Consider the following non-convex quadratic problem:

Example 4.1.4. *Problem 3.5 in [25]:*

$$\begin{aligned} \min_{x} \ & -2x_1 + x_2 - x_3 \\ s.t. \ & 24 - 20x_1 + 9x_2 - 13x_3 + 4x_1^2 - 4x_1x_2 + 4x_1x_3 + 2x_2^2 - 2x_2x_3 + 2x_3^2 \geq 0 \\ & x_1 + x_2 + x_3 \leq 4 \\ & 3x_2 + x_3 \leq 6 \\ & 0 \leq x_1 \leq 2, \qquad 0 \leq x_2, \qquad 0 \leq x_3 \leq 3. \end{aligned} \tag{4.4}$$

Let $G_0 = \{1, 24-20x_1+9x_2-13x_3+4x_1^2-4x_1x_2+4x_1x_3+2x_2^2-2x_2x_3+2x_3^2, 4-x_1-x_2-x_3, 6-3x_2-x_3, x_1, x_2, x_3, 2-x_1, 3-x_3\}$. Solving the master problem (4.2) with $s = 0$, we obtain μ_0=-6.000 which is a lower bound on (4.4) with the optimal dual solution

$$Y_0 = \begin{bmatrix} 1.0 & 2.0 & -0.0 & 2.0 & 5.2 & 0.5 & 3.4 & 1.7 & 0.2 & 5.6 \end{bmatrix}.$$

To obtain a valid inequality for (4.4), we solve subproblem (4.3) with $Y = Y_0$, $G = G_0$, and $d = 2$. We obtain

$$\begin{aligned} p_0(x) =& 0.4693 + 0.3959x_1 + 0.4802x_2 + 0.3553x_3 - 0.3148x_1^2 + 0.1455x_1x_2 \\ & - 0.0530x_1x_3 - 0.3569x_2^2 + 0.0221x_2x_3 - 0.1231x_3^2. \end{aligned}$$

Define $G_1 = G_0 \cup \{p_0\}$. Solving the master problem (4.2) with $s = 1$, we improve the lower bound to $\mu_1 = -5.8746$. Solving the subproblem with $Y = Y_1$, $G = G_1$, and $d = 2$, we obtain

$$\begin{aligned} p_1(x) =& 0.7678 + 0.1591x_1 + 0.2569x_2 + 0.1234x_3 - 0.2707x_1^2 + 0.1081x_1x_2 \\ & - 0.4243x_1x_3 + 0.1007x_2^2 + 0.10431x_2x_3 + 0.1341x_3^2 \end{aligned}$$

Solving the relaxation with $G_2 = G_1 \cup \{p_1\}$ we obtain an objective value of $\mu_2 = -5.6672$.

We perform this approach iteratively obtaining a tighter approximation of the original problem and improving the bound. After adding 25 inequalities we obtain an objective value of -4.0047 which is very close to the optimal value of -4.00 as seen in Figure 4.1 and Table 4.1 the total time is 20.18 seconds.

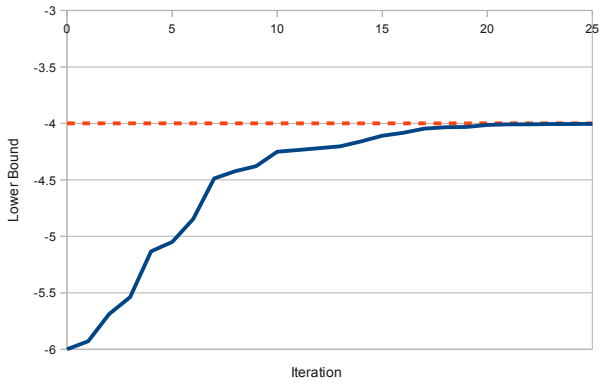

Figure 4.1: DIGS lower bounds for Example 4.1.4. The dotted line is the optimal objective value.

Table 4.1: DIGS Results for Example 4.1.4.

Iteration	0	1	5	10	15	20	25
objective value	-6.0000	-5.8746	-5.0497	-4.2508	-4.1092	-4.0140	-4.0047
Accumulated time (sec)	0.11	0.54	2.95	6.41	10.85	16.35	20.18
Master Problem							
psd matrices	4×4(1)	4×4(1)	4×4(1)	4×4(1)	4×4(1)	4×4(1)	4×4(1)
non-negative vars	8	9	13	18	23	28	33
total # of vars	18	19	23	28	33	38	43
# of constraints	10	10	10	10	10	10	10
time (sec)	0.11	0.19	0.24	0.37	0.38	0.53	0.62
Subproblem							
psd matrices	4×4(8), 10×10(1)	4×4(9), 10×10(1)	4×4(13), 10×10(1)	4×4(18), 10×10(1)	4×4(23), 10×10(1)	4×4(28), 10×10(1)	
non-negative vars	0	0	0	0	0	0	
free vars	10	10	10	10	10	10	
total # of vars	145	185	195	245	295	345	
# of constraints	35	35	35	35	35	35	
time (sec)	0.24	0.39	0.41	0.50	0.60	0.63	

In Table 4.2 can be seen that using Lasserre's approach, the optimal solution is obtained by solving a problem over $\Gamma^8_{G_0}$. However, using DIGS we are able to obtain a bound close to the optimal value while using a cheaper low degree problem ($r = 2$ for the master problem and $r = 4$ for the subproblem).

Table 4.2: Lasserre's Hierarchy for Example 4.1.4.

r	2	4	6	8
objective value	-6.0000	-5.6923	-4.0685	-4.0000
psd matrices	4×4(1)	4×4(8), 10×10(1)	10×10(8), 20×20(1)	20×20(8), 35×35(1)
non-negative vars	8	0	0	0
total # of vars	18	135	650	2310
# of constraints	10	35	84	165
time (sec)	0.11	0.62	8.04	23.55

From Figure 4.2, it is clear that the size of the problem grows exponentially when increasing r in the Lasserre's hierarchy. For $r = 2$, the linear system of equations is of size 10×25 which grows to 165×4426 for $r = 8$.

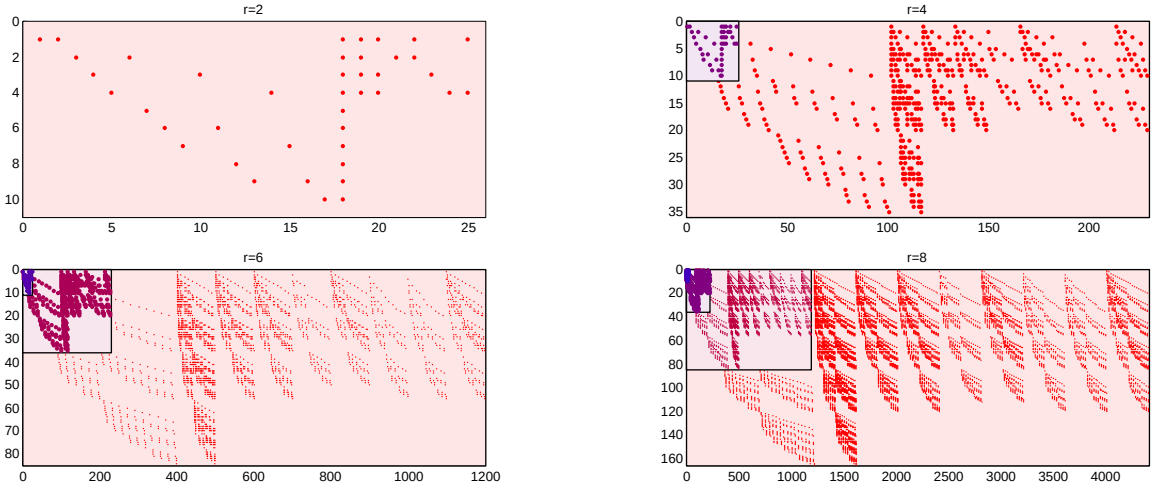

Figure 4.2: Size of the linear system of equations for Lasserre's hierarchy: Example 4.1.4.

More examples of the DIGS method are presented in Section 4.3.

4.2 Binary Case

In this section we specialize the results presented in Section 4.1 to the binary case. Algebraic geometry representation techniques are used to obtain a scheme particular for the binary case that iteratively improves the bound converging to the optimal objective value of the original binary polynomial program. Using the approach proposed in [80] and [103], we obtain a computationally cheaper subproblem for the binary case. Further, we present convergence results for some important cases. We prove that the resulting iterative scheme converges to the global optimal solution of the binary polynomial program when starting from the exact representation of the domain set excluding the binary constraints. As such a representation is not tractable in general, we show that a suitable tractable initial approx-

imation ensures convergence for binary polynomial programs with a quadratic objective and linear constraints.

4.2.1 Specializing the Dynamic Inequality Generation Scheme

In this section, we consider the domain S in (2.15) to be of the form $S = D \cap H$ where $D = \{x : g_i(x) \geq 0,\, i = 1, \ldots, m\} \subseteq [-1, 1]^n$ and $H = \{-1, 1\}^n$. Problem (2.15) becomes

$$\textbf{(BPP-P)}\ z_{BPP} = \begin{array}{l}\max\ f(x) \\ \text{s.t. } x \in D \cap H.\end{array} \qquad \textbf{(BPP-D)}\ z_{BPP} = \begin{array}{l}\min\ \lambda \\ \text{s.t. } \lambda - f(x) \in \mathcal{P}_d(D \cap H).\end{array}$$

To solve (BPP-P), we follow similar approach to the general case presented in Section 4.1. Let $G = \{g_0(x), g_1(x), \ldots, g_m(x)\}$ where $g_0(x) = 1$. Using $\mathbb{K}^r_G$ as the approximation to $\mathcal{P}_d(S)$, we define the polynomial programming master problem

$$\begin{aligned} \varphi^d_G \ = \inf_{\lambda}\ & \lambda \\ \text{s.t. } & \lambda - f(x) \in \mathbb{K}^d_G, \end{aligned} \tag{4.5}$$

which can be written as

$$\begin{aligned} \varphi^d_G \ = \inf_{\lambda, \sigma_i(x), \delta_i(x)}\ & \lambda \\ \text{s.t.} \quad & \lambda - f(x) = \sigma_0(x) + \sum_{i=1}^{m} \sigma_i(x) g_i(x) + \sum_{i=1}^{n} \delta_i(x)(1 - x_i^2). \\ & \sigma_0(x) \in \Psi_d[x] \\ & \sigma_i(x) \in \Psi_{d - \deg(g_i)}[x] && i = 1, \ldots, m \\ & \delta_i(x) \in \mathbf{R}_{d-2}[x] && i = 1, \ldots, n. \end{aligned}$$

Let $H_j = \{x \in \mathbb{R}^n : x_j \in \{-1, 1\}\}$ and $H = \{-1, 1\}^n$. Notice that $H = \cap_{j \in \{1, \ldots, n\}} H_j$. Instead of solving the polynomial generation subproblem over $\mathbb{K}^{d+2}_G$ as defined in Section 4.1.1, we use the following theorem to obtain a polynomial generation subproblem for the binary case that is computationally cheaper:

Theorem 4.2.1. *[80] For any degree d and compact set D,*

$$\mathcal{P}_d(D \cap H_j) = \big((1+x_j)\mathcal{P}_d(D) + (1-x_j)\mathcal{P}_d(D) + (1-x_j^2)\mathbf{R}_{d-1}[x]\big) \cap \mathbf{R}_d[x].$$

Proof. Define $\mathcal{C}_j^d := ((1+x_j)\mathcal{P}_d(D) + (1-x_j)\mathcal{P}_d(D) + (1-x_j^2)\mathbf{R}_{d-1}[x]) \cap \mathbf{R}_d[x]$.

$\supseteq$ **:** Consider $p(x) \in \mathcal{C}_j^d$, that is $p(x) = (1+x_j)q^+(x) + (1-x_j)q^-(x) + (1-x_j^2)r(x)$ where $r(x) \in \mathbf{R}_{d-1}[x]$, and $q^+(x)$, $q^-(x)$ are in $\mathcal{P}_d(D)$.

Let $s \in D \cap H_j$, then $s_j \in \{-1, 1\}$. For the case of $s_j = -1$ we have

$$p(s) = 2q^-(s) \geq 0.$$

Similarly for the case of $s_j = 1$, we obtain

$$p(s) = 2q^+(s) \geq 0.$$

Hence, $p(s) \geq 0$ for all $s \in D \cap H_j$.

$\subseteq$ **:** Take a polynomial $p(x) \in \mathcal{P}_d(D \cap H_j)$, we have $h_j = (1-x_j)(1+x_j)$. Define $\eta_1(x_j) = \frac{(1-x_j)}{2}$ and $\eta_2(x_j) = \frac{(x_j+1)}{2}$. We have $\eta(x_j) = \eta_1(x_j) + \eta_2(x_j) = 1$.

First consider the case where $p(x) \in \mathcal{P}^+(D)$ then $p(x) = \eta_1(x_j)p(x) + \eta_2(x_j)p(x) \in \mathcal{C}_j^d$. For the second case where $p(x) \notin \mathcal{P}^+(D)$, consider $\nu = |\min\{p(x) : x \in D\}| \geq 0$ and $\varepsilon_1 = \{(x_j+1)^2 : p(x) \leq 0,\ x \in D\} > 0$, $\varepsilon_2 = \{(x_j-1)^2 : p(x) \leq 0,\ x \in D\} > 0$. Note: the reason we need compactness on D for this theorem is to ensure that ν, ε_1, and ε_2 are attained and finite.

Let $\phi_1(x) = p(x) + \mu(x_j+1)^2$ and $\phi_2(x) = p(x) + \mu(x_j-1)^2$, where $\mu > \max\{\frac{\nu}{\varepsilon_1}, \frac{\nu}{\varepsilon_2}\}$. We have $\phi_1(x), \phi_2(x) \in \mathcal{P}_d^+(D)$ and hence,

$$p(x) = -\mu(\eta_1(x_j)(x_j+1)^2 + \eta_2(x_j)(x_j-1)^2) + \eta_1(x_j)\phi_1(x) + \eta_2(x_j)\phi_2(x) \in \mathcal{C}_j^d.$$

□

From Theorem 4.2.1, it is natural to define the operator

$$\mathcal{C}_j^d(\mathcal{K}) := \big((1+x_j)\mathcal{K} + (1-x_j)\mathcal{K} + (1-x_j^2)\mathbf{R}_{d-1}[x]\big) \cap \mathbf{R}_d[x],$$

for any $\mathcal{K} \subseteq \mathbf{R}_d[x]$.

The following lemma captures the main properties of the operator $\mathcal{C}_j^d$, which are key to obtain a more efficient DIGS for the binary case:

Lemma 4.2.2. *Let $D \subset [-1,1]^n$ and let $\mathcal{K} \subseteq \mathcal{P}_d(D \cap H)$.*

1. *For every j, $2\mathcal{K} \subseteq \mathcal{C}_j^d(\mathcal{K}) \subseteq \mathcal{P}_d(D \cap H)$,*
2. *If $\mathcal{P}_d(D) \subseteq \mathcal{K}$ then $\mathcal{P}_d(D \cap H_j) \subseteq \mathcal{C}_j^d(\mathcal{K})$,*
3. *Moreover, if $\mathcal{P}_d(D) \subseteq \mathcal{K} \subsetneq \mathcal{P}_d(D \cap H)$ then for some j, $\mathcal{K} \subsetneq \mathcal{C}_j^d(\mathcal{K})$.*

Proof.

1. From the definition of $\mathcal{C}_j^d$, $2\mathcal{K} = (1-x_j)\mathcal{K} + (1+x_j)\mathcal{K} \subseteq \mathcal{C}_j^d(\mathcal{K})$. Now, let $q(x) \in \mathcal{C}_j^d(\mathcal{K})$, then
$$q(x) = (1+x_j)p_1(x) + (1-x_j)p_2(x) + (1-x_j^2)c(x)$$
with $p_1(x), p_2(x) \in \mathcal{K}$ and $c(x) \in \mathbf{R}_{d-1}[x]$. For all $x \in H_j$, $q(x) = 2p_1(x)$ or $q(x) = 2p_2(x)$. Thus, since $\mathcal{K} \subseteq \mathcal{P}_d(D \cap H)$, $q(x) \geq 0$ for all $x \in D \cap H$.
2. Follows from Theorem 4.2.1.
3. For sake of contradiction, assume $\mathcal{K} = \mathcal{C}_j^d(\mathcal{K})$ for all j. Applying part 2 inductively, $\mathcal{P}_d(D \cap \cap_{i \leq j} H_j) \subseteq \mathcal{C}_j^d(\mathcal{K}) = \mathcal{K}$ for all $j = 1, \ldots, n$. In particular $\mathcal{P}_d(D \cap H) \subseteq \mathcal{K}$.

□

From Lemma 4.2.2, if $\mathcal{K} \subset \mathcal{P}_d(S)$ then $\mathcal{C}_j^d(\mathcal{K}) \setminus \mathcal{K} \neq \emptyset$. Therefore, we can substitute $\mathbb{K}_G^{d+2}$ with $\mathcal{C}_j^d(\mathbb{K}_G^d)$ in the definition of Subproblem (4.3). Let Y be the optimal dual variable for (4.5), we define the j-th polynomial generation subproblem for the binary case as:

$$\begin{aligned} \omega_j \;=\; & \min_p \; \langle p, Y \rangle \\ & \text{s.t. } p(x) \in \mathcal{C}_j^d(\mathbb{K}_G^d) \\ & \quad \| \, p \, \| \leq 1. \end{aligned} \tag{4.6}$$

The first constraint of Problem (4.6), implies that $p(x) = (1 + x_j)q^+(x) + (1 - x_j)q^-(x) + (1 - x_j)^2 r(x)$ where $q^+(x)$, $q^-(x) \in \mathbb{K}^d_G$, $r(x) \in \mathbf{R}_{d-1}[x]$, and $p(x) \in \mathbf{R}_d[x]$. Using the proof of Theorem 4.2.1, we have

$$p(x) = q^+(x) - \mu(x_j + 1)^2 = q^-(x) - \mu(x_j - 1)^2$$

where $q^+(x), q^-(x) \in \mathbb{K}^d_G$ and $\mu \in \mathbb{R}^+$. Assuming $p(x) \in \mathbf{R}_d[x]$ the polynomial generation subproblem (4.6) involves double the variables of the original problem (4.5) in addition to the $\binom{n+d}{d}$ variables corresponding to $p(x)$. The number of constraints is at most $\binom{n+d+1}{d+1}$ (since we solve a problem of degree $d+1$, however not all the terms of degree $d+1$ appear).

Related to (4.6), we define a similar but less expensive subproblem by replacing the operator $\mathcal{C}^d_j(\mathbb{K}^d_G)$ in Problem (4.6) with $\bar{\mathcal{C}}^d_j(\mathbb{K}^d_G)$, where

$$\bar{\mathcal{C}}^d_j(\mathbb{K}^d_G) := \{p(x) \in \mathbf{R}_d[x] \ : \ p(x) = q^+(x) - \mu(x_j + 1)^2,\ p(x) = q^-(x) - \mu(x_j - 1)^2\}$$

and the unknowns are $q^+(x)$, $q^-(x) \in \mathbb{K}^d_G$ and $\mu \in \mathbb{R}^+$. Using $\bar{\mathcal{C}}^d_j(\mathbb{K}^d_G)$, the polynomial generation subproblem involves almost the same number of variables as before but the number of constraints in this case is $2\binom{n+d}{d}$ which is significantly less than $\binom{n+d+1}{d+1}$.

For the binary case, DIGS-B is computationally more efficient than the DIGS presented in Section 4.1.1 for the general case. The master problem in both cases is of the same size, but solving the subproblem (4.6) is basically of the same order as solving the master problem (4.5). At each iteration, Subproblem (4.6) has twice the number of variables and at most $\frac{n+d+1}{d+1}$ times the number of constraints of the master problem when using $\mathcal{C}^d_j(\mathbb{K}^d_G)$. This is much smaller than subproblem (4.3) obtained for the general case which has $O(n^2/d^2)$ times the number of variables and $O(n^2/d^2)$ times the number of constraints compared to the master problem. Using Algorithm 2 with the master problem and subproblem replaced by (4.5) and (4.6) respectively, we obtain a specialized DIGS for the binary case, which we call DIGS-B.

Algorithm 3: DIGS-B for Binary Polynomial Programs

Input: $G = \{g_i(x) : i = 1, \ldots, m\}$, $f(x)$

Output: μ

set: $i = 0$, $G_0 = G$, and Stop=0

while *Stop=0* **do**

Let Y^i be a optimal dual solution of:

$$\begin{aligned} \mu = \min_{\lambda} \ & \lambda \\ \text{s.t. } & \lambda - f(x) \in \mathbb{K}^d_{G_i} \end{aligned}$$

;

Let $p_i(x) \in \mathbf{R}_d[x]$ be a solution to:

$$\begin{aligned} \min_{p} \ & \langle p, Y^i \rangle \\ \text{s.t. } & p(x) \in \mathcal{C}^d_j(\mathbb{K}^d_{G_i}) \\ & \| \, p \, \| \leq 1 \end{aligned}$$

where $j \leq n$ is a given index ;

if $\langle p, Y^i \rangle > -10^{-3}$ *for all* j **then**

Stop=1;

else

set: $G_{i+1} := G_i \cup \{p_i\}$;

set: $i := i + 1$;

end

end

The algorithm terminates when for all indexes j, the subproblem has an objective value equal to zero. In practice, we stop when for all j the value of the subproblem is sufficiently close to zero. We need to choose a variable x_j to apply the iterative procedure on. For this, we propose a heuristic that allows us to choose the x_j variable as described below.

Let $[f(x)]_\alpha$ be the vector of coefficients of the polynomial $f(x)$. At a given iteration, we want to minimize $\langle p, Y\rangle$ which is equivalent to minimizing

$$\left\langle [q^+(x) - \mu(x_j+1)^2]_\alpha, Y\right\rangle = \left\langle q^+, Y\right\rangle - \mu\left\langle [(x_j+1)^2]_\alpha, Y\right\rangle$$
$$\left\langle [q^-(x) - \mu(x_j-1)^2]_\alpha, Y\right\rangle = \left\langle q^-, Y\right\rangle - \mu\left\langle [(x_j-1)^2]_\alpha, Y\right\rangle$$

We have $\langle q^+, Y\rangle$ and $\langle q^-, Y\rangle$ are always nonnegative, and μ is nonnegative. Hence to minimize the objective of the subproblem we need to find the x_j that will take a minimum value in:

$$\max_j \min\{\left\langle [1+2x_j+x_j^2]_\alpha, Y\right\rangle, \left\langle [1-2x_j+x_j^2]_\alpha, Y\right\rangle\}$$

so we evaluate $\left\langle [1+2x_j+x_j^2]_\alpha, Y\right\rangle$ and $\left\langle [1-2x_j+x_j^2]_\alpha, Y\right\rangle$ to be equal to $Y_{00}+2Y_{0j}+Y_{jj}$ and $Y_{00}-2Y_{0j}+Y_{jj}$ respectively. Note that Y_{00} and Y_{jj} are each equal to 1, hence we choose the index j that correspond to the most fractional Y_{0j}. That is we choose the x_j closest to zero (most fractional from $\{-1,1\}$).

In order to avoid repetition of variables one can use a weighted version of the method where the variables are assigned with weights w_j. The weights of all the variables are all initialized to one. Once the variable is chosen at a given iteration, then its weight is doubled, i.e., $w_j = 2w_j$. Otherwise, if a variable is not chosen its weight is the maximum of one and its previous weight minus one, i.e., $w_j = \max\{1, w_j - 1\}$.

To show how the DIGS-B works, we consider the following example:

Example 4.2.3. *Consider the non-convex quadratic knapsack problem with $n = 3$ and $d = 2$:*

$$\begin{aligned} \max\ & 62x_1 + 19x_2 + 28x_3 + 52x_1x_2 + 74x_1x_3 + 16x_2x_3 \\ s.t.\ & 12x_1 + 44x_2 + 11x_3 \le 66 \\ & x_1, x_2, x_3 \in \{0,1\}. \end{aligned} \tag{4.7}$$

The optimal value for (4.7) is $z = 164$. Let $f(x) = 62x_1 + 19x_2 + 28x_3 + 52x_1x_2 + 74x_1x_3 + 16x_2x_3$. Setting $r = 2$ and replacing the condition $x_i \in \{0,1\}$ with $0 \le x_i \le 1$ and

$x_i^2 - x_i = 0$, we obtain the following relaxation

$$\min \lambda$$

$$\text{s.t. } \lambda - f(x) = s(x) + a(66 - 12x_1 - 44x_2 - 11x_3) + \sum_{i=1}^{3} b_i(1 - x_i) + \sum_{i=1}^{3} c_i x_i + \sum_{i=1}^{3} d_i(x_i - x_i^2),$$

$$s(x) \in \Psi_2[x], a, b_i, c_i \in \mathbb{R}_+,\ d_i \in \mathbb{R}$$

which has an objective value of 249.16. This is an upper bound on the optimal value of (4.7).

If one wants to improve the value using Lasserre's method, the hierarchy of SDPs shown in Table 4.3 must be solved. From Figure 4.3, one can see the exponential increase in the linear system of equations one has to solve when performing Lasserre's method for $r = 2, 4,$ and 6 respectively. For $r = 2$, the linear system of equations is of size 10×30 which grows to 84×1701 for $r = 6$.

Table 4.3: Results for Lasserre's hierarchy. The optimal solution is obtained with $r = 6$.

r	2	4	6	8
objective value	249.1	226.2	164.0	164.0
psd matrices	4×4(1)	10×10(1)	20×20(1)	35×35(1)
	1×1(13)	4×4(13),	10×10(13),	20×20(13)
total # of vars	23	185	925	3360
total # of constraints	10	35	84	165

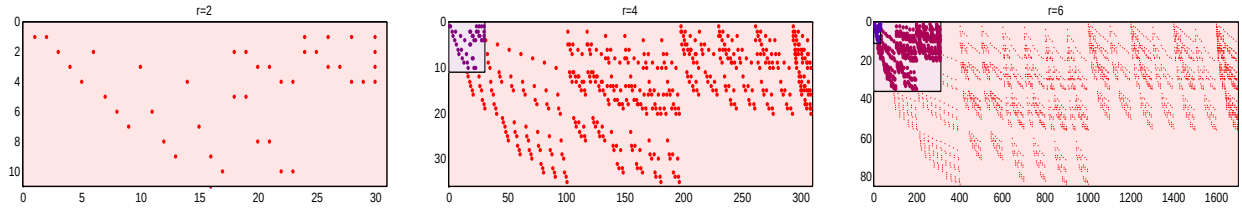

Figure 4.3: Size of the linear system of equations for Lasserre's hierarchy: Example 4.2.3.

By contrast, using our method we first generate a quadratic valid inequality

$$p(x) = 0.809 - 0.388x_1 - 0.037x_2 - 0.361x_3 - 0.099x_1x_2 - 0.086x_2x_3$$

and solve (4.5) with $r = 2$ again, this time taking $G = \{1, 66 - 12x_1 + 44x_2 + 11x_3, x_i, 1 - x_i, x_i^2 - x_i, x_i - x_i^2, p(x)\}$. An objective function value of 243.22 is obtained. Performing this approach iteratively, one is able to improve the bound and obtain a tighter approximation of the original problem. After adding 11 inequalities we obtain an objective of 164.00 which is the optimal value of (4.7). In Table 4.4, the details on the size of the corresponding master problem and polynomial generation subproblem are given.

Table 4.4: Results for DIGS-B.

i	0	1	2	3	$\cdots$	9	10	11
objective value	249.1	243.2	238.9	235.1	$\cdots$	164.2	164.0	164.0
Master Problem								
psd matrices	4×4(1)	4×4(1)	4×4(1)	4×4(1)	$\cdots$	4×4(1)	4×4(1)	4×4(1)
non-negative vars	7	8	9	10	$\cdots$	16	17	18
free vars	3	3	3	3	$\cdots$	3	3	3
total # of vars	20	21	22	23	$\cdots$	29	30	31
total # of constraints	10	10	10	10	$\cdots$	10	10	10
Subproblem								
psd matrices	4×4(2)	4×4(2)	4×4(2)	4×4(2)	$\cdots$	4×4(2)	4×4(2)	
non-negative vars	14	16	18	20	$\cdots$	32	34	
free vars	20	20	20	20	$\cdots$	20	20	
total # of vars	54	56	58	60	$\cdots$	72	74	
total # of constraints	20	20	20	20	$\cdots$	20	20	

4.2.2 Lift-and-Project

Lift-and-project is a well-known method for generating cutting planes. Balas et al. [6] propose an iterative lift-and-project procedure which after n iterations yields the convex hull of the feasible binary points. Sherali and Adams propose a similar lift-and-project approach where the lifting part is obtained through simultaneous multiplication of the original constraint set by all the binary variables and their complements (i.e., x_i and

$(1 - x_i)$) followed by projection to the original space. Lovász and Schrijver presented and LP and SDP-based procedures for obtaining strong relaxations. In this section, we re-derive the lift-and-project approach presented by Balas et al. [6] and show that the approach presented in Section 4.2.1 generalizes this result.

For the case of binary linear programming, define $D = \{x \in \mathbb{R}^n : a_i^T x \geq b_i, i = 1, \ldots, m\} \subseteq [-1, 1]^n$ and $\mathbb{K}_G^1 = \sum_{i=1}^m (a_i^T x - b_i)\Psi_0[x] = \mathcal{P}_1(D)$. From Proposition 4.2.4, we can use Algorithm 3 to obtain Balas et al. results (see Example 4.3.7).

Proposition 4.2.4. *For $d = 1$ and D a polyhedron, our method is a generalization of Balas, Ceria, and Cornuéjols lift-and-project method.*

Proof. The set presented in [6] (denoted as $P_j(x)$ there) is the dual of $\{p(x) \in \mathbf{R}_d[x] : p(x) \in \mathcal{C}_j^1(\mathbb{K}_G^1)\}$, the set of valid inequalities of the convex hull of $D \cap H_j$. □

To show the relation between Balas et al. method and our approach in more details, we consider the LP relaxation of the following binary problem

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } Ax \geq b, \\ &\qquad x \in \{0, 1\}. \end{aligned}$$

Using Balas et al. lift and project on the variable x_j (for a given j), we obtain for the lifting step

$$\begin{aligned} &\min c^T x \\ &\text{s.t. } (1 + x_j)(Ax - b) \geq 0 \\ &\qquad (1 - x_j)(Ax - b) \geq 0, \end{aligned}$$

where the constraints include the box constraints $-1 \leq x_i \leq 1$ for all i. Next we apply the linearization/projection by substituting the product of $x_j x_k$ by a new variable x_{jk} and

setting $x_j^2 = 1$,

$$\begin{aligned}
\textbf{(P}_\textbf{L}\textbf{)}\ \min & \sum_i c_i x_{0i} \\
\text{s.t.}\ & \sum_i a_{ik} x_{0i} + \sum_i a_{ik} x_{ij} - b_k x_{0j} \geq b_k \qquad \forall k \qquad (4.8) \\
& \sum_i a_{ik} x_{0i} - \sum_i a_{ik} x_{ij} + b_k x_{0j} \geq b_k \qquad \forall k \qquad (4.9) \\
& x_{jj} = 1. \qquad (4.10)
\end{aligned}$$

On the other hand, the problem

$$\begin{aligned}
& \max \lambda \\
\text{s.t.}\ & \sum_i c_i x_i - \lambda \in \mathcal{C}_j^1(\mathbb{K}_G^1)
\end{aligned}$$

is equivalent to

$$\max \lambda$$
$$\text{s.t.}\ \sum_i c_i x_i - \lambda = (1 + x_j) \left[\sum_k \mu_k (a_k^T x - b_k) \right] + (1 - x_j) \left[\sum_k \nu_k (a_k^T x - b_k) \right] + r(1 - x_j^2).$$

Equating the coefficients of the monomials of the left side to the ones on the right side

$$\begin{aligned}
&\max \lambda \\
&\text{s.t. } \lambda = \sum_k \mu_k b_k + \sum_k \nu_k b_k - r \\
&\quad c_j = -\sum_k b_k \mu_k + \sum_k b_k \nu_k + \sum_k a_{kj} \mu_k + \sum_k a_{kj} \nu_k \\
&\quad c_i = \sum_k a_{kj} \mu_k + \sum_k a_{kj} \nu_k \\
&\quad 0 = -\sum_k a_{kj} \mu_k - \sum_k a_{kj} \nu_k - r \\
&\quad 0 = -\sum_k a_{kj} \mu_k - \sum_k a_{kj} \nu_k
\end{aligned}$$

Substituting λ by $\sum_k \mu_k b_k + \sum_k \nu_k b_k - r$ in the objective, we obtain the dual problem of ($\mathrm{P_L}$) where μ_k, ν_k, and r are the dual variables of constraints (4.8)-(4.10) respectively.

4.2.3 Convergence Results

We provide proof of convergence for DIGS-B procedure for some important cases including binary polynomial programs with quadratic (or linear) objective and linear constraints (including the unconstrained case).

We first start by providing a proof of convergence for the case when the approximation $\mathbb{K}^d_G$ contains $\mathcal{P}_d(D)$. Notice that if $\mathbb{K}^d_G \supseteq \mathcal{P}_d(D)$ by Lemma 4.2.2-2, Subproblem (4.6) is equivalent to optimizing over a set containing $\mathcal{P}_d(D \cap H_j)$. Intuitively, if the subproblem has a value of 0, it is because "using the fact that x_j is binary cannot help", i.e., the solution is already "binary" in that coordinate. Thus if the value of all subproblems is zero we have converged to the optimal value. This intuition is formally expressed in Theorem 4.2.5.

Theorem 4.2.5. *Let $d \geq 2$ and let $D \subseteq [-1,1]^n$ be a compact set. Assume $\mathcal{P}_d(D) \subseteq \mathbb{K}^d_G \subseteq \mathcal{P}_d(D \cap \{-1,1\}^n)$. Let ω_j, φ, z be the optimal objective value of subproblem* (4.6)*, master problem* (4.5)*, and the original binary polynomial program (BPP-P) respectively. If $\omega_j = 0$ for all indexes j, then $\varphi = z$.*

Proof. Let (λ, Y) be the optimal primal-dual solution of (4.5). Then $\varphi = \lambda \geq z$ and $Y \in \{X \in (\mathbb{K}^d_G)^* : \langle 1, X\rangle = 1\} \subseteq \text{conv}(\mathcal{M}_d(D))$, using Lemma 2.2.30. As D is compact, $\text{conv}(\mathcal{M}_d(D))$ is compact. Thus, by Caratheodory's Theorem, Y can be written as $Y = \sum_i a_i \mathcal{M}_d(u_i)$ with $a_i > 0$, $\sum_i a_i = 1$ and each $u_i \in D$.

Notice that $\langle f, Y\rangle = \sum_i a_i \langle f, \mathcal{M}_d(u_i)\rangle = \sum_i a_i f(u_i)$. If $u_i \in H$ for all i, $\varphi = \langle f, Y\rangle \leq z$ and we are done. To get a contradiction, assume $u_k \notin H$ for some k. Then there is $j \leq n$ such that $u_k \notin H_j$. Consider $p(x) = 1 - x_j^2$. We have $p(x) \in \mathcal{P}_d(H_j) \subseteq \mathcal{P}_d(D \cap H_j) \subseteq \mathbb{K}^d_G$, and $p(u_k) > 0$. Therefore,

$$\omega_j \geq \langle p, Y\rangle = \sum_i a_i \langle p, \mathcal{M}_d(u_i)\rangle = \sum_i a_i p(u_i) \geq a_k p(u_k) > 0,$$

which is a contradiction. □

We apply Theorem 4.2.5 in the case of pure quadratic binary programming. Taking D as the ball $\mathcal{B} = \{x \in \mathbb{R}^n : \|x\|^2 \leq n\}$ and $d = 2$, it follows from the $\mathcal{S}-$Lemma [83] that $\mathcal{P}_2(D) = \Psi_2[x] + (n - \|x\|^2)\mathbf{R}_+$ and we can apply Theorem 4.2.5.

Theorem 4.2.6. *When Algorithm 3 is applied to the case of pure binary quadratic programming, starting with $G_0 = \{n - \|x\|^2, 1\}$, if all the subproblems have an optimal value 0, then the value of the master is equal to the optimal value.*

For linearly constrained binary quadratic programming, when all the linear constrains are equalities, we can also show that Algorithm 3 converges. This case does not follow directly from Theorem 4.2.5 but rather from a modification of the proof.

Theorem 4.2.7. *When Algorithm 3 is applied to the case of a binary quadratic programming, constrained to $Ax = b$, starting with $G_0 = \{1, n - \|x\|^2, (A_i^T x - b_i)^2, -(A_i^T x - b_i)^2,\ i = 1, \ldots, n\}$ if all the subproblems have an optimal value 0, then the value of the master problem is equal to the optimal value.*

Proof. Let Y be the optimal dual solution of (4.5). As in the proof of Theorem 4.2.5 we write $Y = \sum_j a_j \mathcal{M}_2(u_j)$, with $u_j \in \mathcal{B}$. Let $h(x) = \sum_i (A_i^T x - b_i)^2$, we claim $h(u_j) = 0$ for all j. Notice that this is enough, as Theorem 4.2.5 implies that each u_j is binary. During

any step s of the Algorithm 3, $\pm h(x) \in \mathbb{K}^2_{G_{A,b}} \subseteq \mathbb{K}^2_{G_s}$. If we are at step s, $Y \in (\mathbb{K}^2_{G_s})^*$ and hence $\langle \pm h, Y \rangle \geq 0$. Thus, $0 = \langle h, Y \rangle = \sum_j a_j h(u_j)$, which implies $h(u_j) = 0$ for each j. □

4.3 Examples and Computational Results

In this section, to illustrate the iterative dynamic scheme, we apply Algorithms 2 and 3 to several examples and report computational results. To solve these examples, we developed a Matlab code that constructs and builds the resulting relaxations of the polynomial program and solves them using the SeDuMi solver [93], see APPS in Chapter 6 for details. The subproblems are solved with 10^{-4} precision. To obtain a fair comparison, Lasserre's relaxation was solved using the same code, on the same machine.

4.3.1 General case

Unless otherwise specified, the following topping criteria are used

- For all algorithms a time limit of 5 hours (18000 seconds) is imposed. When the algorithm does not terminate in the time limit this is expressed using a dash (-).
- For the dynamic scheme, DIGS, an iteration limit of at most 10 added inequalities.

Example 4.3.1.

$$\begin{aligned} \min_x \quad & x_1 - x_1x_3 - x_1x_4 + x_2x_4 + x_5 - x_5x_7 - x_5x_8 + x_6x_8 \\ s.t. \quad & x_3 + x_4 \leq 1 \\ & x_7 + x_8 \leq 1 \\ & 0 \leq x_i \leq 1 \qquad \forall i \in \{1, \ldots, 8\}. \end{aligned}$$

The optimal objective value of the above problem is 0. Lasserre's hierarchy needs $r \geq 10$ to obtain the optimal value. However, for $r = 8$ even constructing the problem couldn't

be done within five hours as shown in Table 4.6. Using DIGS, we are able to use relaxations of degree 2 and add quadratic inequalities. Table 4.5, presents the bounds and the computational time for 10 iterations.

Table 4.5: DIGS Results for Example 4.3.1.

Iter.	0	1	2	3	4	5	6	7	8	9	10
Obj.	unb.	-0.109	-0.073	-0.069	-0.068	-0.066	-0.062	-0.060	-0.060	-0.059	-0.057
T(sec)											
Master	0.2	0.3	0.3	0.4	0.5	0.5	0.5	0.5	0.6	0.6	0.6
Sub.	1.5	1.8	2.1	1.9	2.0	2.1	2.2	2.3	2.4	2.5	
Accum.	1.7	3.8	6.2	8.5	11.0	13.6	16.3	19.1	22.1	25.2	25.8

Performing 50 iterations of DIGS, we obtain a lower bound of value -0.014 in 200.1 seconds. Figure 4.4 illustrates the bound improvement.

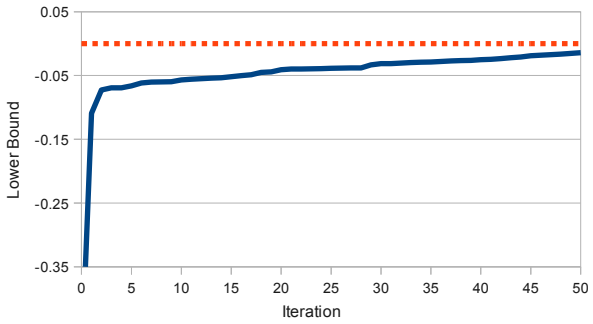

Figure 4.4: DIGS lower bounds for Example 4.3.1. The dotted line is the optimal objective value.

Table 4.6: Lasserre's Hierarchy for Example 4.3.1.

r	2	4	6	8	10
Obj.	unb.	-0.03550	-0.00192	-	-
T(sec)	1.02	2.81	726.50	-	-

Example 4.3.2. ***Motzkin and Robinson Polynomials:***

$$\min_{x,y,z\in\mathbb{R}} \quad x^2y^2(x^2+y^2-3z^2)+z^6 \tag{4.11}$$

$$\min_{x,y,z\in\mathbb{R}} \quad x^6+y^6+z^6-(x^4y^2+x^2y^4+x^4z^2+x^2z^4+y^4z^2+y^2z^4)+3x^2y^2z^2 \tag{4.12}$$

The Motzkin and Robinson Polynomials ((4.11) and (4.12) respectively) are non-negative polynomial that are not a sum-of-squares. The global optimum when minimizing over these polynomials is 0. For both problems, at Iteration 0 we obtain an unbounded objective function value. Using the dynamic scheme and adding 10 nonlinear inequalities of degree 6 we obtain the following results as shown in Tables 4.7 and 4.8.

Table 4.7: DIGS Results for the Motzkin Polynomial, Example 4.3.2-(4.11).

Iter.	0	1	2	3	4	5	6	7	8	9	10
Obj.	unb.	-8591.8	-5687.1	-663.8	-643.8	-640.7	-640.4	-618.8	-618.3	-614.4	-613.5
T(sec)											
Master	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Sub.	0.4	0.3	0.3	0.4	0.4	0.4	0.5	0.5	0.5	0.5	
Accum.	0.7	1.3	1.9	2.6	3.3	4	4.8	5.6	6.4	7.2	7.5

Table 4.8: DIGS Results for the Robinson Polynomial, Example 4.3.2-(4.12).

Iter.	0	1	2	3	4	5	6	7	8	9	10
Obj.	unb.	-7848.5	-5501.7	-5149.3	-5122.1	-5110.6	-4228.7	-3287.6	-3024.3	-1862.0	-1849.1
T(sec)											
Master	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.3	0.3	0.3
Sub	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.5	
Accum.	0.6	1.2	1.8	2.6	3.4	4.2	5.0	5.8	6.5	7.3	7.6

Note that because the Motzkin and the Robinson are homogeneous polynomials then the optimum value is either 0 or $-\infty$. Using the dynamic scheme we obtain a finite lower bound after iteration 1 and therefore we have a proof that the global minimum is zero. As a result, we can terminate the algorithm after iteration 1 of DIGS with an optimum value of zero. On the other hand, using Lasserre's hierarchy we obtain an unbounded problem for all values of r since the Motzkin and Robinson Polynomials cannot be written as sum-of-squares.

Example 4.3.3. *Consider the Motzkin Polynomial again but with the additional constraint where the solution has norm at least 1:*

$$\begin{aligned} \min_{x,y,z\in\mathbb{R}} \quad & x^2y^2(x^2+y^2-3z^2)+z^6 \\ s.t. \quad & x^2+y^2+z^2 \geq 1. \end{aligned}$$

We obtain an unbounded problem at Iteration 0 of DIGS. However, after applying one iteration of DIGS we have a finite bound. Comparing with Lasserres relaxation, when $r = 6$ and 8 the problem is still unbounded as show in Table 4.10. Gloptipoly [38] gives numerical errors for $10 \leq r \leq 16$. Using DIGS we obtain a finite bound within 1.4 seconds. Table 4.9 shows the results for 10 iterations of DIGS.

Table 4.9: DIGS Results for Example 4.3.3.

Iter.	0	1	2	3	4	5	6	7	8	9	10
Obj.	unb.	-6485.9	-3516.3	-3508.8	-2999.4	-2998.6	-2910.7	-2900.2	-2896.0	-2877.3	-2871.3
T(sec)											
Master	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
Sub.	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	
Accum.	0.7	1.4	2.2	3.1	4.0	4.9	5.9	6.9	7.9	9.0	9.5

Table 4.10: Lasserre's Hierarchy for Example 4.3.3. A (*) means that SeDuMi reported a gap between the primal and dual optimal values.

r	6	8	10	12	14	16	18	20	22	24
Obj.	unb.	unb.	*	*	*	*	-3.0×10^{-3}	-4.4×10^{-4}	-4.3×10^{-5}	-
T(sec)	0.7	3.9	*	*	*	*	160.9	368.4	1132.1	-

4.3.2 Binary case

As examples for the binary case, we consider constrained binary polynomial programs of degree 2 and 3. Unless otherwise specified, the following stopping criteria are used

- For all algorithms a time limit of 5 hours (18000 seconds) is imposed. When the algorithm does not terminate in the time limit this is expressed using a dash (-)
- For the dynamic schemes, DIGS and DIGS-B, an iteration limit of at most 10 added inequalities.
- For DIGS-B we stop if the subproblems have a value close to 0 ($\leq 10^{-3}$) or if we are able to extract a feasible optimal solution and report optimality. To extract a solution, we round the linear monomials to -1 or 1 according to their sign and check for feasibility and optimality.

For the examples presented, we report the objective function value at Iteration 0 and after performing a number of iterations of DIGS-B. As a reference for comparison we also present Lasserre's results. Bold values indicate that the approach used terminated reporting optimality.

For the stable set example, we report results of DIGS-B within a specific time limit of 300 seconds and compare with Balas et al. lift-and-project method. We note that using the Matlab code we developed, it is possible to capture Lasserre and Balas et al. approaches and hence we use the same code and machine for all comparisons.

Example 4.3.4. ***Quadratic Knapsack Problem (QKP):***
Recall the QKP problem (QKP-P) defined in Section 3.2.4.

$$\begin{aligned} &\max x^T P x \\ &s.t.\ w^T x \leq c \\ &\quad x \in \{0,1\}^n. \end{aligned}$$

Table 4.11 presents computational results for QKP instances where the parameters are generated according to [82]. The results show that DIGS-B is much more time efficient than Lasserre's approach, in particular when n gets large. For $n > 20$, we are not able to go beyond $r = 2$ for Lasserre's hierarchy in the given time limit of 5 hours while using DIGS-B we are able to improve the bounds by using this iterative scheme.

Table 4.11: Computational results for quadratic knapsack instances.

		Lasserre $r=4$		Lasserre $r=2$		DIGS-B				
n	Optimal	Obj.	T(sec)	Obj.	T(sec)	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
10	1653	1707.3	28.1	1857.7	0.8	1857.7	1821.9	1797.4	1784.8	5.8
20	8510	8639.7	17269.1	9060.3	2.9	9060.3	9015.3	8925.9	8850.3	35.4
30	18229	-	-	19035.9	4.3	19035.9	18920.2	18791.7	18727.2	196.6
40	2679	-	-	4735.9	6.8	4735.9	4590.7	4248.2	4126.7	1009.7
50	16192	-	-	21777.9	19.2	21777.9	21390.3	20162.1	19407.1	7014.3
60	58451	-	-	62324.4	126.6	62324.4	62019.1	60906.0	60585.5	17961.1
70	16982	-	-	23884.9	231.4	23884.9	23484.0	22852.8	-	15582.2
80	-	-	-	80482.7	365.4	80482.7	79738.9	-	-	11072.3

Example 4.3.5. ***Quadratic Assignment Problem (QAP):***
Recall the QAP problem (QAP-P) defined in Section 3.2.2:

$$\begin{aligned} &\min \textstyle\sum_{i\neq k, j\neq l} f_{ik} d_{jl} x_{ij} x_{kl} \\ &s.t.\ \textstyle\sum_i x_{ij} = 1 && 1 \leq j \leq n \\ &\quad \textstyle\sum_j x_{ij} = 1 && 1 \leq i \leq n \\ &\quad x \in \{0,1\}^{n\times n}. \end{aligned}$$

Table 4.12 presents computational results for quadratic assignment instances where f_{ik} and d_{jl} are integers randomly generated uniformly between 0 and 5 with density 80%. Using Lasserre's hierarchy we can only solve the case $r = 2$ for instances of dimension $n > 4$ within 5 hours, while the dynamic scheme improves significantly on the bounds of Lasserre's $r = 2$ relaxation without as much computational effort.

Table 4.12: Computational results for quadratic assignment instances.

		Lasserre $r = 4$		Lasserre $r = 2$		DIGS-B				
n	Optimal	Obj.	T(sec)	Obj.	T(sec)	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
3	46			**46.0**	0.3	**46.0**				0.3
4	52	**52.0**	1154.8	50.8	1.0	50.8	51.8	**52.0**		6.3
5	110	-	-	104.3	3.4	104.3	105.1	106.3	106.8	68.5
6	272	-	-	268.9	9.3	268.9	269.4	269.8	270.2	404.4
7	356	-	-	344.2	18.1	344.2	344.9	345.6	346.0	3331.3
8	100	-	-	77.2	73.2	77.2	77.8	78.9	-	11413.9
9	280	-	-	247.5	281.7	247.5	248.6	-	-	13171.5

Example 4.3.6. ***Max-Sat Problem:***
Given a set of boolean clauses on n variables, find an assignment that maximizes the number of satisfied clauses. The max-sat problem can be formulated as follows

$$\min \sum_{j=1}^{m} \left(\prod_{i \in P_j} \frac{(1+x_i)}{2} \times \prod_{k \in N_j} \frac{(1-x_k)}{2} \right)$$
$$s.t.\ x \in \{-1, 1\}^n.$$

where P_j and N_j are subsets of $\{1, \ldots, n\}$, $P_j \cap N_j = \emptyset$, and m is the number of clauses.

In Tables 4.13-4.15, we present results on max-2-sat instances with randomly generated clauses and applying Laserre's relaxation, DIGS and DIGS-B. In Tables 4.14 and 4.15 results for iterations 0, 1, 5, and 10 and the total time in seconds are reported for DIGS and DIGS-B respectively.

Table 4.13: Computational results for the max-2-sat problem: Lasserre's relaxation.

			Lasserre $r = 4$		Lasserre $r = 2$	
n	m	Optimal	Obj.	T(sec)	Obj.	T(sec)
10	45	4	**4.00**	15.6	3.54	0.4
15	65	3	**3.00**	1103.8	2.75	0.9
20	85	8	**8.00**	11567.2	6.93	1.7
30	130	6	-	-	5.47	6.5
40	170	13	-	-	12.00	14.7
50	215	15	-	-	13.86	41.3
60	252	-	-	-	11.37	106.0
70	295	-	-	-	17.08	258.3
80	340	-	-	-	16.77	579.1

Table 4.14: Computational results for the max-2-sat problem: DIGS.

			DIGS				
n	m	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
10	45	4	3.54	3.85	**4.00**		97.8
15	65	3	2.75	2.78	2.98	**3.00**	7451.8
20	85	8	6.93	7.34	-	-	11829.0

In Table 4.15, instances of sizes 10, 15, and 20 are the same instances used in Table 4.14. Comparing the results of Table 4.14 with Table 4.15, it is clear that using the specialized DIGS-B reduces the computational time significantly and provides better bounds.Further, for $n \geq 25$ we are not able to perform any iterations of DIGS within the time limit, whereas we are able to go up to 80 variables using DIGS-B. Looking at the instance with $n = 15$ we can perform 10 iterations of DIGS-B in around 16 seconds whereas using the general DIGS, solving 10 iterations for $n = 15$ takes almost two hours of computational time.

Table 4.15: Computational results for the max-2-sat problem: DIGS-B.

			DIGS-B				
n	m	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
10	45	4	3.54	3.80	**4.00**		6.0
15	65	3	2.75	2.77	2.90	2.97	16.0
20	85	8	6.93	7.04	7.29	7.42	28.5
30	130	6	5.47	5.54	5.69	5.80	175.6
40	170	13	12.00	12.16	12.41	12.63	793.5
50	215	15	13.86	14.12	14.67	14.96	4025.8
60	252	-	11.37	11.56	11.94	12.19	14548.6
70	295	-	17.08	17.35	17.55	-	16722.3
80	340	-	16.77	16.88	-	-	10062.8

Further, since we have a proof of convergence for the DIGS-B that applies to the max 2-sat problem, we can prove optimality for the instance with $n = 10$. From Tables 4.13 and 4.15, the results show that DIGS-B is much more efficient than Lasserre's approach, in particular for $n > 20$, we are not able to go beyond $r = 2$ for Lasserre's hierarchy in the given time limit while using DIGS-B we are able to improve the bounds iteratively.

Example 4.3.7. ***Maximum Stable Set Problem:***
Given an undirected graph $G(V, E)$, a stable set of G is a set of vertices $U \subseteq V$ such that there is no edge connecting any two vertices in U. The maximum stable set problem is to find a stable set of maximal cardinality. Letting $n = |V|$, and identifying V with $\{1, \ldots, n\}$ the maximum stable set problem can be formulated as follows:

$$\begin{aligned} &\max \left(\sum_{i=1}^{n} x_i \right)^2 \\ &s.t.\ x_i x_j = 0 \qquad \forall (i,j) \in E \\ &\sum_{i=1}^{n} x_i = 1. \end{aligned}$$

We present computational results on the maximum stable set problem on instances taken from [79]. Notice that Laserre's relaxation for $r = 2$, denoted by (SS-L2), is equivalent to

the Lovász theta relaxation, $\vartheta(G)$, as shown below:

$$\begin{aligned}\vartheta(G) = \max \ & \sum_{i,j>0} X_{ij} \\ \text{s.t. } & X_{ij} = 0 \qquad \forall (i,j) \in E \\ & \sum_{i} X_{0i} = 1 \\ & X \succeq 0.\end{aligned}$$

The Lasserre's stable set relaxation for $r = 2$ is formulated as follows:

$$\begin{aligned}&\min \ \lambda \\ &\text{s.t. } \lambda - \left(\sum_{i=1}^{n} x_i\right)^2 = s(x) + \sum_{(i,j)\in E} d_{ij} x_i x_j + c(1 - \sum_{i} x_i)\end{aligned}$$

where $s(x)$ is SOS of degree 2, d_{ij}, and c are free variables. The above formulation is equivalent to

$$\begin{aligned}\textbf{(SS-L2)} \ \min \ & \lambda && && (4.13)\\ \text{s.t. } & \lambda - S_{00} - c = 0 && \to u && (4.14)\\ & S_{0i} - c = 0 \quad \forall i = 1, \ldots, n && \to v_i && (4.15)\\ & -2S_{ij} - 2d_{ij}\delta_{ij} = 2 \quad \forall i, j = 1, \ldots, n && \to Y_{ij} && (4.16)\\ & S \succeq 0, \quad \lambda, c, d_{ij} \in \mathbb{R}, && && (4.17)\end{aligned}$$

where δ_{ij} equals 1 if $(i,j) \in E$ and 0 otherwise. The dual variables of each constraint are given on the right hand side of each constraint. Taking the dual of (SS-L2), we obtain:

$$\max \sum_{i,j} Y_{ij} \tag{4.18}$$

$$\text{s.t. } u = 1 \qquad \rightarrow \lambda \tag{4.19}$$

$$-u - \sum_i v_i = 0 \qquad \rightarrow c \tag{4.20}$$

$$Y_{ij} = 0 \qquad \forall (i,j) \in E \qquad \rightarrow d_{ij} \tag{4.21}$$

$$u \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} - \sum_i v_i \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & Y \end{pmatrix} \succeq 0 \qquad \rightarrow S_{ij}. \tag{4.22}$$

From equations (4.19) and (4.20), we get $-\sum_i v_i = 1$. Let $X = \begin{pmatrix} 1 & -v_i \\ -v_i & Y \end{pmatrix}$ we obtain:

$$\begin{aligned} \max & \sum_{i,j>0} X_{ij} \\ \text{s.t. } & \sum_i X_{0i} = 1 \quad \forall i = 1, \ldots, n \\ & X_{ij} = 0 \qquad \forall (i,j) \in E \\ & X \succeq 0 \end{aligned}$$

which is equivalent to $\vartheta(G)$.

Thus, DIGS can be interpreted here as adding quadratic valid inequalities to strengthen the Lovász theta relaxation, as described in Section 4.1. In Table 4.16, Iteration 0 refers to the Lovász theta bound and iterations 1, 5, and 10 correspond to the upper bounds obtained after performing 1, 5, and 10 iterations of DIGS respectively. The total time is reported in seconds.

Table 4.16: Computational results for the stable set problem: DIGS

		Lasserre $r=4$		Lasserre $r=2$		DIGS				
n	Optimal	Obj.	T(sec)	Obj.	T(sec)	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
8	3	**3.00**	3.8	3.44	0.4	3.44	3.10	3.04	3.02	12.3
11	4	**4.00**	32.2	4.63	0.7	4.63	4.18	4.09	4.08	55.8
14	5	**5.00**	359.4	5.82	0.9	5.82	5.26	5.13	5.11	608.8
17	6	**6.00**	2386.4	7.00	1.3	7.00	6.33	6.24	6.01	4642.8
20	7	**7.00**	13793.7	8.18	1.6	8.18	7.40	-	-	10834.7
23	8	-	-	9.36	2.0	9.36	8.48	-	-	16965.8

The maximum stable set problem can also be formulated as a binary problem as follows:

$$\begin{aligned} \textbf{(SS-LP)}\ \max & \textstyle\sum_i x_i \\ \text{s.t. } & x_i + x_j \leq 1 \qquad \forall (i,j) \in E \\ & x \in \{0,1\}^n. \end{aligned}$$

It can also be formulated as a problem with quadratic constraints by replacing the constraint $x_i + x_j \leq 1$ with $x_i x_j = 0$, we refer to this problem as (SS-D2).

In this case, we compare DIGS-B with the lift-and-project method of Balas et al. [6]. This comparison provides a fair indication of the advantages of our method in terms of bound quality. For each instance we impose a 300 seconds time limit for each procedure. The upper bound for Balas et al. is compared to three approaches of DIGS-B. Linear refers to generating linear inequalities that are added to the master problem by using a non-negative multiplier. SOC refers to generating linear inequalities that are added to the master problem by using a polynomial multiplier that is in $\mathcal{P}_1(\mathcal{B})$ as described in Section 2.2.2. Quadratic refers to generating quadratic inequalities similar to the previous examples described.

Table 4.17: Computational results for the stable set problem with a time limit of 300 seconds.

						DIGS-B					
		(SS-LP)	Balas et al.		(SS-D2)	Linear		SOC		Quadratic	
n	Optimal	UB	UB	Iter.	UB	UB	Iter.	UB	Iter.	UB	Iter.
8	3	4.00	3.00	197	3.44	3.00	186	3.00	126	3.02	49
11	4	5.50	4.00	160	4.63	4.00	139	4.00	130	4.05	109
14	5	7.00	5.02	135	5.82	5.02	114	5.01	91	5.14	82
17	6	8.50	6.22	121	7.00	6.23	84	6.09	63	6.30	54
20	7	10.00	7.46	104	8.18	7.43	68	7.25	45	7.42	38
23	8	11.50	8.81	88	9.36	8.61	50	8.36	33	8.67	22
26	9	13.00	10.11	77	10.54	9.84	37	9.60	25	9.96	14
29	10	14.50	11.65	65	11.71	11.10	24	10.87	17	11.18	10
32	11	16.00	13.03	56	12.89	12.37	18	12.20	14	12.53	6
35	12	17.50	14.48	49	14.07	13.49	13	13.32	10	13.66	4
38	13	19.00	16.05	43	15.24	14.80	8	14.74	7	14.85	4
41	14	20.50	17.69	39	16.42	15.88	7	15.77	6	16.26	1
44	15	22.00	19.10	34	17.59	17.19	6	17.09	5	17.30	1
47	16	23.50	20.78	29	18.77	18.39	4	18.26	4	18.59	1
50	17	25.00	22.18	27	19.94	19.52	4	19.42	4	19.77	1

The results are reported in Table 4.17. Balas et al. performs the largest number of iterations for these instances since it utilizes linear programming which is computationally more efficient, however this efficiency comes at the expense of the bounds. For all instances the bounds obtained by using the SOC version of DIGS-B are the best bounds obtained within 300 seconds. These bounds are comparable with those from the Linear and Quadratic approaches, however Quadratic performs the least number of iterations and still achieves a competitive bound.

Example 4.3.8. ***Degree Three BPP***

As a final example, we consider the general BPP problem with degree 3 objective. The

problem can be formulated as follows

$$\begin{aligned} \max & \sum_{|\alpha|\leq 3} c_\alpha x^\alpha \\ s.t.\ & a^T x \leq b \\ & x \in \{-1,1\}^n. \end{aligned}$$

In Tables 4.18-4.20, we present Lasserre's, DIGS, and DIGS-B results on degree three instances where c_α is generated randomly between 0 and 10, each a is generated randomly between 1 and 50, and b is generated randomly between 50 and $\sum_{i=1}^{n} a_i$.

Laserre's relaxation is not defined for $r = 2$. For $n \geq 25$, we are not able to compute any bound using Laserre's relaxation within the time limit bound of 5 hours.

Table 4.18: Computational results for degree 3 BPP: Lasserre's relaxation

		Lasserre $r = 6$		Lasserre $r = 4$	
n	Optimal	Obj.	T(sec)	Obj.	T(sec)
5	58	**58.00**	9.6	59.37	2.1
10	139	**139.00**	4866.0	148.97	35.9
15	1371	-	-	1524.71	1436.2
20	1654	-	-	1707.95	18106.6
25	-	-	-	-	-

As we are working with a degree 3 BPP, we apply DIGS with a master problem of degree 3 (i.e. $d = 3$) and a subproblem of total degree 4, creating inequalities of degree 2. That is, applying Algorithm 2 with Step 3 replaced by:

3. `LET` $p_i(x) \in \mathbf{R}_{d-1}[x]$ `be a solution of:`

$$\begin{aligned} \min_{p} & \left\langle p, Y^i \right\rangle \\ \text{s.t. } & p(x) \in \mathbb{K}_{G_i}^{d+1} \\ & \| p \| \leq 1. \end{aligned}$$

The results for iterations 0, 1, 5, and 10 are reported in Table 4.19 and the total time is given in seconds.

Table 4.19: Computational results for degree 3 BPP: DIGS

		DIGS				
n	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
5	58	67.16	63.62	**58.00**	-	7.1
10	139	154.59	153.22	142.32	139.19	606.3
15	1371	1582.04	1569.92	1498.60	1470.98	9622.2
20	1654	1718.53	1716.67	-	-	16009.2
25	-	3967.12	-	-	-	5038.6

Remark 4.3.9. *Experimentally, for degree 3 BPP using the same number of iterations of DIGS* $\mathbb{K}^5_G$ *provides better bounds than* $\mathbb{K}^4_G$ *but is* $\mathbb{K}^5_G$ *is more expensive in terms of computational time. For example for the case where* $n = 10$*, performing one iteration of Algorithm 2 with* $\mathbb{K}^5_g$ *results in a bound of* 147.34 *in* 656.5 *seconds while having* $\mathbb{K}^4_G$ *results in a better bound of* 142.32 *in* 5 *iterations and* 297.3 *seconds.*

Table 4.20 presents computational results of applying Algorithm 3. We report results for iterations 0, 1, 5, and 10 and the total time in seconds. At Iteration 0, we use a relaxation of order 3 and then apply DIGS-B to add valid degree 3 inequalities to improve the bounds of the relaxation. Comparing the results of Table 4.19 with Table 4.20, we see that using the specialized DIGS-B reduces the computational time significantly and provides better bounds. In addition to the cubic inequalities we compare two types of inequalities generated by DIGS-B. Tables 4.20-4.22 present computational results for DIGS-B with cubic, quadratic, and linear inequalities respectively. Cubic, quadratic, and linear inequalities are added to the master problem by using a polynomial multiplier that is in $\mathbf{R}^+_0$, $\mathcal{P}_1(\mathcal{B})$, and $\Psi_2[x]$ respectively. The bounds obtained using linear inequalities are the best compared to quadratic and cubic inequalities. Additionally, the bounds provided by using quadratic inequalities outperform the ones obtained when using cubic inequalities. This is mainly due to the polynomial multiplier that provides better certificates and better approximations of the original BPP.

Table 4.20: Computational results for degree 3 BPP: DIGS-B cubic inequalities

		DIGS-B				
n	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
5	58	67.16	58.45	**58.00**		5.2
10	139	154.59	148.85	143.41	139.12	75.3
15	1371	1582.04	1575.49	1519.88	1494.01	1319.9
20	1654	1718.53	1716.00	1708.66	1705.15	15763.9
25	-	3967.12	3960.78	-	-	14287.3

Table 4.21: Computational results for degree 3 BPP: DIGS-B quadratic inequalities

		DIGS-B				
n	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
5	58	67.16	**58.00**			2.4
10	139	154.59	147.00	**139.00**		31.9
15	1371	1582.04	1558.37	1477.52	1436.68	1003.0
20	1654	1718.53	1713.69	1691.76	1682.42	11913.7
25	-	3967.12	3955.07	-	-	10993.4

Table 4.22: Computational results for degree 3 BPP: DIGS-B linear inequalities

		DIGS-B				
n	Optimal	Iter. 0	Iter. 1	Iter. 5	Iter. 10	T(sec)
5	58	67.16	**58.00**			2.1
10	139	154.59	143.86	**139.00**		35.8
15	1371	1582.04	1494.34	1391.12	1371.34	1164.1
20	1654	1718.53	1706.65	1674.87	1654.74	12518.5
25	-	3967.12	3910.05	-	-	13332.7

4.4 Concluding Remarks

In this section, we have presented a dynamic scheme for iteratively generating valid polynomial inequalities that can be used to improve the approximations of polynomial programs. The dynamic scheme described in this section is quite general and applicable to any polynomial programming problem. For the binary case we present a specialized scheme. This scheme is more efficient and provides theoretical convergence warranties.

Additionally, we provide several examples and present comparisons with existing solution methodologies for solving polynomial programs. The methodology presented provides a means to tightly approximate polynomial programs that, unlike previously proposed hierarchies of SDP relaxations, is in principle scalable to large general combinatorial optimization problems.

A topic of further research is to improve the efficiency of solving the subproblem which is essential in terms of the computational efficiency of the algorithm. In future work, we focus on approaches to solve the subproblem more efficiently and utilizing the subproblem to generate more than one inequality. The solving process for the generating subproblems can be improved either by reducing the size of the subproblems or by the application of fast solvers since it is not necessary to find the optimal solution of the subproblem but a feasible solution with negative objective value should suffice. In the next section, the valid inequalities obtained are exploited within a branch-and-dig framework.

Chapter 5

Branch-and-Dig Scheme

One of the most successful frameworks for solving binary programs is branch-and-bound algorithms [52, 61, 68]. Similar to previous sections, we consider optimization problems where the objective function is a multivariate polynomial of degree d and a set of polynomial equalities and inequalities with (some of) the variables being binary. We focus on pure binary quadratic and cubic polynomial programs, however the proposed method is targeted to solve general mixed-binary polynomial programs. The success and the computational efficiency of the branch-and-bound procedure strongly depends on the quality of the relaxation bounds, the early generation of good binary feasible solutions, and the branching rules used to obtain the subproblems [68]. Applying the dynamic inequality generation scheme can help speed up the branch-and-bound process by improving the bounds at each node, thus reducing the number of nodes of the tree, we will refer to this approach as branch-and-dig.

The following sections describe the branch-and-bound algorithm that we implemented to solve binary polynomial programming problems. The general algorithm, node selection strategies, branching rules, and integration with DIGS-B are described in details. We also provide computational results on BQPP and cubic BPP where we compare various bounding schemes and branching rules in the branch-and-dig framework.

Similar to previous sections, we let $S = D \cap H$ be the feasible set of the binary polynomial program where $D = \{x \in \mathbb{R}^n : g_i(x) \geq 0, i = 1, \ldots, m\} \subseteq [-1, 1]^n$ and $H = \{x \in \mathbb{R}^n : x \in$

$\{-1, 1\}^n\}$. Each node N_G of the branch-and-bound tree is defined by a set of polynomial equalities and inequalities. Let $G_0 = \{g_i(x) : i = 1 \dots m\}$, N_{G_0} is the root node of the branch-and-bound tree.

5.1 Bounding Function

The bounding function is the key component of any branch-and-bound algorithm. To obtain a bound at node N_G, we use the approximation $\mathbb{K}^r_G$. Any approximation $\mathcal{K} \subseteq \mathcal{P}_d(S)$ can be used and there is a trade off between quality of the bound and computational time when choosing the approximation $\mathcal{K}$. The approximation $\mathbb{K}^r_G$ is used throughout this section for illustration purposes. Therefore, at a given node N_G we solve the following problem:

$$\begin{aligned} \textbf{(BPP-N}_G\textbf{)}\ \mu^r_G = \quad & \min\ \lambda \\ & \text{s.t. } \lambda - f(x) \in \mathbb{K}^r_G. \end{aligned} \tag{5.1}$$

In this case, μ^r_G is an upper bound on the optimal solution of (BPP-P).

5.2 Branching Rules

At each node of the branch-and-bound tree, N_G, we branch by adding equality or inequality constraints to obtain two children nodes, i.e., adding polynomials to the set G to obtain G_1 and G_2. For (G_1, G_2) to be a branching rule, the following conditions need to be satisfied:

1. $G \subseteq G_1, G_2$,
2. $\{x : g(x) \geqq 0\ \forall g \in G\} = \{x : g(x) \geqq 0\ \forall g \in G_1\} \cup \{x : g(x) \geqq 0\ \forall g \in G_2\}$,

where the symbol $\geqq$ refers to equalities and inequalities. Each node in the tree is equal to its parent node plus additional constraints. Each of these nodes of the tree is the root of a subtree. Each node in the branch-and-bound tree provides an upper bound on the objective value of all feasible solutions in the subtree of this node.

To show that the upper bound is improving in the tree we consider a node N_G and its two children nodes, N_{G_1} and N_{G_2}.

Lemma 5.2.1. *Assume $G \subseteq G_1, G_2$, then*

$$\mu^r_G \geq \mu^r_{G_1} \text{ and } \mu^r_G \geq \mu^r_{G_2}.$$

Proof. Since $G \subseteq G_1, G_2$, then $\mathbb{K}^r_G \subseteq \mathbb{K}^r_{G_1}, \mathbb{K}^r_{G_2}$. □

Let

$$z_G = \max\ f(x) \quad \text{s.t. } x \in \{x : g(x) \geqq 0,\ \forall g \in G\}.$$

Lemma 5.2.2. *If $\bar{\mu}_{G_1} \geq z_{G_1}$ and $\bar{\mu}_{G_2} \geq z_{G_2}$, then $\max(\bar{\mu}_{G_1}, \bar{\mu}_{G_2}) \geq z_G$.*

Proof. The statement follows from condition 2 of (G_1, G_2) being a branching rule. □

Applying Lemma 5.2.2 inductively, we obtain the following corollary:

Corollary 5.2.3. *The maximum bound over the leaves of the branch-and-bound tree is a global upper bound of (BPP-P).*

Since we are dealing with polynomial programs, the constraints forming the branching rules need not be linear. In the branching rules we consider, we branch by adding one equality constraint $p_1(x)$ for the left child and one equality constraint $p_2(x)$ for the right child. Therefore, $G_1 = G \cup \{p_1\}$ and $G_2 = G \cup \{p_2\}$. We present three different branching rules that can be applied to BPPs. The three branching rules that we consider in this section are:

Branching on a Single Variable: The first branching rule is by branching on a variable x_j to two subproblems with $x_j = 1$ or $x_j = -1$. This is done by adding a constraint $1 + x_j = 0$ for the left subproblem and $1 - x_j = 0$ for the right subproblem. So given the parent node, N_G, the child nodes are N_{G_1} and N_{G_2} such that $G_1 = G \cup \{1 + x_j\}$ and $G_2 = G \cup \{1 - x_j\}$.

Since the variables are binary, the set of possible configurations of the variables is finite and equal to 2^n thus the branch-and-bound procedure stops after a finite number of nodes.

Branching on Two Variables: Linear Case: Next, we describe branching on two variables x_j and x_l to two subproblems with $x_i + x_j = 0$ or $x_i - x_j = 0$. The child nodes in this case are N_{G_1} and N_{G_2} where $G_1 = G \cup \{x_i + x_j\}$ and $G_2 = G \cup \{x_i - x_j\}$.

Note that in this case the branch-and-bound approach might not guarantee an integral solution. However, this can be overcome by branching on $x_j = \pm 1$ after the algorithm presents no change in the solution and in the corresponding upper bound when branching on $x_i \pm x_j = 0$.

Branching on Two Variables: Non-linear Case: Similar to the previous point, we branch on two variables x_i and x_j to obtain two subproblems but with non-linear constraints: $x_i x_j = 1$ or $x_i x_j = -1$. The child nodes in this case are N_{G_1} and N_{G_2} where $G_1 = G \cup \{x_i x_j + 1\}$ and $G_2 = G \cup \{x_i x_j - 1\}$.

5.3 Feasible Solution

During a branch-and-bound procedure, it is of particular importance to find a feasible solution to be able to fathom nodes in the tree reducing the search space. Unfortunately, it is hard to obtain a feasible solution for general BPP since it is already hard to find feasible solutions for general binary linear programs [68]. In our branch-and-bound algorithm, we overcome this problem by rounding the optimal dual solution Y.

Let Y be the optimal dual solution of (5.1). At a given node, we set $\bar{x}$ to be the sign of the terms of Y that correspond to the linear monomials. Since $\bar{x}$ is already binary, we check whether $\bar{x}$ is feasible or not. If $g_i(\bar{x}) \geq 0$ for $i = 1, \ldots, m$ then $x_f = \bar{x}$ is a feasible

solution for (BPP-P) and one can find the corresponding objective value. The algorithm is described in details in Algorithm 4.

Algorithm 4: Generating a Feasible Solution

Input: $Y = (Y_\alpha)_\alpha$ where α is all the monomials up to degree d and $g_i(x)$
Output: x feasible solution for (BPP-P)

Let $\bar{x} = \text{sign}((Y_\alpha)_{|\alpha|=1})$;
if $g_i(\bar{x}) \geq 0 \qquad \forall i = 1, \ldots, m.$ **then**
    return $\bar{x}$;
else
    return Fail ;
end

Every feasible solution occurring during the branch-and-bound process provides a lower bound on the optimal objective value of (BPP-P).

5.4 Node Selection

The choice of the next node to solve affects the performance of the branch-and-bound tree. For the node selection we consider three options:

Depth First The next node to be solved of the branch-and-bound tree is one of the child nodes of the current node solved. Depth-first node selection goes deep into the branch and bound tree at each iteration, so it reaches the leaf nodes quickly. This is one way of achieving an early incumbent solution.

Breadth First All the nodes at each level of the branch-and-bound tree have to be considered before a node in a new level can be considered.

Worst Bound The next node to be selected in the branch-and-bound tree is the one with the greatest upper bound so far. This leads to reducing the gap between the upper and the lower bounds.

Choosing which node selection criterion to use depends on empirical studies and is application dependent [68]. In our computational results, we use breadth first strategy for node selection.

5.5 Inequality Generation Scheme

The branch-and-bound approach can be sped up considerably by the employment of an inequality generation scheme, either just at the root node of the tree, or at every node of the tree. Hence in Algorithm 5, one can add inequalities to improve the performance of the branch-and-bound method. Using DIGS-B, we produce polynomial inequalities that are valid for (BPP-P) and at the same time are intended to be violated by Y, the dual variable of the corresponding relaxation. The effectiveness of an inequality can be measured by its depth, $\langle p, Y \rangle$. The more negative $\langle p, Y \rangle$ the deeper the inequality is relative to Y. As discussed in Section 4.2, these inequalities are generated by solving a polynomial generating subproblem. Because of the expense, the inequalities are only generated at the root node, but since they are valid to all the children nodes, they are added to each node of the tree. At the root node, valid inequalities are added until no significant improvement in the bound is obtained.

5.6 Branch-and-Dig Algorithm

Before describing the branch-and-bound algorithm, we discuss the conditions for eliminating a node from further consideration, i.e., fathoming the node. A node of the branch-and-bound tree is fathomed in one of the following cases:

- If the objective value of a node is the same as the objective value of $\bar{x}$ obtained from Algorithm 4, the children of this node will not produce better solutions and the subtree must not be searched.
- If the objective value of a node is less than or equal to the current incumbent (the best lower bound), we do not need to branch further on this node, since all possibly

feasible solutions in the subtree will not provide better objective values than the current feasible solution.

- If a node problem is unbounded since this implies that the primal problem of the relaxation of (BPP-P) is infeasible and hence all primal subproblems will be infeasible too.

Given $f(x)$, $g_i(x)$, and the binary index set $J \subseteq \{1, \ldots, n\}$, a sketch of the branch-and-bound algorithm is given as follows:

Algorithm 5: Branch-and-Bound Algorithm for Binary Polynomial Programs

Input: $f(x)$, $G := \{g_i(x) : i = 1, \ldots, m\}$, $J \subseteq \{1, \cdots, n\}$

Output: x_{BPP} and z_{BPP} optimal solutions for (BPP-P)

set: $G_0 = G \cup \{1 - x_i^2 : i \in J\}$, $\mu_{G_0} = \infty$, $z_{BPP} = -\infty$, $x_G = [\,]$, $Nodes = \{N_{G_0}\}$
while $Nodes \neq \phi$ **do**
    Choose node $N_G \in Nodes$;
    set: $Nodes = Nodes \backslash N_G$;
    Solve (BPP-N$_G$) to obtain μ_G and Y_G;
    if $\mu_G = -\infty$ **then**
        Fathom N_G;
    else if $\mu_G \leq z_{BPP}$, **then**
        Fathom N_G;
    else
        Apply Algorithm 4 to Y_G to obtain x_G;
        if $f(x_G) \geq z_{BPP}$ **then**
            $z_{BPP} = \mu_G$, $x_{BPP} = x_G$;
        end
        if $\mu_G = f(x_G)$ **then**
            Fathom N_G;
        else
            Choose (G_1, G_2) as a branching rule for G;
            set: $Nodes = Nodes \cup \{N_{G_1}, N_{G_2}\}$;
        end
    end
end

5.7 Computational Results

In this section we describe application problems that are used to test the branch-and-bound algorithm. The test instances considered in this section are quadratic knapsack, quadratic assignment, and cubic problems. For the branch-and-bound algorithm, we use breadth first branching strategy with a limit of 1000 nodes for the branch-and-bound tree. Based on these instances we compare in the branch-and-bound and the branch-and-dig algorithms. For the branch-and-dig algorithm we add valid inequalities at the root node using the DIGS-B scheme discussed in Section 5.5. The number of inequalities added depend on the bound improvement and the subproblem objective function value. In this case, we keep adding inequalities as long as the improvement of the bound is greater than 0.5% and the subproblem objective is greater than -10^{-3}. Additionally, we compare with respect to different branching rules in the framework of the branch-and-bound approach. The branch-and-dig algorithm is implemented in Matlab and the SOC and/or the SDP relaxations are constructed using APPS (see Chapter 6) and solved using SeDuMi [93].

For BPPs with degree 2, we use the SOC-based relaxation presented in Section 3.1.2 and add the branching constraints for each subproblem hence obtaining SOC relaxations. The computational results presented in Section 5.7 for the quadratic knapsack, and quadratic assignment problem are based on SOC relaxations. While the results for the cubic BPP are based on SOC and SDP relaxations.

5.7.1 QKP Instances

Recall the QKP problem (QKP-P) defined in Section 3.2.4, we use (QKP_{SOC}) relaxation as a bounding function. The instances used in this section are generated using the approach presented in Example 4.3.4.

Tables 5.1-5.3 provide computational results for QKP instances using different branching rules. In Table 5.1 we study branching on $x_j = \pm 1$, Table 5.2 we present results when branching on two variables using the linear branching rule $x_i \pm x_j = 0$, and Table 5.3 we present branching on two variables using the quadratic branching rule $x_i x_j \pm 1$. For

each case, we also study the effect of applying valid inequalities at the root node of the branch-and-dig tree.

It is clear from the results that the number of nodes of the branch-and-bound tree are reduced once DIGS-B is applied at the root node. For example, the average reduction in the number of nodes is 16% for Table 5.1, 20% for Table 5.2, and 7% for Table 5.3. The impact of the reduction of the number of nodes is not directly translated to savings in the computational time. This is due to the DIGS-B subproblem computational time. The second observation is that branching on $x_j = \pm 1$ and $x_i \pm x_j = 0$ is much better than branching on $x_i x_j = \pm 1$ for this particular problem. A possible explanation is that when the quadratic branching rules of the form $x_i x_j = \pm 1$ are added, the polynomial multiplier in this case is a free variable in $\mathbf{R}_0$ while using linear branching rules like $x_j = \pm 1$ and $x_i \pm x_j = 0$ the polynomial multiplier is a degree one polynomial of free coefficient variables. Adding a linear polynomial multiplier, $p(x)$, seems to perform better than a constant multiplier, c, since having $p(x)(1 \pm x_j)$ or $p(x)(x_i \pm x_j)$ provides more freedom in the polynomial representation than $cx_i x_j$. See also Example 4.3.7 in Section 4.3.2.

Table 5.1: QKP: Branching on $x_j = 1$ and $x_j = -1$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
10	1653.0	1789.9	6.1	13	0	1711.3	17.3	9	0	8
15	2022.0	2042.7	30.2	63	0	2040.7	22.2	11	0	2
20	8510.0	8690.1	36.2	53	0	8683.1	47.4	41	0	2
25	8806.0	8974.1	257.5	205	0	8962.6	253.3	197	0	2
30	18229.0	18626.7	212.6	99	0	18613.3	286.6	89	0	2
35	25457.0	25717.6	76.9	15	0	25707.2	97.3	15	0	2
40	2679.0	2746.6	546.6	79	0	2743.9	551.3	71	0	2
45	42494.0	43054.2	1877.4	169	0	43045.8	1955.1	163	0	2
50	16192.0	16741.0	8144.9	1000	3.01	16726.8	10280.9	1000	3.01	2
55	19594.0	19929.1	4601.4	405	0	19905.8	5056.8	391	0	2
60	58451.0	58809.1	5819.6	403	0	58778.5	5078.8	295	0	2
65	33878.0	33878.0	133.7	1	0	33878.0	133.7	1	0	0
70	16982.0	17135.5	10072.8	359	0	17111.5	11444.9	301	0	2

Table 5.2: QKP: Branching on $x_i + x_j = 0$ and $x_i - x_j = 0$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
10	1653.0	1789.9	6.8	15	0	1711.3	16.5	11	0	8
15	2022.0	2042.7	17.9	33	0	2040.7	23.1	11	0	2
20	8510.0	8690.1	31.6	39	0	8683.1	70.6	31	0	2
25	8806.0	8974.1	288.9	201	0	8962.6	223.2	159	0	2
30	18229.0	18626.7	179.8	77	0	18613.3	209.0	75	0	2
35	25457.0	25717.6	201.7	35	0	25707.2	120.3	15	0	2
40	2679.0	2746.6	482.4	63	0	2743.9	363.3	55	0	2
45	42494.0	43054.2	1734.4	137	0	43045.8	1490.2	129	0	2
50	16534.0	16741.0	4735.9	601	0	16726.8	5252.8	577	0	2
55	19588.0	19929.1	4050.9	319	0	19905.8	5674.5	315	0	2
60	58451.0	58809.1	4261.0	277	0	58778.5	6775.4	227	0	2
65	33878.0	33878.0	134.7	1	0	33878.0	134.7	1	0	0
70	16982.0	17135.5	7486.1	271	0	17121.2	7306.0	193	0	2

5.7.2 QAP Instances

Recall the QAP problem (QAP-P) defined in Section 3.2.2, we use (QAP_{SOC}) relaxation as a bounding function however we do not include the inequalities $-1 \leq x_{ij}x_{kl} \leq 1$ to obtain faster relaxations. Instead for the branch-and-dig algorithm we add valid inequalities as discussed in Section 5.5. The instances presented in this section generated using the same approach discussed in Example 4.3.5 but the density in this case is 100%.

Tables 5.4-5.6 present computational results for the branch-and-bound and branch-and-dig algorithms for the QAP problem using the three different branching rules. Table 5.4 uses a single variable and linear branching rule $x_j = \pm 1$, Table 5.5 uses two variables and a linear branching rule $x_i \pm x_j = 0$, and finally Table 5.6 uses two variables and a quadratic branching rule $x_i x_j = \pm 1$.

A similar conclusion as the QKP results can be extracted from these tables. The $x_i \pm x_j = 0$ branching rule perform the best in terms of number of nodes and computational time.

Table 5.3: QKP: Branching on $x_i x_j = -1$ and $x_i x_j = 1$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
10	1653	1789.9	67.4	43	0	1711.3	92.4	23	0	8
15	2022	2042.7	50.7	17	0	2040.7	65.6	17	0	2
20	8510	8690.1	546.3	189	0	8683.1	695.7	183	0	2
25	8806	8974.1	4106.8	1000	1.39	8962.6	4232.6	1000	1.38	2
30	18229	18626.7	8010.2	1000	1.42	18613.3	9008.2	1000	1.31	2
35	25457	25717.6	1377.8	85	0	25707.2	1955.1	83	0	2
40	2679	2746.6	13481.0	545	0	2743.9	11068.5	337	0	2
45	42494	43054.2	44331.9	1000	0.95	43045.8	50162.3	1000	0.94	2
50	16192	16741.0	16062.2	1000	3.18	16726.8	23788.5	1000	3.16	2
55	19594	19929.1	22791.4	1000	1.30	19905.8	48574.0	1000	1.28	2
60	58451	58809.1	32330.3	1000	0.41	58778.5	63784.4	1000	0.40	2
65	33878	33878.0	124.0	1	0	33878.0	124.0	1	0	0
70	16982	17135.5	117973.2	1000	0.70	17121.2	120183.9	1000	0.68	2

Additionally, the linear branching rule $x_i = \pm 1$ performs better than the quadratic rule $x_i x_j = \pm 1$ as in the previous case. Adding valid inequalities enhances the performance of the branch-and-bound tree by reducing the number of nodes visited particularly for Tables 5.4 and 5.5. The reduction can be significant in some cases, for example in Table 5.4 the largest instance ($n = 7$) the reduction in the number of nodes is around 28% and in Table 5.5 the reduction is 36%. On average the reduction in the number of nodes is 37%, 43%, and 8% for Tables 5.4-5.6 respectively.

Table 5.4: QAP: Branching on $x_j = 1$ and $x_j = -1$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
4	62	55.0	19.5	6	0	61.3	34.9	3	0	5
5	116	93.2	63.5	15	0	106.4	86.8	9	0	8
6	272	249.3	425.1	19	0	259.9	438.5	13	0	4
7	292	239.3	8620.9	135	0	252.3	7880.4	97	0	5

Table 5.5: QAP: Branching on $x_i + x_j = 0$ and $x_i - x_j = 0$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
4	62	55.0	21.6	6	0	61.3	41.3	3	0	5
5	116	93.2	66.2	15	0	106.4	89.1	9	0	8
6	272	249.3	406.2	13	0	259.9	305.0	7	0	4
7	292	239.3	7825.3	105	0	252.3	6110.9	67	0	5

Table 5.6: QAP: Branching on $x_i x_j = -1$ and $x_i x_j = 1$

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
4	62	55.0	19.0	4	0	61.7	55.3	3	0	7
5	116	93.2	324.1	94	0	107.4	987.0	88	0	10
6	272	249.3	21154.8	1000	2.45	259.9	20694.3	1000	1.74	4
7	292	239.3	73038.6	1000	15.22	252.3	87066.3	1000	12.50	5

5.7.3 Cubic BPP

Recall the BPP Problem with cubic objective function given in Example 4.3.8:

$$\begin{aligned} \max & \sum_{|\alpha| \leq 3} c_\alpha x^\alpha \\ \text{s.t. } & a^T x \leq b \\ & x \in \{-1, 1\}^n. \end{aligned}$$

For the branch-and-bound results we use two types of relaxations. The first is (CBPP_{L_1}) Lasserre's first order SDP-based relaxation [54] and the other relaxation is a new SOC-

based relaxation. Let $q(x) = \sum_{|\alpha|\leq 3} c_\alpha x^\alpha$, the SOC relaxation is given as follows:

$$\begin{aligned}
\textbf{(CBPP}_{\textbf{soc}}\textbf{)} \min\ & \lambda \\
\text{s.t. } \lambda - q(x) = & \sum_i (b - a^T x)(1 + x_i) {d_i^+}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i (b - a^T x)(1 - x_i) {d_i^-}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_{i<j} (1 + x_i x_j) {h_{ij}^+}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i<j} (1 - x_i x_j) {h_{ij}^-}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} \\
& + \sum_i (1 + x_i) {f_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_i (1 - x_i) {g_i}^T \begin{pmatrix} \sqrt{n} \\ x \end{pmatrix} + \sum_{i=1}^n c_i (1 - x_i^2) \\
c_i \in \mathbb{R}, \qquad & d_i^+, d_i^-, f_i, g_i, h_{ij}^+, h_{ij}^- \in \mathcal{L}^{n+1}.
\end{aligned}$$

The instances of this section are the same instances of Example 4.3.8. Tables 5.7-5.9 provide the branch-and-bound and the branch-and-dig results for the cubic BPP problem using two different branching rules and two different relaxations. The first branching rule branches on x_j being 1 or -1 and is presented in Tables 5.7 and 5.9. The second branching rules is quadratic, branching on $x_i x_j$ being -1 or 1 and it is presented in Table 5.8. We also present two different relaxations, Lasserre's first order SDP-based relaxation results are presented in Tables 5.7 and 5.8 while the SOC relaxation results are presented in Table 5.9.

Unlike the quadratic problems discussed earlier, in this case using the quadratic branching rule performs better than the linear as shown in Tables 5.7 and 5.8 particularly when the problem size is large. A possible explanation for this difference is that the polynomial multiplier is no longer a constant but is a polynomial in $\mathbf{R}_1[x]$. Additionally, adding inequalities to the problem seems to improve in terms of reducing the number of nodes for all three tables. Although few inequalities are added the number of nodes is reduced for each instance. For example from the results of Table 5.7, the number of nodes is reduced by 37% on average over all the instances and from Table 5.8 the number of nodes is reduced by 43% on average. Finally, from Table 5.9, we notice that the SOC relaxation results in more nodes than the SDP relaxation for the cubic case since the bounds at the root node are weaker in this case as opposed to quadratic problems where the bound of the SOC

relaxation at the root node is strong. For cubic problems, we need to further investigate finding a stronger SOC relaxation than the one presented.

Table 5.7: CBPP: Branching on $x_j = 1$ and $x_j = -1$, (CBPP_{L_1})

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
5	58	67.2	14.2	3	0	57.9	18.4	1	0	1
10	139	154.6	91.8	7	0	138.7	126.4	3	0	3
15	1371	1582.0	1008.9	17	0	1391.1	1500.6	9	0	5
20	1654	1718.5	11559.6	29	0	1664.0	12876.1	13	0	4

Table 5.8: CBPP: Branching on $x_i x_j = 1$ and $x_i x_j = -1$, (CBPP_{L_1})

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
5	58	67.2	13.3	3	0	57.9	18.9	1	0	1
10	139	154.6	111.3	7	0	138.7	186.5	3	0	3
15	1371	1582.0	1225.1	13	0	1391.1	1992.7	9	0	5
20	1654	1718.5	10666.5	19	0	1664.0	11668.0	10	0	4

Table 5.9: CBPP: Branching on $x_j = 1$ and $x_j = -1$, (CBPP_{SOC})

n	Optimal	Branch-and-Bound				Branch-and-Dig				
		Root Obj.	T(sec)	Nodes	Gap	Root Obj.	T(sec)	Nodes	Gap	#Ineq.
5	58	76.3	16.5	5	0	59.9	27.6	3	0	5
10	139	388.2	229.5	31	0	357.6	357.7	29	0	5
15	1371	2359.8	1601.1	49	0	2333.6	2006.1	45	0	2
20	1654	5047.4	26365.1	1000	42.4	4781.5	35668.3	1000	39.8	2

5.8 Concluding Remarks

In this chapter, we presented an algorithm to find global solutions for mixed binary polynomial programming problems using a non-linear branch-and-bound method. Inequality generating techniques based on a hierarchy of lift-and-project relaxations of the binary feasible set were developed for binary polynomial problems. For the generation of non-linear inequalities, a sequence of second order cone or positive semidefinite programs are solved.

An implementation of the polynomial programming solver are used in the branch-and-bound and the branch-and-dig framework. Computational results for small test problems and application problems are given. In the computational study we investigate the performance of different branching rules and the impact of the inequality generation technique, DIGS-B. In the context of branch-and-dig we observed a reduction of the search trees of most of the test problems when DIGS-B was applied. Thereby, the non-linear inequalities were able to reduce the search trees of most of the problems and led also to the best reductions with respect to all test instances.

A drawback of the encouraging reductions of the search trees by DIGS-B are the high computational costs due to the polynomial generating subproblems. For higher degree BPPs, we can investigate the performance of different types of inequalities with different degrees and different sparsity. We can also use a heuristic to generate a feasible solution earlier in the branch-and-bound tree based on Lasserre's solution extraction for PP [37].

Another way to improve the implementation of the branch-and-dig algorithm is preprocessing at each node. That is instead of adding the constraints $x_j = \pm 1$ for the child nodes, one can fix the variables to their corresponding value and thus reducing the number of monomials of the polynomial program. Once some variables have been fixed in this manner and unnecessary constraints have been eliminated, the problem dimension can be significantly reduced as we go further down in the tree.

Chapter 6

APPS: A Polynomial Programming Solver

APPS is a Matlab-based solver that builds and solves conic relaxations of polynomial programs. The techniques behind APPS are based on the earlier chapters of this thesis. To solve the conic relaxations, APPS calls SeDuMi or SDPT3 solvers. The interface has been designed to be as simple as possible while keeping a large degree of flexibility for the user. General features of APPS are listed below:

1. Builds and solves general conic relaxations of PPs.
2. Generates and adds valid inequalities for general and binary PPs using DIGS and/or DIGS-B.
3. Solves binary PPs to optimality.
4. Generates input and output data in SeDuMi format.

6.1 Formulating and Solving PP

Recall, we are interested in solving PPs of degree d of the form:

$$\begin{aligned} &\max f(x) \\ &\text{s.t. } g_i(x) \geqq 0 \qquad\qquad i = 1, \ldots, m. \end{aligned}$$

To define and solve a PP using APPS we need to follow these steps:

1. Declare the PP variables.
2. Initialize the PP.
3. Define the PP constraints.
4. Define the PP objective function.
5. Call conic solver.
6. Obtain solutions.

The key element of defining the PP is the structure `prog` with the following fields

Variables	Degree	Monomials	Conic Program	Polynomial Multipliers
`prog.vars`	`prog.maxDeg`	`prog.alpha`	`prog.Ased`	`prog.Free`
`prog.numVars`	`prog.totalDeg`	`prog.numAlpha`	`prog.bsed`	`prog.Linear`
			`prog.csed`	`prog.Soc`
			`prog.Ksed`	`prog.Sdp`

`prog` contains all the information necessary to build and solve the PP. For building the PP, we have `prog.vars` is a vector of `prog.numVars` variables. `prog.alpha` is a matrix whose rows are the degree vectors of all the monomials where the variable `prog.vars[i]` has a maximum degree `prog.maxDeg[i]` and total degree $\leq$ `prog.totalDeg`. `prog.numAlpha` is the number of monomials, i.e, the number of rows in `prog.alpha`. Some other information kept in `prog` is related to initializing the conic relaxation. In particular, `prog.Ased`, `prog.bsed`, `prog.csed`, and `prog.Ksed` are the elements of the conic program in SeDuMi

format. For instance, `prog.Ased` has `prog.numAlpha` rows where each row i corresponds to the coefficient of the monomial saved in `prog.alpha[i]`. `prog.Free`, `prog.Linear`, `prog.Soc`, and `prog.Sdp` are structures used to keep track of the constraints according to the type of polynomial multiplier to be used.

Next, we describe a small example to illustrate the basic use APPS . We consider non-convex quadratic problem of Example 4.1.4:

$$\begin{aligned}
\min_{x} \; & -2x_1 + x_2 - x_3 \\
\text{s.t.} \; & 24 - 20x_1 + 9x_2 - 13x_3 + 4x_1^2 - 4x_1x_2 + 4x_1x_3 + 2x_2^2 - 2x_2x_3 + 2x_3^2 \geq 0 \\
& x_1 + x_2 + x_3 \leq 4 \\
& 3x_2 + x_3 \leq 6 \\
& 0 \leq x_1 \leq 2, \qquad 0 \leq x_2, \qquad 0 \leq x_3 \leq 3.
\end{aligned}$$

The following Matlab script uses APPS to solve Lasserre's approximation using $\mathbb{K}_G^r$ with $r = 2$:

Matlab Code: General PP

```
%Creating the variables and the monomials up to degree r
n=3;
r=2;
x=varsVector('x',n);
prog=createProg(x,r,[r*ones(1,n)]);

%Adding sos polynomial multipliers
prog=add_sos(prog,1,r);
cS1=24-20*x(1)+9*x(2)-13*x(3)+4*x(1)^2-4*x(1)*x(2)...
      +4*x(1)*x(3)+2*x(2)^2-2*x(2)*x(3)+2*x(3)^2;
prog=add_sos(prog,cS1,r-2);
prog=add_sos(prog,4-x(1)-x(2)-x(3),r-1);
prog=add_sos(prog,6-3*x(2)-x(3),r-1);
prog=add_sos(prog,x(1),r-1);
prog=add_sos(prog,x(2),r-1);
prog=add_sos(prog,x(3),r-1);
prog=add_sos(prog,2-x(1),r-1);
prog=add_sos(prog,3-x(3),r-1);

%Adding objective function
q=-(-2*x(1)+x(2)-x(3));        %negative for minimization.
prog=add_objective(prog,q);

%Call solver
prog = prog_solve(prog);
```

The following describe the steps of the above script.

- The function `varsVector` creates the vector $x \in \mathbb{R}^n$:

  Matlab Code: Construct Variable Vector

  ```
  n=3;
  x=varsVector('x',n)
  ```

  Matlab Output: Construct Variable Vector

  ```
  x =
    [ x1 , x2 , x3 ]
  ```

- The function `createProg` initializes the polynomial program by taking as inputs: the vector `x`, the degree of the relaxation of the polynomial program r, and the maximum degree of each variable (usually this is r). This can be done by typing the following commands:

Matlab Code: Construct Prog

```
r=2;
n=3;
prog = createProg(x,r,[r*ones(1,n)]);
```

Matlab Output: Construct Prog

```
prog =
          vars: [3x1 polynomial]
       numVars: 3
        maxDeg: [2 2 2]
      totalDeg: 2
         alpha: [10x3 double]
      numAlpha: 10
          Ased: 0
          bsed: 0
          csed: 0
          Ksed: 0
          Free: [1x1 struct]
        Linear: [1x1 struct]
           Soc: [1x1 struct]
           Sdp: [1x1 struct]
```

- The functions `add_free`, `add_linear`, `add_soc`, and `add_sos` handle the addition of constraints by generating a polynomial multiplier of degree k to constraint $g(x)$ in $\mathbf{R}_k[x], \mathbf{R}_k^+[x], \mathcal{P}_1(\mathcal{B})$, and $\Psi_k[x]$ respectively. Each function takes three inputs, the first input is `prog`, the second is the polynomial $g(x)$, and the third input is the degree of the polynomial multiplier which is optional (by default k is set to $r - \deg(g)$ where r is the degree of the relaxation). For `add_sos` function, $\Psi_k[x] = \Psi_{k-1}[x]$ for odd k and for `add_soc` the degree of the polynomial multiplier should be 1.

- The function `add_objective` adds the objective function of the PP.

- The function `prog_solve` generates A, b, and c matrices and the cone $\mathcal{C}$ of the conic program as defined in (2.2). The function also solves the conic program using SeDuMi (default) or SDPT3 and returns a primal-dual solution, `prog.primalSol` and `prog.dualSol` respectively, and the objective function value, `prog.obj`, which is equal to λ.

Next we describe the use of APPS to solve a binary PP. Recall the QKP problem described in Section 3.2.4. The following script describes how to code (QKP_{SS}) relaxation. Given Q, c, w and n we use the following Matlab script to solve the QKP problem using APPS :

Matlab Code: Binary PP

```
%Creating the variables and the monomials up to degree r
x = varsVector('x',n);
prog = createProg(x,r,[r*ones(1,n)]);
r=2;

%Adding sos polynomial multipliers
prog= add_sos(prog,1,r);

%Adding soc and free polynomial multipliers
for i = 1:n
    prog= add_free(prog,1-x(i)^2,r-2);
    prog= add_soc(prog,1+x(i),r-1);
    prog= add_soc(prog,1-x(i),r-1);
end

prog= add_soc(prog,c-w*x',r-1);

%Adding objective function
q=x*Q'*x';
prog=add_objective(prog,q);

%Call solver
prog = prog_solve(prog);
```

For a quadratic knapsack problem of 10 variables, we obtain the following result:

Matlab Output: Binary PP

```
prog =
         vars: [10x1 polynomial]
      numVars: 10
       maxDeg: [2 2 2 2 2 2 2 2 2 2]
     totalDeg: 2
        alpha: [66x10 double]
     numAlpha: 66
         Ased: [66x163 double]
         bsed: [66x1 double]
         csed: [163x1 double]
         Ksed: [1x1 struct]
    primalSol: [163x1 double]
         Free: [1x10 struct]
       Linear: [1x1 struct]
          Soc: [1x1 struct]
          Sdp: [1x21 struct]
          q_x: [1x1 polynomial]
         info: [1x1 struct]
          obj: 1.8011e+03
      dualSol: [66x1 double]
```

The objective function value is 1801.1, the dual optimal solution, and the total cpu time are stored in `prog.obj`, `prog.dualSol`, and `prog.info.totaltime` respectively.

6.2 Inequality Generation

In this section we describe how to solve the subproblem and generate a valid polynomial inequality of degree r. Recall the generating subproblem of DIGS-B presented in

Section 4.2.1. The subproblem is of the form

$$\begin{aligned} &\min \ \langle p_i, Y^i \rangle \\ &\text{s.t. } p_i(x) \in \big((1+x_j)\mathcal{K} + (1-x_j)\mathcal{K} + (1-x_j^2)\mathbf{R}_{r-1}[x]\big) \cap \mathbf{R}_r[x] \\ &\quad \sum_{|\alpha|>0} (p_i)_\alpha^2 \leq 1, \end{aligned}$$

where i is the iteration number, j is the index of x chosen, and Y is `prog.dualSol` the optimal dual solution of master problem. The following is an implementation of Algorithm 3 of DIGS-B.

Matlab Code: DIGS-B

```
weight=ones(1,n);
while(iter < iterlimit & cputime-t < timelimit & objSubi < -1e-3)
        [indexi weight]=index_select(prog,weight);
        [digi objSubi timeSubi] = prog_createdigs(prog,indexi);
        %transform from vector to polynomial
        poly_digi=vec2pol(digi,prog);
        prog= add_sos(prog,poly_digi,0);
        prog=prog_solve(prog);
        iter=iter+1
end
```

- The function `index_select` generates an index based on the heuristic described in Section 4.2.1.
- The function `prog_createdigs` implements line 4 of Algorithm 3. It generates a vector of coefficients `digi` which is then transformed into a polynomial `poly_digi` using `vec2pol` function.

The inequality is then added to the set of constraints, using `prog= add_sos(prog,poly_dig,0)` and the master problem is then solved again using `prog=prog_solve(prog)`.

Performing one iteration of DIGS-B:

Matlab Output: DIGS-B

```
          obj: [1.8011e+03  1.7879e+03]
      dualSol: [66x1 double]
        index: 3
       Subobj: -0.7930
         info: [1x1 struct]
```

The upper bound improved from 1801.1 to 1787.9 and the subproblem objective value is -0.793 and the index j chosen for the subproblem is 3. In case it is needed to generate a polynomial inequality of degree less than r, the following script is used:

Matlab Code: DIGS-B (inequality of degree $< r$)

```
weight=ones(1,n);
while(iter < iterlimit & cputime-t < timelimit & objSubi < -1e-3)
        [indexi weight]=index_select(prog,weight);
        [digi objSubi timeSubi] = prog_binarydigs(prog,indexi,deg);
        %transform from vector to polynomial
        poly_digi=vec2pol(digi,prog);
        prog= add_sos(prog,poly_digi,r-deg);
        prog=prog_solve(prog);
        iter=iter+1
end
```

To apply DIGS, first one needs to construct the subproblem. Recall Algorithm 2 in Section 4.1.1). The subproblem has the form:

$$\begin{aligned} &\min \ \langle p_i, Y^i \rangle \\ &\text{s.t. } p_i(x) \in \mathbb{K}_G^{r+2} \\ &\qquad \sum_{|\alpha|>0} (p_i)_\alpha^2 \leq 1, \end{aligned}$$

To apply DIGS for Example 4.1.4, we set $r = 2$ and generate quadratic inequalities. First we need to build the subproblem which is in $\mathbb{K}_G^4$ in this case, then generate the quadratic

inequalities, and add them to the master and subproblem. `prog` stores information of the master problem and `prog_sub` stores those of the subproblem. Algorithm 2 of DIGS can be called using the following script:

Matlab Code: DIGS

```
t=cputime;
r2=r+2;
prog_sub=createsubproblem(prog,r2);
while(iter < iterlimit & cputime-t < timelimit & objSubi < -1e-3)
        [prog_sub digi] = prog_generaldigs(prog_sub,prog.dualSol);
        %transform from vector to polynomial
        poly_digi=vec2pol(digi,prog);
        prog=  add_sos(prog,poly_digi,0);
        prog_sub= add_sos(prog_sub,poly_digi,r2-r);
        prog=prog_solve(prog);
        iter=iter+1
end
```

Performing one iteration of DIGS:

Matlab Output: DIGS

```
          obj: [-6.0000 -5.8746]
      dualSol: [10x1 double]
       Subobj: -1.1621
         info: [1x1 struct]
```

Therefore, the lower bound improves from -6.00 to -5.87.

6.3 Concluding Remarks

APPS is a general-purpose software that provides bounds and solves a wide range of non-convex polynomial programs. It constructs and solves Linear, SOC, and/or SDP relaxations for PPs and strengthens these relaxations by adding valid inequalities. Further, for binary PPs, APPS finds optimal solutions by using a branch-and-dig algorithm.

Chapter 7

Conclusion and Future Directions

In this dissertation, we have discussed several theoretical and computational aspects related to solving general polynomial programs and in particular binary polynomial programs.

In Chapter 3, we introduced several conic relaxations for binary polynomial programs that results in SOC and/or SDP relaxations. The SOC-based relaxations for binary quadratic polynomial programs provide comparable bounds to SDP-based relaxations and are computationally efficient. On the other hand, we presented SOC-SDP-based relaxations that provide stronger bounds than the current practical SDP-relaxations presented in the literature. Computational Results on four quadratic applications were provided.

Chapter 4 then proceeded to introduce a dynamic inequality generation scheme for polynomial programs. We further specialized the scheme to binary polynomial programs. We showed theoretically that for special cases the iterative scheme for the binary case converges to the optimal solution of the original BPP. We considered several applications for quadratic and cubic problems and we showed that adding valid inequalities to the SDP-based relaxations of the binary polynomial programs is more computationally efficient than applying Lasserre's hierarchy and substantially reduces the optimality gap. The iterative scheme can also be applied to SOC-based relaxations to strengthen them further as done in the Chapter 5. The main improvements that can be done to the performance of the iterative scheme are in terms of choosing the index for the subproblem and solving the

subproblem more efficiently. In future work one can investigate the effect of generating sparse inequalities on the improvement of the bounds and running times.

Chapter 5 considered integrating the conic relaxations with the iterative scheme to solve binary polynomial programs to optimality in a branch-and-bound framework. The second-order conic or semidefinite programming relaxations can be further strengthened by adding valid inequalities using DIGS-B. This can be observed in the reduction of the search space of the resulting branch-and-dig algorithm. Computational results confirm that the number of nodes of the branch-and-bound algorithm are reduced once these inequalities are added. In the context of branch-and-dig we observed a remarkable reduction of the search tree on most of the test problems when the dynamic inequality generation scheme is applied. The average reduction in the number of nodes ranges between 7% and 43% for the test instances used. Thereby, the non-linear inequalities based on BPP problems were able to reduce the search trees of most of the problems. In the computational study we investigated the impact of the dynamic scheme and different branching rules. A possible challenge for the future is the attempt to translate the search tree reductions induced by the dynamic scheme into an improvement of the running time. One the one hand, the solving process for the generating subproblems must be improved. This could be tried either by reducing the size of these problems or by the application of fast convergent solvers. On the other hand, investigating different branching rules are also a topic for future research which can include general branching for general PPs. Finally extracting a feasible solution early in the tree might enhance the performance of the branch-and-dig algorithm.

Finally, Chapter 6 presented an implementation of APPS , a Matlab based solver for polynomial programs. The solver utilizes the ideas used in the Chapters 3, 4, and 5 to provide bounds and optimal solutions for polynomial programs. APPS is used for all the computational results presented in this thesis. For the sake of fair comparisons between our proposed approaches and the results from the literature APPS is also used to construct and solve the relaxations presented in the literature.

References

[1] B. Alidaee, G. Kochenberger, and A. Ahmadian. 0-1 quadratic programming approach for the optimal solution of two scheduling problems. *International Journal of Systems Science*, 25:401–408, 1994. 37

[2] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming*, 95(1):3–51, 2003. 8

[3] M.F. Anjos, A. Kennings, and A. Vannelli. A semidefinite optimization approach for the single-row layout problem with unequal dimensions. *Discrete Optimization*, 2(2):113–122, 2005. 37

[4] M.F. Anjos and A. Vannelli. Computing globally optimal solutions for single-row layout problems using semidefinite programming and cutting planes. *INFORMS Journal on Computing*, 20(4):611–617, 2008. 10

[5] Y.Q. Bai, E. de Klerk, D.V. Pasechnik, and R. Sotirov. Exploiting group symmetry in truss topology optimization. *Optimization and Engineering*, 10(3):331–349, 2009. 3

[6] E. Balas, S. Ceria, and G. Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0-1 programs. *Mathematical Programming*, 58:295–324, 1993. 4, 76, 95, 96, 111

[7] M. Bellare and P. Rogaway. The complexity of approximating a nonlinear program. *Mathematical Programming*, 69:419–441, 1995. 25

[8] A. Ben-Tal and A. Nemirovskii. *Lectures on modern convex optimization: analysis, algorithms, and Engineering Applications*, volume 2. MPS/SIAM Series on Optimization, SIAM, Philadelphia, 2001. 8

[9] I. Bomze and E. de Klerk. Solving standard quadratic optimization problems via linear, semidefinite and copositive programming quadratic programs. *Journal of Global Optimization*, 24(2):163–165, 2002. 26

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

[11] E. Boros and P. Hammer. Pseudo-boolean optimization. *Discrete Applied Mathematics*, 123(1-3):155–225, 2002. 37

[12] E. Boros and A. Prekopa. Probabilistic bounds and algorithms for the maximum satisfiability problem. *Annals of Operations Research*, 21:109–126, 1989. 37

[13] S. Burer. Private communication. 55, 69, 70

[14] R.E. Burkard, S.E. Karisch, and F. Rendl. QAPLIB - a quadratic assignment problem library. *Journal of Global Optimization*, 10, 1997. 67

[15] E. de Klerk. The complexity of optimizing over a simplex, hypercube or sphere: a short survey. *Central European Journal of Operations Research*, 2008. 23, 26, 27

[16] E. de Klerk. Exploiting special structure in semidefinite programming: A survey of theory and applications. *European Journal of Operational Research*, 201(1):1–10, 2010. 3

[17] E. de Klerk, G.E.E Elfadul, and D. den Hertog. Optimization of univariate functions on bounded intervals by interpolation and semidefinite programming. Technical Report 2006-26, Tilburg University, Center for Economic Research, 2006. 21

[18] E. de Klerk, M. Laurent, and P. Parrilo. A ptas for the minimization of polynomials of fixed degree over the simplex. *Theoretical Computer Science*, 361(2):210–225, 2006. 26

[19] E. de Klerk, D. Pasechnik, and A. Schrijver. Reduction of symmetric semidefinite programs using the regular*-representation. *Mathematical Programming*, 109(2-3):613–624, 2007. 3

[20] E. de Klerk and D.V. Pasechnik. Approximation of the stability number of a graph via copositive programming. *SIAM Journal on Optimization*, 12(4):875–892, 2002. 2

[21] E. de Klerk and R. Sotirov. Exploiting group symmetry in semidefinite programming relaxations of the quadratic assignment problem. *Mathematical Programming*, 122(2):225–246, 2010. 3, 49

[22] J.W. Dickey and J.W. Hopkins. Campus building arrangement using topaz. *Transportation Research*, 6, 1972. 47

[23] R. Duffin. Infinite programs. *In Linear inequalities and related systems, Annals of Mathematics Studies*, 38:157–170, 1956. 7

[24] A.N. Elshafei. Hospital layout as a quadratic assignment problem. *Operations Research Quarterly*, 28(1):167–179, 1977. 47

[25] C. A. Floudas, P. M. Pardalos, C. S. Adjiman, W. R. Esposito, Z. H. Gümüs, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, Dordrecht, 1999. 84

[26] K. Fujisawa, M. Kojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997. 46

[27] G. Gallo, P.L. Hammer, and B. Simeone. Quadratic knapsack problems. *European Journal of Operational Research*, 12:132–149, 1980. 59

[28] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979. 24, 27

[29] K. Gatermann and P. Parrilo. Symmetry groups, semidefinite programs, and sums of squares. *Journal of Pure and Applied Algebra*, 192(1-3):95–128, 2004. 3

[30] A.M. Geoffrion and G.W. Graves. Scheduling parallel production lines with changeover costs: Practical applications of a quadratic assignment/LP approach. *Operations Research*, 24:595–610, 1976. 47

[31] B. Ghaddar, M.F. Anjos, and F. Liers. A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem. *Annals of Operations Research*, November 2008. Available online, DOI: 10.1007/s10479-008-0481-4. 10

[32] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6):1115–1145, 1995. 27

[33] M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3):493–513, 1988. 37

[34] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. 11

[35] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2000. 12

[36] Christoph Helmberg, Franz Rendl, and Robert Weismantel. A semidefinite programming approach to the quadratic knapsack problem. *Combinatorial Oprimization*, 4(2):197–215, 2000. 61, 63, 71, 72

[37] D. Henrion and J. B. Lasserre. Detecting global optimality and extracting solutions in GloptiPoly. *LAAS-CNRS Research Report No. 03541*, 29(2), 2003. 132

[38] D. Henrion and J. B. Lasserre. GloptiPoly: global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2), 2003. 103

[39] D. Hilbert. Uber die darstellung definiter formen als summe von formenquadraten. *Mathematische Annalen*, 32, 1888. 14

[40] L. Hubert. *Assignment methods in combinatorial data analysis*, volume 73. Statistics: Textbooks and Monographs Series, Marcel Dekker, 1987. 47

[41] P. Hungerländer and F. Rendl. Semidefinite relaxations of ordering problems. Technical report, Alpen-Adria-Universität Klagenfurt, August 2010. 11

[42] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–151, 1993. 60

[43] S. Kim and M. Kojima. Second-order cone programming relaxation of nonconvex quadratic optimization problems. *Optimization Methods and Software*, 15(3-4):201–224, 2001. 12

[44] S. Kim and M. Kojima. Exact solutions of some nonconvex quadratic optimization problems via SDP and SOCP relaxations. *Computational Optimization and Applications*, 26(2):143–154, 2003. 12

[45] S. Kim, M. Kojima, M. Mevissen, and M. Yamashita. Exploiting sparsity in linear and nonlinear matrix inequalities via positive semidefinite matrix completion. *To appear in Mathematical Programming*, 2009. 3

[46] S. Kim, M. Kojima, and Ph. Toint. Recognizing underlying sparsity in optimization. *Mathematical Programming*, 9(2):273–303, 2009. 3

[47] K. Kobayashi, S. Kim, and M. Kojima. Correlative sparsity in primal-dual interior-point methods for LP, SDP and SOCP. *Applied Mathematics and Optimization*, 58(1):69–88, 2008. 3

[48] M. Kojima, S. Kim, and H. Waki. Sparsity in sums of squares of polynomials. *Mathematical Programming*, 103(1):45–62, 2003. 3

[49] M. Kojima and M. Muramatsu. A note on sparse SOS and SDP relaxations for polynomial optimization problems over symmetric cones. *Computational Optimization and Applications*, 42(1):31–41, 2009. 3

[50] T.C. Koopmans and M.J. Beckmann. Assignment problems and the location of economic activities. *Econometrica*, 25:53–76, 1957. 47

[51] Y. Kuo and H. Mittelmann. Interior point methods for second-order cone programming and or applications. *Computational Optimization and Applications*, 28(3):255–285, 2004. 10

[52] A.H. Land and A.G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):pp. 497–520, 1960. 117

[53] J. Lasserre. Global optimization problems with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11:796–817, 2001. 1, 2, 28, 29, 32, 76

[54] J.B. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization*, 12(3):756–769, 2001. 2, 4, 43, 76, 129

[55] J.B. Lasserre. Semidefinite programming vs. LP relaxations for polynomial programming. *Mathematics of Operations Research*, 27(2):347–360, 2002. 2

[56] D.J. Laughhunn. Quadratic binary programming with application to capital-budgeting problems. *Operations Research*, 18:454–461, 1970. 37

[57] M. Laurent. A comparison of the Sherali-Adams, Lovász-Schrijver and Lasserre relaxations for 0-1 programming. *Mathematics of Operations Research*, 28:470–496, 2001. 2

[58] M. Laurent. Semidefinite representations for finite varieties. *Mathematical Programming*, 109(Ser. A):1–26, 2007. 2

[59] M. Laurent and F. Rendl. *Handbook on Discrete Optimization*, volume 12, chapter Semidefinite programming and integer programming, pages 393–514. Elsevier, Amsterdam, Netherlands, 2005. 5, 10

[60] E. Lawler. The quadratic assignment problem. *Management Science*, 9:586–599, 1963. 37

[61] E. L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):pp. 699–719, 1966. 117

[62] A.S. Lewis and M.L. Overton. Eigenvalue optimization. *Acta Numerica*, 5:149–190, 1996. 8

[63] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1-3):193–228, 1998. 8, 10, 12

[64] E.M. Loiola, N.M. Maia de Abreu, P.O. Boaventura-Netto, P. Hahn, and T. Querido. A survey for the quadratic assignment problem. *European Journal of Operational Research*, 176(2):657–690, 2007. 47

[65] L. Lovász and A. Schrijver. Cones of matrices and set-functions and 0-1 optimization. *SIAM Journal on Optimization*, 1:166–190, 1991. 57, 76

[66] T. S. Motzkin. The arithmetic-geometric inequality. *O. Shisha, Inequalities*, pages 205–224, 1967. 14

[67] K.G. Murty and S.N. Kabadi. Some np-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39:117–129, 1987. 18

[68] G.L. Nemhauser and L.A. Wolsey. 117, 120, 122

[69] Y. Nesterov. Semidefinite relaxation and non-convex quadratic optimization. *Optimization Methods and Software*, 12:1–20, 1997. 27

[70] Y. Nesterov. Structure of non-negative polynomials and optimization problems. Technical report, Technical Report 9749, CORE, 1997. 2

[71] Y. Nesterov. Random walk in a simplex and quadratic optimization over convex polytopes. Core discussion papers, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2003. 23

[72] J. Nie. Regularization Methods for Sum of Squares Relaxations in Large Scale Polynomial Optimization. *eprint arXiv:0909.3551*, 2009. 2

[73] J. Nie and J. Demmel. Sparse SOS relaxations for minimizing functions that are summation of small polynomials. *SIAM Journal on Optimization*, 19(4):1534–1558, 2008. 3

[74] J. Nie, J. Demmel, and B. Sturmfels. Minimizing polynomials via sum of squares over the gradient ideal. *Mathematical Programming: Series A and B*, 106(3):587–606, 2006. 2

[75] P. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA., 2000. 2

[76] P. Parrilo. An explicit construction of distinguished representations of polynomials nonnegative over finite sets. Technical report, IFA Technical Report AUT02-02, Zurich - Switzerland, 2002. 2

[77] P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003. 2

[78] P.A. Parrilo and B. Sturmfels. Minimizing polynomial functions, algorithmic and quantitative real algebraic geometry. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 60:83–89, 2003. 2

[79] J. Peña, J. Vera, and L.F. Zuluaga. Computing the stability number of a graph via linear and semidefinite programming. *SIAM Journal on Optimization*, 18(1):87–105, February 2007. 108

[80] J. F. Peña, J. C. Vera, and L. F. Zuluaga. Exploiting equalities in polynomial programming. *Operations Research Letters*, 36(2), 2008. 2, 29, 30, 87, 89

[81] A.T. Phillips and J.B. Rosen. A quadratic assignment formulation of the molecular conformation problem. *Journal of Global Optimization*, 4:229–241, 1994. 37

[82] D. Pisinger. The quadratic knapsack problem-a survey. *Discrete Applied Mathematics*, 155(5):623–648, 2007. 71, 72, 105

[83] I. Pólik and T. Terlaky. A survey of the $\mathcal{S}$-lemma. *SIAM Review*, 49:371–418, 2007. 2, 24, 99

[84] M.A. Pollatschek, N. Gershoni, and Y.T. Radday. Optimization of the typewriter keyboard by simulation. *Angewandte Informatik*, 17:438–439, 1976. 47

[85] B. T. Polyak. Convexity of quadratic transformations and its use in control and optimization. *Journal of Optimization Theory and Applications*, 99:553–583, 1998. 23, 24

[86] V. Powers and B. Reznick. Polynomials that are positive on an interval. *Transactions of the American Mathematical Society*, 352:4677–4692, 2000. 21

[87] M. Putinar. Positive polynomials on compact semi-algebraic sets. *Indiana University Mathematics Journal*, 42:969–984, 1993. 17, 28, 30, 34

[88] F. Rendl, G. Rinaldi, and A. Wiegele. A branch and bound algorithm for max-cut based on combining semidefinite and polyhedral relaxations. *Integer programming and combinatorial optimization*, 4513:295–309, 2007. 10

[89] F. Rendl and R. Sotirov. Bounds for the quadratic assignment problem using bundle method. *Mathematical Programming, Series B*, 109:505–524, 2007. 10, 49, 50

[90] H. D. Sherali and W. P. Adams. 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, 1990. 76

[91] N.Z. Shor. A class of global minimum bounds of polynomial functions. *Cybernetics*, 23(6):731–734, 1987. 2, 14, 15

[92] L. Steinberg. The backboard wiring problem: A placement algorithm. *SIAM Review*, 3:37–50, 1961. 47

[93] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12, 1999. 65, 100, 125

[94] J Sturm and S. Zhang. On cones of nonnegative quadratic functions. Technical report, Chinese University of Hong Kong, 2001. 24

[95] M.J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001. 8

[96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996. 2, 8, 12

[97] J.C. Vera, L.F. Zuluaga, and J. Peña. Positive polynomials on equality constrained domains. *Working Paper*. 56

[98] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity. *SIAM Journal on Optimization*, 17(1):218–242, 2006. 3

[99] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. SparsePOP : a sparse semidefinite programming relaxation of polynomial optimization problems. *ACM Transactions on Mathematical Software*, 35(2):15, 2008. 3

[100] H. Wolkowicz, R. Saigal, and L. Vandenberghe. *Handbook of Semidefinite programming -Theory, Algorithms, and Applications*. Kluwer, 2000. 8, 14

[101] Y. Ye and S. Zhang. New results on quadratic minimization. *SIAM Journal on Optimization*, 14:245–267, 2001. 24, 25

[102] Q. Zhao, S.E. Karisch, F. Rendl, and H. Wolkowicz. Semidefinite programming relaxations for the quadratic assignment problem. *Journal of Combinatorial Optimization*, 2:71–109, 1998. 49, 50, 67, 68

[103] L.F. Zuluaga. *A conic programming approach to polynomial optimization problems: Theory and applications*. PhD thesis, The Tepper School of Business, Carnegie Mellon University, Pittsburgh, 2004. 87

[104] L.F. Zuluaga, J. C. Vera, and J. Peña. LMI approximations for cones of positive semidefinite forms. *SIAM Journal on Optimization*, 16(4), 2006. 2, 14, 17