

# Semidefinite Relaxation Based Branch-and-bound Method for Nonconvex Quadratic Programming

by

Sha Hu

Submitted to the School of Engineering  
in partial fulfillment of the requirements for the degree of  
Master of Science in Computation for Design and Optimization  
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2006

© Massachusetts Institute of Technology 2006. All rights reserved.

Author .....

School of Engineering

August 11, 2006

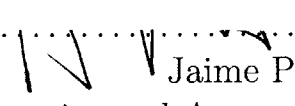
Certified by .....

 Pablo A. Parrilo

Associate Professor

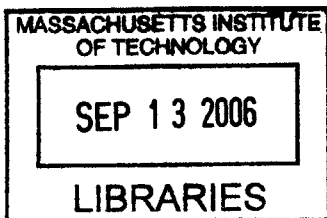
Thesis Supervisor

Accepted by .....

 Jaime Peraire

Professor of Aeronautics and Astronautics

Codirector, Computation for Design and Optimization Program



**BARKER**



# Semidefinite Relaxation Based Branch-and-bound Method for Nonconvex Quadratic Programming

by

Sha Hu

Submitted to the School of Engineering  
on August 11, 2006, in partial fulfillment of the  
requirements for the degree of  
Master of Science in Computation for Design and Optimization

## Abstract

In this thesis, we use a semidefinite relaxation based branch-and-bound method to solve nonconvex quadratic programming problems. Firstly, we show an interval branch-and-bound method to calculate the bounds for the minimum of bounded polynomials. Then we demonstrate four SDP relaxation methods to solve nonconvex Box constrained Quadratic Programming (BoxQP) problems and the comparison of the four methods. For some lower dimensional problems, SDP relaxation methods can achieve tight bounds for the BoxQP problem; whereas for higher dimensional cases (more than 20 dimensions), the bounds achieved by the four Semidefinite programming (SDP) relaxation methods are always loose. To achieve tight bounds for higher dimensional BoxQP problems, we combine the branch-and-bound method and SDP relaxation method to develop an SDP relaxation based branch-and-bound (SDPBB) method. We introduce a sensitivity analysis method for the branching process of SDPBB. This sensitivity analysis method can improve the convergence speed significantly. Compared to the interval branch-and-bound method and the global optimization software BARON, SDPBB can achieve better bounds and is also much more efficient. Additionally, we have developed a multisection algorithm for SDPBB and the multisection algorithm has been parallelized using Message Passing Interface (MPI). By parallelizing the program, we can significantly improve the speed of solving higher dimensional BoxQP problems.

Thesis Supervisor: Pablo A. Parrilo  
Title: Associate Professor



## Acknowledgments

This thesis could not have been completed without the support of numerous individuals. I would like to thank my thesis advisor, Pablo A. Parrilo, for his guidance and support. I gained a lot from every discussion we had. It is a pleasure to work with such an insightful supervisor.

I would like to thank the professors and staff of the Singapore-MIT-Alliance (SMA). Prof. Toh Kim Chuan has given me precious guidance and Prof. Khoo Boo Cheong has encouraged me continuously. Terrence has helped me a great deal on the computer cluster. Without the financial and hardware support of SMA, I could not have finished this thesis.

My sincere gratitude goes to the CDO faculties and the fellow students in my class and research group. It is especially enjoyable to work together with Jiali and Sandeep in LIDS. I would also like to thank all the new friends I have made in MIT, who have made my life in MIT so wonderful.

Finally, I would like to thank my family for their unconditional love and support.

THIS PAGE INTENTIONALLY LEFT BLANK

# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	Motivation and literature review . . . . .	13
1.2	Problem statement and solution strategy . . . . .	15
1.3	Structure of the thesis . . . . .	16
<b>2</b>	<b>Interval Branch-and-Bound Method</b>	<b>17</b>
2.1	Implementation . . . . .	17
2.2	Interval arithmetic . . . . .	18
2.3	Convergence of the method . . . . .	20
2.4	Sample problem . . . . .	21
<b>3</b>	<b>Semidefinite Relaxation Methods</b>	<b>25</b>
3.1	Diagonal SDP relaxation method . . . . .	25
3.2	Full-matrix SDP relaxation method . . . . .	28
3.3	Block-matrix SDP relaxation method . . . . .	30
3.4	Tridiagonal SDP relaxation method . . . . .	31
3.5	Comparison of the four relaxation methods . . . . .	32
<b>4</b>	<b>Semidefinite Relaxation based Branch-and-Bound Method</b>	<b>37</b>
4.1	Branch-and-bound scheme for semidefinite relaxation methods . . . . .	38
4.2	Scaling the QP problem over arbitrary box to the BoxQP problem . . . . .	40
4.3	Sensitivity analysis for the four semidefinite relaxations . . . . .	41
4.3.1	Sensitivity analysis for diagonal method . . . . .	42

4.3.2	Sensitivity analysis for full-matrix method . . . . .	43
4.3.3	Sensitivity analysis for block-matrix method . . . . .	44
4.3.4	Sensitivity analysis for tridiagonal method . . . . .	45
4.4	Result and comparison . . . . .	45
4.4.1	Comparison of SDPBB with various SDP relaxation and splitting methods . . . . .	45
4.4.2	Comparison of SDPBB and interval branch-and-bound method	48
4.4.3	Comparison of SDPBB and BARON . . . . .	48
<b>5</b>	<b>Parallelization of Semidefinite Relaxation based Branch-and-bound Method</b>	<b>51</b>
5.1	Multisection in SDPBB . . . . .	51
5.2	Parallelization scheme . . . . .	54
5.3	Implementation of the PSDPBB method . . . . .	54
5.3.1	Standardized SDP problems for CSDP solver . . . . .	56
5.4	Results and comparison . . . . .	59
<b>6</b>	<b>Conclusions and Future Work</b>	<b>63</b>
6.1	Conclusions . . . . .	63
6.2	Future work . . . . .	63
<b>A</b>	<b>SDPBB and PSDPBB User's Guide</b>	<b>65</b>
A.1	Introduction . . . . .	65
A.1.1	SDP relaxation methods and sensitivity analysis . . . . .	66
A.2	Installation of SDPBB and PSDPBB . . . . .	68
A.3	Usage of SDPBB . . . . .	69
A.4	Usage of PSDPBB . . . . .	70



# List of Figures

2-1	Flow chart of interval branch-and-bound method . . . . .	19
2-2	Convergence for the global minimum of the polynomial in Section 2.4	22
2-3	Plot for the converged result for the polynomial in Section 2.4 . . . . .	23
3-1	Comparison of average computation time for the four SDP relaxation methods . . . . .	33
3-2	Time vs. bound quality for four randomly generated 10-dimensional BoxQP instances . . . . .	35
3-3	Time vs. bound quality for four randomly generated 20-dimensional BoxQP instances . . . . .	36
4-1	Flow chart for branch-and-bound based semidefinite relaxation methods	39
4-2	Comparison of SDPBb for a 20-dimensional QP problem . . . . .	46
4-3	Comparison of SDPBb for a 30-dimensional QP problem . . . . .	47
4-4	Number of active problems for each iteration during convergence pro- cess. Top row: three 20-dimensional instances; Bottom row: three 30-dimensional instances. . . . .	49
5-1	Two multisection methods. Left: multibisection; Right: multisplitting	53
5-2	Parallelization scheme for PSDPBb . . . . .	55
5-3	Two-dimensional linked list for constraint matrices in both diagonal and full-matrix method . . . . .	56
5-4	Convergence comparison for using different number of processors on a 50-dimensional instance . . . . .	60

5-5	Comparison of diagonal and full-matrix methods for 10-dimensional instances . . . . .	61
5-6	Computation result for instances of dimensions 20 to 80 . . . . .	61
A-1	Input file for SDPBB and PSDPBB . . . . .	69
A-2	Sample SGE file: qpbbsge.sh . . . . .	71

# List of Tables

2.1	Notations for section 2.3 . . . . .	21
3.1	Computation time for the four relaxation methods . . . . .	34
3.2	Bound results for the 10-dimensional instances . . . . .	34
3.3	Bound results for the 20-dimensional instances . . . . .	34
4.1	Computation time for the 20-dimensional QP problem using SDPBB	46
4.2	Computation time for the 30-dimensional QP problem using SDPBB	47
4.3	Comparison of SDPBB and interval branch-and-bound method on four randomly generated instances of different sizes . . . . .	49
4.4	Comparison of SDPBB and BARON on four randomly generated in- stances of different sizes . . . . .	50
5.1	Convergence comparison for PSDPBB using different number of pro- cessors for solving the 50-dimensional instance . . . . .	59

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 1

## Introduction

### 1.1 Motivation and literature review

Quadratic optimization is one of the most important areas in nonlinear programming, both from the mathematical and application viewpoints. Numerous real world problems such as the problems in planning and scheduling, game theory, economies of scale, engineering design, and microeconomics are naturally expressed as Quadratic Programming (QP) problems. Moreover, QP problems encompass all the Linear Programming (LP) problems since QP problems with linear constraints can be viewed as a generalization of the LP problem with a quadratic objective function.

In general, QP problems can be classified into convex QP problems and nonconvex QP problems. For convex QP problems, there exists several algorithms that can solve the problems in polynomial time [12]. However, nonconvex problems are often NP-hard, which means that the problems cannot be solved in polynomial time. Nonconvex QP problems can be classified into three types: bilinear, concave quadratic and indefinite quadratic. The indefinite quadratic problems, in particular, have been arousing lots of researchers' interests for decades. Other than Semidefinite Programming (SDP) methods, most efforts are focused on firstly reducing the indefinite quadratic problem to either a bilinear or a concave minimization problem. There are only a few algorithms that directly solves the indefinite quadratic problem which includes the branch-and-bound method raised by Hansen et al. [9]. Gould and Toint [7] pre-

sented a survey on nonconvex QP problem solution methods using nonlinear programming techniques such as active-set or interior-point methods. Detailed introduction about quadratic optimization can be found in the survey presented by Floudas and Visweswaran [4]. Some global optimization softwares, such as BARON [18], can be used to solve nonconvex QP problems.

Since Goemans and Williamson [6] proposed a SDP relaxation for the quadratic maximization formulation of the max-cut problem, lots of work have been focused on solving the nonconvex QP problems using SDP relaxation methods. Goemans and Williamson [6] showed that SDP relaxation could yield a very good approximation algorithm. Fujie and Kojima [5] presented an SDP relaxation for a general nonconvex QP having a linear objective function and quadratic inequality constraints. Nesterov [15, 16], Ye [25, 26, 16], Nemirovski et al. [14] and Zhang [27] further extended Goemans and Williamson's model to other cases of nonconvex QP problems. Tseng [21] presented an approximation bound for the SDP relaxation of quadratically constrained quadratic optimization problems. Huang and Zhang [10] presented approximation algorithms for indefinite complex quadratic maximization problems.

The SDP techniques have been proven to be powerful both in theory and practice in the past decade. SDP problems can be solved by a number of solvers such as SeDuMi, SDPA, Yalmip, SDPT3, CSDP and SOSTOOLS. Two authoritative surveys about SDP techniques have been written by Todd [20] and Vandenberghe and Boyd [22].

In this thesis, we discuss the branch-and-bound method and SDP relaxations for nonconvex Quadratic Problems with Box constraints (BoxQP). BoxQP problems, as the simplest form of global nonconvex optimization, appear frequently in the solution of partial differential equations, discretized continuous time optimal control problems and linear least square problems. Vandebussche and Nemhauser [24, 23] presented a branch-and-cut algorithm based on first-order or second-order Karush-Kuhn-Tucker (KKT) conditions to solve BoxQP problems. Burer and Vandebussche [2] expanded Vandebussche and Nemhauser's approach and presented a branch-and-bound method using SDP relaxation to solve BoxQP problems. In this thesis, we present a different

SDP method for BoxQP problems.

## 1.2 Problem statement and solution strategy

The general form of QP can be shown as follows:

$$\begin{aligned} & \min_x x^T Q x + 2b^T x + c \\ \text{subject to } & Ax \leq d, \end{aligned} \tag{1.1}$$

where  $x \in \mathbb{R}^n$  is the variable and  $(Q, A, b, c, d) \in \mathbb{R}^{n \times n} \times \mathbb{R}^{m \times n} \times \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^m$  are the data. In particular, the problem is called concave QP problem when  $Q$  has all the eigenvalues to be negative; it is called indefinite QP problem when  $Q$  has both positive and negative eigenvalues; both concave QP problems and indefinite QP problems are nonconvex QP problems. The problem (1.1) is called BoxQP when  $A = [I; -I]$  and  $b = e$ . The BoxQP problem is shown as the following:

$$\begin{aligned} f_{\text{BoxQP}} &= \min_x x^T Q x + 2b^T x + c \\ \text{subject to } & -e \leq x \leq e. \end{aligned} \tag{1.2}$$

We denote the optimal value of (1.2) as  $f_{\text{BoxQP}}$ . The BoxQP problems we consider in this thesis are all nonconvex. It is obvious that all the QP problems in the following form can be scaled into BoxQP problems:

$$\begin{aligned} & \min_x x^T Q x + 2b^T x + c \\ \text{subject to } & x^L \leq x \leq x^U, \end{aligned} \tag{1.3}$$

where  $x^L$  and  $x^U$  are the lower bound and upper bound of  $x$  respectively.

We present several solution strategies for the above BoxQP problems. The first strategy is an interval branch-and-bound algorithm for minimization of polynomial over bounded regions. Another solution strategy is called the SDP relaxation method. In this thesis, we present four SDP relaxation methods for BoxQP problems. For some lower dimensional ( $\leq 20$ ) BoxQP problems, some of the SDP relaxation meth-

ods can achieve tight bounds on  $f_{\text{BoxQP}}$ . For larger dimensional BoxQP problems, a combination of the first two strategies, the SDP relaxation based Branch-and-Bound (SDPBB) method, is used. The branch-and-bound algorithm splits the feasible domain of  $x$  into sub-domains, and the SDP relaxation method is used to calculate the bounds for the global minimum over every sub-domain. This process is iterated until the branch-and-bound method converges, and tight bounds for the object value  $f_{\text{BoxQP}}$  can be achieved. To make the SDPBB method more efficient, we parallelize SDPBB. Multisection method is used in the parallel approach, where one subproblem is split into more than two subproblems during the branching process.

### 1.3 Structure of the thesis

The rest of the thesis is organized as follows. In Chapter 2, we explore the interval branch-and-bound method and its convergence. In Chapter 3, we present four SDP relaxations for BoxQP problems and compare their relative advantages and disadvantages. In Chapter 4, we combine the branch-and-bound method and the SDP relaxation techniques to solve higher dimensional BoxQP problems; and we present a sensitivity analysis to improve the convergence speed of the SDPBB method. In Chapter 5, we describe the parallel approach for the method in Chapter 4, and show the computation result for the parallel approach. In Chapter 6, we present conclusions and future directions.



# Chapter 2

## Interval Branch-and-Bound Method

In this chapter, we consider the minimization of a multivariate polynomial over bounded regions:

$$f = \min_{x \in I^n} \sum_{\alpha} p_{\alpha} x^{\alpha} \quad (2.1)$$

where  $x^{\alpha} = x_1^{\alpha_1} x_2^{\alpha_2} \dots x_n^{\alpha_n}$  and  $\sum_i \alpha_i \leq m$ ,  $m$  is the degree of the polynomial,  $n$  is the dimension of  $x$ ,  $I^n$  is an  $n$ -dimensional closed real interval. We denote the objective function as  $f$  in this chapter. All the intervals are denoted in capital letters, while other variables are denoted in lowercase letters.

### 2.1 Implementation

We use a branch-and-bound algorithm to find the objective value of problem (2.1). As shown in Figure 2-1, we have the following steps for the branch-and-bound algorithm.

**Step 1.** Initialize the list of active subproblems with the original given interval, and calculate the bounds for the minimum over this interval. Active subproblems are the subproblems that may contain the global optimizer.

**Step 2.** Select one subproblem to split. This subproblem is either the one with the

smallest lower bound (best-bound-first strategy) or the one with the largest interval size (largest-size-first strategy).

**Step 3.** Split the selected interval along the longest edge of the interval. Update the list of active subproblems with the two newly generated subproblems. Calculate the new bounds over the two newly generated sub-intervals.

**Step 4.** Prune the active list by deleting invalid intervals whose lower bound are larger than the minimum upper bound of all the subproblems in the active list.

We keep iterating Step 2 to Step 4 until the convergence criteria is satisfied. Two phases of branch strategies are used in this algorithm: 1) best-bound-first strategy and 2) largest-size-first strategy. The first stage of the program uses the best-bound-first branch strategy, which provides the best lower bound of the global minimum. The second stage of the algorithm uses the largest-size-first branch strategy, which provides the estimation of the interval that provides the best lower bound. The first stage of the algorithm stops when the subproblem with the best bound has an interval of very small size and the improvement of the lower bound between the last iteration and the current iteration is very small. The second stage of the algorithm stops when all the active subproblems have intervals of very small size. The acceptable small size for the algorithm to stop is called tolerance for interval branch-and-bound method. Interval analysis [8] is used to find the lower bound of the minimum of the polynomial over one interval; the value of the polynomial over the mid-point of the interval is used as the upper bound of the minimum. A simple description of the interval arithmetic is shown in section 2.2.

## 2.2 Interval arithmetic

In this section, we present a finite interval arithmetic that we used to calculate the lower bound of the polynomial. This arithmetic is presented by Hansen in [8].

Let  $x$  and  $y$  be two real numbers. If  $\bullet$  denotes the operations of addition, subtraction and multiplication, we have the operation of two interval numbers  $X$  and  $Y$

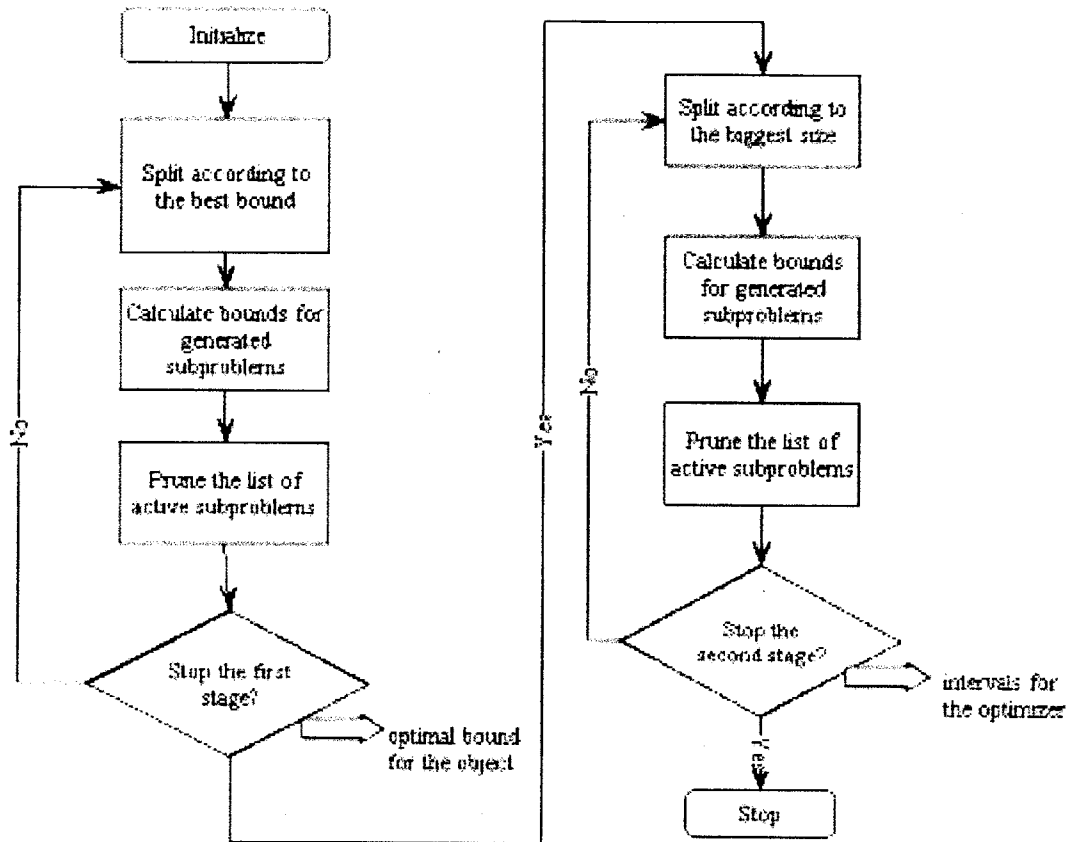


Figure 2-1: Flow chart of interval branch-and-bound method

as

$$X \bullet Y = \{x \bullet y | x \in X, y \in Y\}.$$

Let  $X = [a, b]$  and  $Y = [c, d]$ , for addition, we have

$$X + Y = [a + c, b + d].$$

For subtraction, we have

$$X - Y = [a - d, b - c].$$

For multiplication, we have

$$X \times Y = \begin{cases} [ac, bd] & \text{if } a \geq 0 \text{ and } c \geq 0 \\ [bc, bd] & \text{if } a \geq 0 \text{ and } c < 0 < d \\ [bc, ad] & \text{if } a \geq 0 \text{ and } d \leq 0 \\ [ad, bd] & \text{if } a < 0 < b \text{ and } c \geq 0 \\ [\min(bc, ad), \max(ac, bd)] & \text{if } a < 0 < b \text{ and } c < 0 < d \\ [bc, ac] & \text{if } a < 0 < b \text{ and } d \leq 0 \\ [ad, bc] & \text{if } b \leq 0 \text{ and } c \geq 0 \\ [ad, ac] & \text{if } b \leq 0 \text{ and } c < 0 < d \\ [bd, ac] & \text{if } b \leq 0 \text{ and } d \leq 0. \end{cases}$$

Another operation on one interval number  $X$  is power, which can be defined for nonnegative integer  $n$  as

$$X^n = \begin{cases} [1, 1] & \text{if } n = 0 \\ [a^n, b^n] & \text{if } a \geq 0 \text{ or if } n \text{ is odd} \\ [b^n, a^n] & \text{if } b \leq 0 \text{ or if } n \text{ is even} \\ [0, \max(a^n, b^n)] & \text{if } a \leq 0 \leq b \text{ and } n > 0 \text{ is even.} \end{cases}$$

## 2.3 Convergence of the method

In this section, we show that the interval branch-and-bound method converges to the optimal value of  $f$  at the first stage, and converges to the optimizer  $x$  at the second stage. The notations we use in this section are shown in Table 2.1.

For the first stage of our algorithm, the following assertion holds.

**Theorem 2.3.1** [17] *If  $w(Y_i) \rightarrow 0$  implies  $w(F(Y_i)) \rightarrow 0$  or  $w(Y) \rightarrow 0$  implies  $w(F(Y)) - w(f(Y)) \rightarrow 0$ , then the Moore-Skelboe algorithm converges to the global minimum:  $\min_{Y \in A} lbF(Y) \rightarrow f(x^*)$ , where  $x^*$  denotes one of the global minimizers.*

The Moore-Skelboe algorithm [13, 19, 17] is the same as what we have implemented in our first stage of interval branch-and-bound algorithm. Therefore, by the end of

Notation	Description
$\square$	The set of all closed one dimensional real intervals
$\square^n$	The set of all closed $n$ -dimensional real intervals
$F : \square^n \rightarrow \square$	Interval function of the object function $f$ , where for $\forall Y \in \square^n, \forall y \in Y$ has $f(y) \in F(Y)$
$A$	The union of the intervals for all the active subproblems
$lbY, ubY$	Lower bound and upper bound of the interval $Y$
$w(Y)$	Width of the interval, which is $\max_i(ubY_i - lbY_i)$

Table 2.1: Notations for section 2.3

the first stage, we can get an optimal lower bound of  $f$ . The minimizer is contained in  $A$  at the end of the first stage. In the worst case,  $A$  could be larger than half of the original interval  $I^n$  in size. Therefore, we need to reduce the size of  $A$  in the second stage of our algorithm.

In the second stage of our algorithm, we keep splitting the active subproblems with the largest interval size until the interval size for the active subproblems are all very small. In this way, we can reduce the range of the optimizer. If the union  $A$  we obtained at the end of the second stage is small, we can have a well-defined optimizer. If the union  $A$  is large, we know that there is a wide choice of parameters that yield near-optimal function values.

The second stage of the algorithm will not change the value of the lower bound obtained from the first stage, because the interval that generates the optimal lower bound is very small and will not be further split in the second stage. For other active intervals, in the second stage, they can only have tighter bound than that at the end of the first stage, which means the lower bounds over these intervals can only become larger in the second stage.

## 2.4 Sample problem

Polynomial  $p = 4x^2 - \frac{21}{10}x^4 + \frac{1}{3}x^6 + xy - 4y^2 + 4y^4$  harbors four local minimums in the box specified by the four vertices  $(-2, -1)$ ,  $(-2, 1)$ ,  $(2, -1)$  and  $(2, 1)$ . We set the tolerance for the first stage and second stage of the interval branch-and-bound algorithm as 0.01. The interval branch-and-bound method converges in 4670

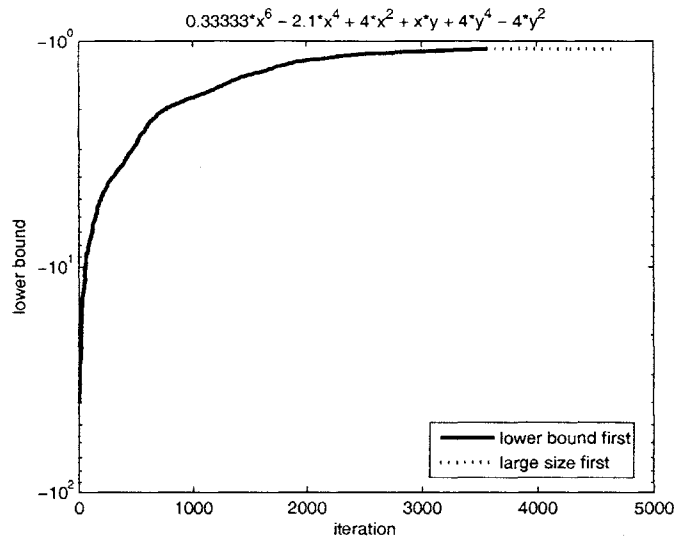


Figure 2-2: Convergence for the global minimum of the polynomial in Section 2.4

iterations as shown in Figure 2-2. The first stage finishes in 3567 iterations, and the second stage finishes in 1103 iterations. The run time for this problem in Matlab is 577.95 seconds. The lower bound for the problem is  $-1.0829$ , and the upper bound for the problem is  $-1.0306$ . The area with the active boxes when the method stops is shown in Figure 2-3 as the shaded part of the contour plot.

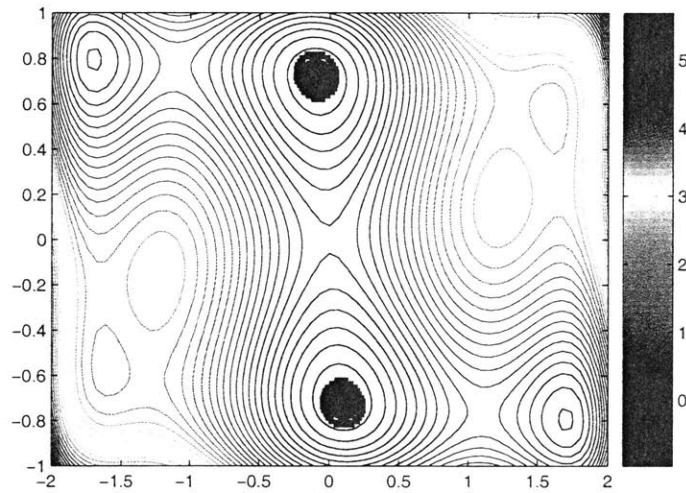


Figure 2-3: Plot for the converged result for the polynomial in Section 2.4

THIS PAGE INTENTIONALLY LEFT BLANK



# Chapter 3

## Semidefinite Relaxation Methods

In this chapter, we show four SDP relaxation methods to solve BoxQP problems. These four SDP relaxation methods are called diagonal method, full method, block-matrix method and tridiagonal method. We shall discuss the diagonal method and full-matrix method in detail. The block-matrix method and tridiagonal method are derived from the first two methods. The diagonal method is the cheapest computationally but it achieves the loosest bounds. The full-matrix method is the most expensive computationally but it achieves the tightest bounds.

### 3.1 Diagonal SDP relaxation method

First we show the diagonal SDP relaxation method. The BoxQP problem can be rewritten as

$$\begin{aligned} \min_x & \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \\ \text{subject to} & \quad x_i^2 \leq 1, \quad i = 1, 2, \dots, n. \end{aligned} \tag{3.1}$$

We can convert the problem (3.1) to the SDP form by introducing an  $(n+1) \times (n+1)$  symmetric matrix  $X$ , for which we have

$$x_i^2 \leq 1 \Leftrightarrow \begin{cases} X = \begin{bmatrix} x \\ 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}^T \\ X_{i,i} \leq 1, i = 1, 2, \dots, n \end{cases} \quad (3.2)$$

$$\Leftrightarrow \begin{cases} X_{n+1,n+1} = 1 \\ \text{rank}(X) = 1 \\ X \succeq 0 \\ X_{i,i} \leq 1, i = 1, 2, \dots, n. \end{cases} \quad (3.3)$$

By discarding the constraint  $\text{rank}(X) = 1$ , we can get an SDP relaxation as

$$\begin{aligned} & \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\ \text{subject to} & \quad X_{n+1,n+1} = 1 \\ & \quad X_{i,i} \leq 1, i = 1, 2, \dots, n \\ & \quad X \succeq 0. \end{aligned} \quad (3.4)$$

The dual of the problem (3.4) is:

$$\begin{aligned} & \max \gamma \\ \text{subject to} & \quad \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \succeq \begin{bmatrix} -\lambda_1 & & & \\ & \ddots & & \\ & & -\lambda_n & \\ & & & \sum_{i=1}^n \lambda_i \end{bmatrix} \\ & \quad \lambda_i \geq 0, i = 1, 2, \dots, n. \end{aligned} \quad (3.5)$$

By multiplying the first constraint of the problem (3.5)  $\begin{bmatrix} x \\ 1 \end{bmatrix}^T$  and  $\begin{bmatrix} x \\ 1 \end{bmatrix}$  on the

left and right respectively, we obtain the expression

$$x^T Ax + 2b^T x + c - \gamma \geq \sum_{i=1}^n \lambda_i (1 - x_i^2), \quad (3.6)$$

in which the right hand side is nonnegative when  $\lambda_i$  is nonnegative and  $x_i$  is bounded by  $-1$  and  $1$ . The optimal value of  $\gamma$  from (3.5) is a lower bound of  $f_{\text{BoxQP}}$  in (1.2).

An upper bound of  $f_{\text{BoxQP}}$  can be calculated from a primal feasible solution of (1.2). When the SDP relaxation provides a very close approximation of the original problem, we can have  $\text{rank}(X) \approx 1$ , which means

$$X = \begin{bmatrix} x_1 x_1 & x_1 x_2 & \cdots & x_1 x_n & x_1 \\ x_2 x_1 & x_2 x_2 & \cdots & x_2 x_n & x_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_n x_1 & x_n x_2 & & x_n x_n & x_n \\ x_1 & x_2 & \cdots & x_n & 1 \end{bmatrix}.$$

Therefore, if the SDP relaxation is good enough, the minimizer  $x^*$  extracted from the last column of  $X$  ( $x^* = [X_{1,n+1}, X_{2,n+1}, \dots, X_{n,n+1}]^T$ ) will give us a very close upper bound of  $f_{\text{BoxQP}}$ .

If  $\text{rank}(X)$  is not close to one,  $x^*$  is still feasible according to the diagonally dominant property of the positive semidefinite matrix  $X$ . We can always use the value of  $x^{*T} Ax^* + 2x^{*T} b + c$  as an upper bound for  $f_{\text{BoxQP}}$ .

## 3.2 Full-matrix SDP relaxation method

An alternative expression for BoxQP is

$$\begin{aligned}
 & \min_x \begin{bmatrix} x \\ 1 \end{bmatrix}^T \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \\
 \text{subject to} & \quad (1 + x_i)(1 + x_j) \geq 0, & \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n \\
 & \quad (1 + x_i)(1 - x_j) \geq 0, & \quad i, j = 1, 2, \dots, n \\
 & \quad (1 - x_i)(1 - x_j) \geq 0, & \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n.
 \end{aligned} \tag{3.7}$$

which is equivalent to

$$\begin{aligned}
 & \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\
 \text{subject to} & \quad X_{n+1, n+1} = 1 \\
 & \quad \text{rank}(X) = 1 \\
 & \quad 1 + X_{i, n+1} + X_{j, n+1} + X_{i, j} \geq 0, \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n \\
 & \quad 1 + X_{i, n+1} - X_{j, n+1} - X_{i, j} \geq 0, \quad i, j = 1, 2, \dots, n \\
 & \quad 1 - X_{i, n+1} - X_{j, n+1} + X_{i, j} \geq 0, \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n \\
 & \quad X \succeq 0.
 \end{aligned} \tag{3.8}$$

Relaxing the constraint  $\text{rank}(X) = 1$ , we can have a rank one SDP relaxation for equation (3.8). The dual of the rank one relaxation is shown as follows:

$$\begin{aligned}
 & \max \lambda \\
 \text{subject to} & \quad \begin{bmatrix} A & b \\ b^T & c - \lambda \end{bmatrix} \succeq \begin{bmatrix} I & e \\ -I & e \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \\
 & \quad \lambda_{i, j} \geq 0, \quad i = 1, 2, \dots, 2n \\
 & \quad \quad \quad j = i + 1, \dots, 2n
 \end{aligned} \tag{3.9}$$

where

$$\Lambda_{full} = \begin{bmatrix} 0 & \lambda_{1,2} & \cdots & \lambda_{1,2n} \\ \lambda_{1,2} & 0 & & \\ & \ddots & \ddots & \ddots \\ \vdots & & & 0 & \lambda_{2n-1,2n} \\ \lambda_{1,2n} & \cdots & \lambda_{2n-1,2n} & 0 \end{bmatrix}. \quad (3.10)$$

Here,  $I$  is the  $n \times n$  identity matrix and  $e$  is an all-one vector with dimension  $n$ .

By multiplying  $\begin{bmatrix} x \\ 1 \end{bmatrix}^T$  and  $\begin{bmatrix} x \\ 1 \end{bmatrix}$  on the left and right respectively, we can get the expression

$$x^T A x + 2b^T x + c - \gamma \geq \begin{bmatrix} 1+x \\ 1-x \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x \\ 1-x \end{bmatrix}. \quad (3.11)$$

The right hand side of equation (3.11) is nonnegative when  $-1 \leq x_i \leq 1$  and  $\lambda_{i,j} \geq 0, \forall i, j$ . The optimal  $\gamma$  from equation (3.9) is a lower bound of  $f_{\text{BoxQP}}$ . We can calculate an upper bound for  $f_{\text{BoxQP}}$  the same way as in the diagonal SDP relaxation method.

The full-matrix SDP relaxation method is a full version of the diagonal SDP relaxation method. The diagonal SDP relaxation method could be written in the same format as that of the full-matrix relaxation method by replacing the matrix  $\Lambda_{full}$  with the matrix  $\Lambda_{diag}$ , where

$$\Lambda_{diag} = \begin{bmatrix} & & & \lambda_{1,1} & & \\ & 0 & & & \ddots & \\ & & & & & \lambda_{n,n} \\ \lambda_{1,1} & & & & & \\ & \ddots & & & & \\ & & \lambda_{n,n} & & & \\ & & & 0 & & \end{bmatrix}.$$

The full-matrix relaxation method has more Lagrange multipliers  $(\lambda_{i,j})$  than the diagonal relaxation method. The full-matrix relaxation method has  $n \times (2n - 1)$  entries in the matrix  $\Lambda_{full}$  while the diagonal relaxation method has only  $n$  entries in the matrix  $\Lambda_{diag}$ . The full-matrix relaxation method takes longer time to solve the SDP problem, and it achieves better approximation to BoxQP at the same time. Detailed comparisons and examples are shown in Section 3.5.

### 3.3 Block-matrix SDP relaxation method

A third relaxation for BoxQP is derived from the first two relaxation methods. The rank one relaxation is shown as follows:

$$\begin{aligned}
 & \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\
 \text{subject to} & \quad X_{n+1,n+1} = 1 \\
 & 1 - X_{i,n+1} + X_{j,n+1} - X_{i,j} \geq 0, \quad i = 1, 2, \dots, n, j = 1, 2, \dots, n \\
 & X \succeq 0.
 \end{aligned} \tag{3.12}$$

The dual of the problem (3.12) is shown as follows:

$$\begin{aligned}
& \max \lambda \\
& \text{subject to} \\
& \begin{bmatrix} A & b \\ b^T & c - \lambda \end{bmatrix} \succeq \begin{bmatrix} I & e \\ -I & e \end{bmatrix}^T \Lambda_{block} \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \\
& \lambda_{i,j} \geq 0 \quad i, j = 1, 2, \dots, n, \\
& \text{where} \\
& \Lambda_{block} = \begin{bmatrix} & & & \lambda_{1,1} & \cdots & \lambda_{1,n} \\ & & & \vdots & \ddots & \vdots \\ & & 0 & & & \\ & & & \lambda_{n,1} & \cdots & \lambda_{n,n} \\ \lambda_{1,1} & \cdots & \lambda_{1,n} & & & \\ \vdots & \ddots & \vdots & & 0 & \\ \lambda_{n,1} & \cdots & \lambda_{n,n} & & & \end{bmatrix}. \tag{3.13}
\end{aligned}$$

The block-matrix relaxation method has  $n \times n$  entries in its  $\Lambda_{block}$  matrix. The number of entries in matrix  $\Lambda_{block}$  is between that of the  $\Lambda_{diag}$  in the diagonal relaxation method and the  $\Lambda_{full}$  in the full-matrix relaxation method.

### 3.4 Tridiagonal SDP relaxation method

In the fourth SDP relaxation, we reduce the number of entries in matrix  $\Lambda_{block}$  to make it linear in the dimension of the QP problem. The rank one relaxation is shown as follows:

$$\begin{aligned}
& \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\
& \text{subject to} \\
& X_{n+1,n+1} = 1 \\
& 1 - X_{i,n+1} + X_{j,n+1} - X_{i,j} \geq 0, \quad i = 1, 2, \dots, n, j = i - 1, i, i + 1 \\
& X \succeq 0. \\
& \tag{3.14}
\end{aligned}$$

The dual of the problem (3.14) is shown as follows:

$$\begin{aligned}
& \max \lambda \\
& \text{subject to} \\
& \begin{bmatrix} A & b \\ b^T & c - \lambda \end{bmatrix} \succeq \begin{bmatrix} I & e \\ -I & e \end{bmatrix}^T \Lambda_{tridiag} \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \\
& \lambda_{i,j} \geq 0 \quad i = 2, \dots, n-1, j = i-1, i, i+1 \\
& \quad \quad \quad i = 1, j = 1, 2 \\
& \quad \quad \quad i = n, j = n-1, n,
\end{aligned} \tag{3.15}$$

where

$$\Lambda_{tridiag} = \left[ \begin{array}{ccc|ccc}
& & & \lambda_{1,1} & \lambda_{1,2} & & & & \\
& & & \lambda_{2,1} & \ddots & \ddots & & & \\
& & 0 & & \ddots & & & & \lambda_{n-1,n} \\
& & & & & & & \lambda_{n,n-1} & \lambda_{n,n} \\
\hline
\lambda_{1,1} & \lambda_{1,2} & & & & & & & \\
\lambda_{2,1} & \ddots & \ddots & & & & & & \\
& \ddots & & \lambda_{n-1,n} & & & & & \\
& & \lambda_{n,n-1} & \lambda_{n,n} & & & & & \\
& & & & & & & 0 & 
\end{array} \right].$$

The tridiagonal relaxation method has  $3n - 2$  entries in its  $\Lambda_{tridiag}$  matrix. The number of entries in the matrix  $\Lambda_{tridiag}$  is between the  $\Lambda_{diag}$  in the diagonal relaxation method and  $\Lambda_{block}$  in the block-matrix relaxation method.

### 3.5 Comparison of the four relaxation methods

Figure 3-1 and Table 3.1 show the average computation time for the four relaxation methods. The computation time is averaged from 20 randomly generated BoxQP problems of dimensions 2, 5, 10, 15 and 20. From Figure 3-1 and Table 3.1, we can observe that the computation time for the diagonal method and the tridiagonal method



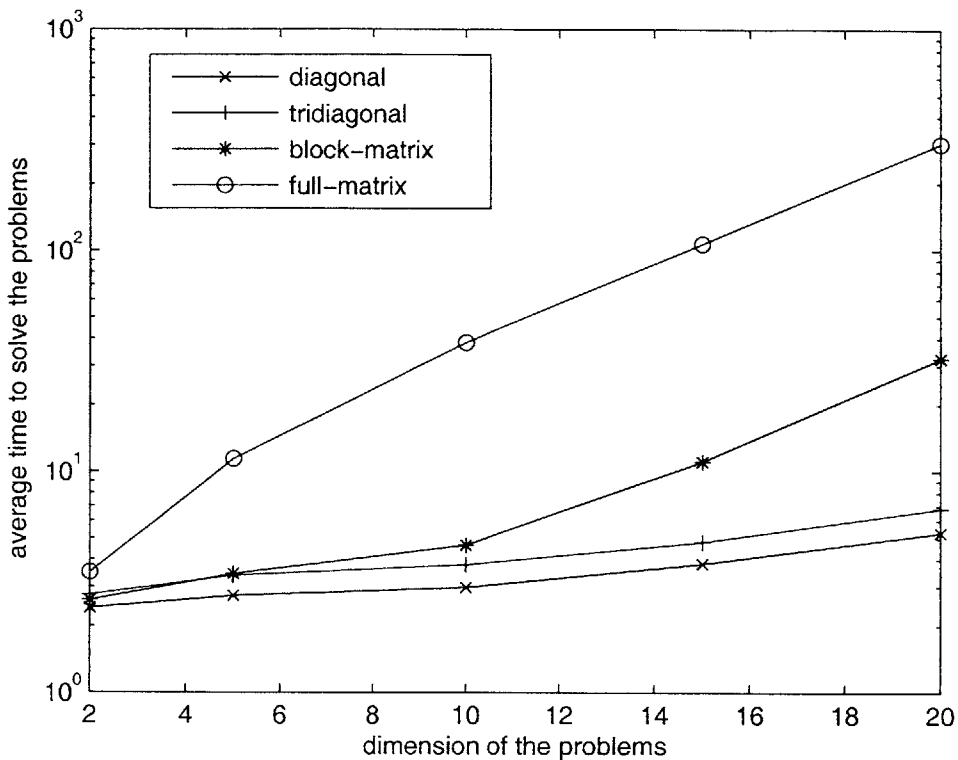


Figure 3-1: Comparison of average computation time for the four SDP relaxation methods

are short and increase relatively slowly as the dimension of the QP problem increases. The computation time for the block-matrix method and full-matrix method are longer and increase relatively fast as the dimension of the QP problem increases. The full-matrix method takes the longest computation time and it increases fastest as the dimension of the QP problem increases. Table 3.1 also shows that the computation time increases as the number of entries in the matrix  $A$  increases, and the rate of increase in computation time is higher than the rate of increase in the number of entries.

Figure 3-2 and Figure 3-3 show the relationship between the computation time and the quality of the lower bound achieved using the four relaxation methods. Each figure shows the results of the four randomly generated BoxQP problems. We observe that the tridiagonal method achieves better bounds than the diagonal method with

	dimension	diagonal method	tridiagonal method	block-matrix method	full-matrix method
time (sec.)	2	2.42	2.77	2.62	3.50
	5	2.74	3.38	3.44	11.39
	10	2.99	3.78	4.64	38.28
	15	3.83	4.78	11.05	106.80
	20	5.30	6.78	32.38	302.97
# of entries in $\Lambda$		$n$	$3n - 2$	$n^2$	$n \times (2n - 1)$

Table 3.1: Computation time for the four relaxation methods

		diagonal	tridiagonal	block-matrix	full-matrix
instance 1	lower bound	-315.9689	-311.9894	-308.582	-308.582
	upper bound	-242.1424	-254.0224	-308.582	-308.582
instance 2	lower bound	-273.0769	-264.9856	-252.1098	-249.15
	upper bound	-116.2217	-117.1016	-73.5387	-87.7108
instance 3	lower bound	-332.8964	-331.6831	-327.3333	-326.4389
	upper bound	-276.3527	-292.4958	-286.4482	-326.4387
instance 4	lower bound	-284.6661	-277.9293	-268.9722	-262.1849
	upper bound	-175.3825	-212.4848	-211.5864	-221.4125

Table 3.2: Bound results for the 10-dimensional instances

approximately the same computation time. The full-matrix method takes much longer time than the block-matrix method while the improvement of the quality of the bound may not be very obvious. Therefore, for a single BoxQP problem, the tridiagonal method is better than the diagonal method in terms of relative computation time and quality of the bound, and the block-matrix method is better than the full-matrix methods in the same aspects. Table 3.2 and Table 3.3 show the bounds for the

		diagonal	tridiagonal	block-matrix	full-matrix
instance 1	lower bound	-941.9511	-937.7298	-912.1942	-908.7084
	upper bound	-679.8002	-665.3004	-580.8613	-582.4111
instance 2	lower bound	-799.8424	-786.4626	-771.7601	-763.6481
	upper bound	-493.0752	-463.6194	-403.9468	-264.3009
instance 3	lower bound	-818.9348	-808.105	-795.2391	-784.9429
	upper bound	-503.2527	-580.6928	-426.708	-244.5714
instance 4	lower bound	-704.393	-697.8788	-683.015	-678.4359
	upper bound	-487.8156	-455.3614	-394.4857	-343.2326

Table 3.3: Bound results for the 20-dimensional instances

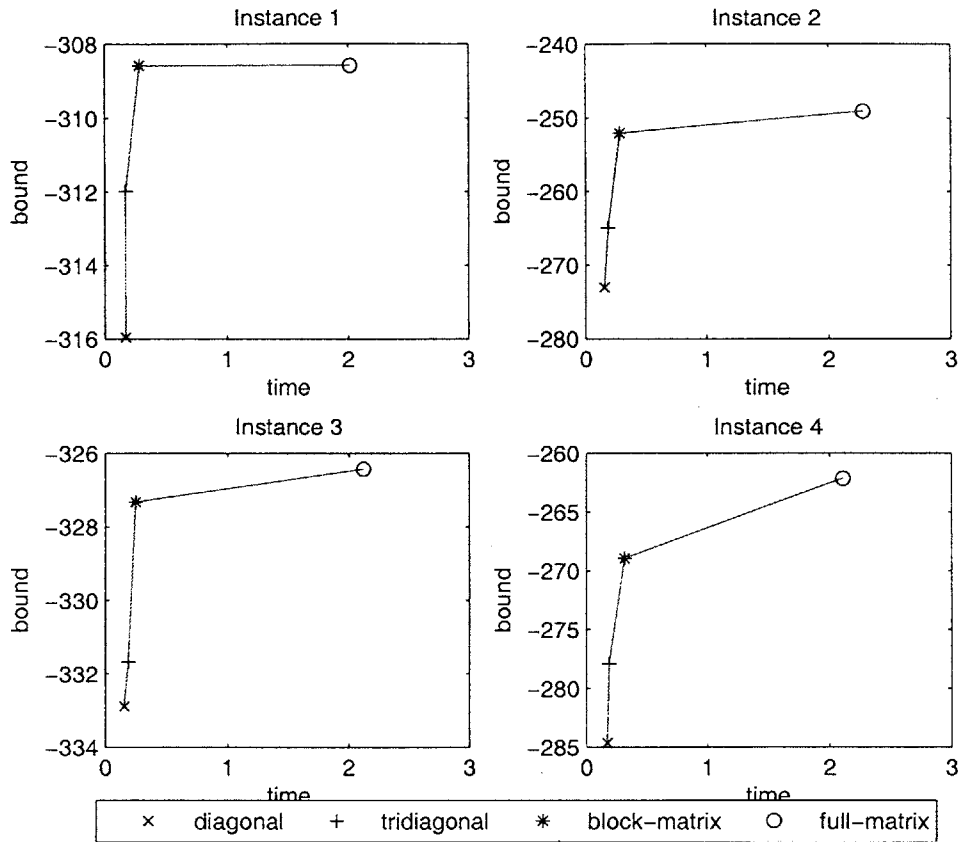


Figure 3-2: Time vs. bound quality for four randomly generated 10-dimensional BoxQP instances

problems shown in Figure 3-2 and 3-3 respectively. We can see that the block-matrix method and full-matrix method obtain tight bounds for the first instance of the 10-dimensional BoxQP problems.

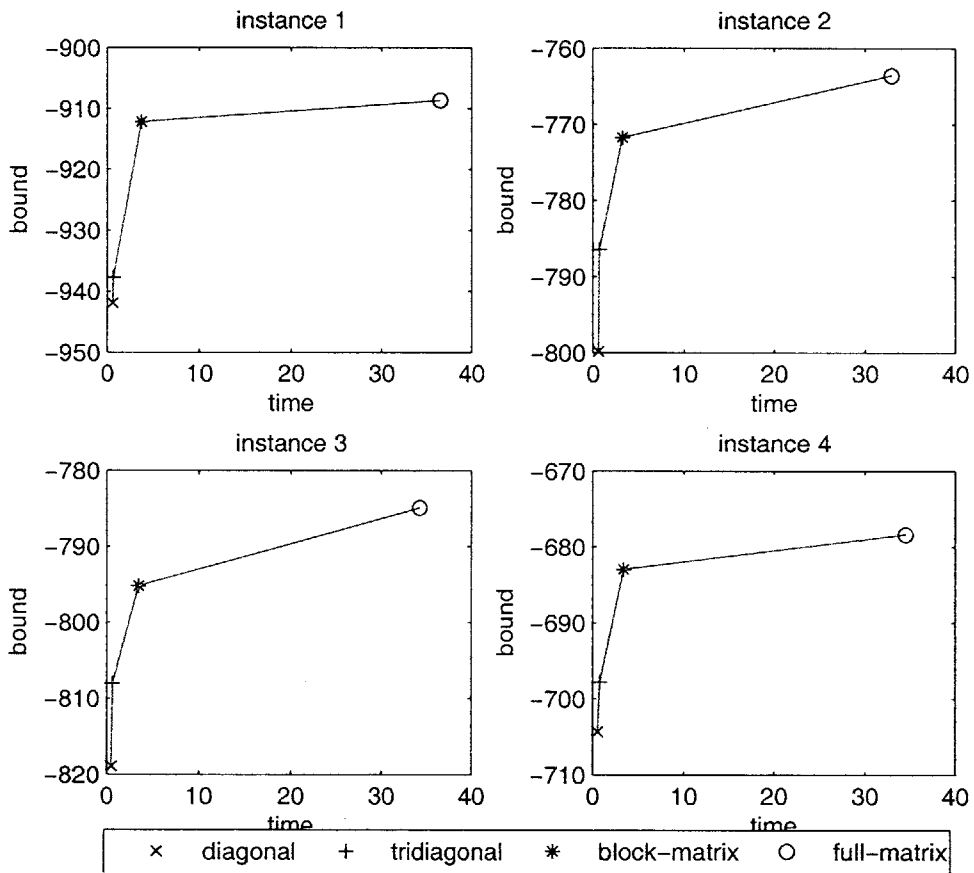


Figure 3-3: Time vs. bound quality for four randomly generated 20-dimensional BoxQP instances

## Chapter 4

# Semidefinite Relaxation based Branch-and-Bound Method

In the previous chapter, we provide four SDP relaxation methods to achieve the lower bound and upper bound for the minimum of nonconvex BoxQP problems. In this chapter, we use the combination of these SDP relaxation methods and branch-and-bound method to achieve the exact value of the minimum for large dimensional BoxQP problems. This Semidefinite Relaxation based Branch-and-Bound method is referred to as SDPBB. We present a sensitivity analysis method that can improve the convergence of SDPBB significantly. We show the reason why the branch-and-bound works for SDP relaxation methods as follows. Consider the QP problem over an arbitrary box:

$$\begin{aligned} \min \quad & x^T Ax + 2b^T x + c \\ & x_i^L \leq x_i \leq x_i^U, \quad i = 1, 2, \dots, n, \end{aligned} \tag{4.1}$$

we have the expression for diagonal method as

$$x^T Ax + 2b^T x + c - \gamma \geq \sum_{i=1}^n \lambda_i (x_i^U - x_i)(x_i - x_i^L), \tag{4.2}$$

and the expression for full-matrix method as

$$x^T Ax + 2b^T x + c - \gamma \geq \begin{bmatrix} -x^L + x \\ x^U - x \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} -x^L + x \\ x^U - x \end{bmatrix}. \quad (4.3)$$

We can observe from expressions (4.2) and (4.3) that the right hand side of these two equations may become close to zero when the bounds of variables,  $x^L$  and  $x^U$ , are close enough. Therefore, the bounds become tighter when we split a big box to several small boxes and calculate the bounds over small boxes, and this is why the SDPBB method works. By carefully consider the  $\lambda$ s and  $x$ , we can choose one dimension which, when split, can provides the most improvement of the bounds. This method of choosing the best dimension to split is called the sensitivity analysis method.

## 4.1 Branch-and-bound scheme for semidefinite relaxation methods

The branch-and-bound scheme of SDPBB is similar to that of the interval branch-and-bound method. Figure 4-1 shows the flowchart for SDPBB. The main difference is that the SDPBB method only uses best-bound-first strategy during branching. The following shows the strategies used in SDPBB:

- **Branching criteria:** we choose the subproblem to split using the best-bound-first strategy, which means that the subproblem with the smallest lower bound will be chosen to be split in each iteration. After choosing the subproblem, we choose one dimension to split according to the interval size of the dimension or sensitivity of the dimension. The sensitivity analysis method we use is shown in the Section 4.3.
- **Pruning criteria:** we search for the smallest upper bound in the list of active subproblems. We prune the subproblem whose lower bound is bigger than the smallest upper bound.

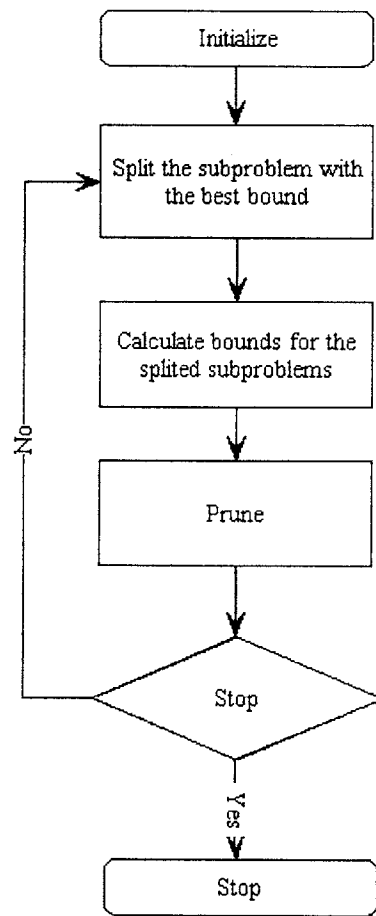


Figure 4-1: Flow chart for branch-and-bound based semidefinite relaxation methods

- Stopping criteria: the algorithm stops once the subproblem with the smallest lower bound achieves tight bounds, which means that the upper bound and lower bound for the problem are close enough. We call the biggest acceptable difference between tight bounds as tolerance in SDPBB.

Since the SDP relaxation method can provide tight bounds when the box size for one subproblem is small enough, and the minimizer is achieved once the tight bound is achieved, we only need the best-bound-first strategy in SDPBB. Since several active subproblems may achieve tight bounds during the branch-and-bound process, we need to handle the subproblems with tight bounds carefully. It is unsafe to stop once we get a subproblem with tight bound, because the algorithm may stop at a local minimum in this way. It is also unsafe to delete the subproblem with tight but not the lowest bound directly from the list of active problems, because this subproblem can still be the one with the global minimum: this tight bound may not be the smallest at that time because the lowest bound may happen to be loose at that time. Therefore, for every calculated subproblem, we keep a sign of whether it achieves tight bounds or not. When we search over the list of active subproblems for the one to be split, we only consider the active subproblems whose bounds are not tight. It is only safe for SDPBB algorithm to stop when the subproblem with the smallest lower bound is tight.

## 4.2 Scaling the QP problem over arbitrary box to the BoxQP problem

In the SDPBB method, the subproblems generated in each iteration are no longer standard BoxQP problems. To solve these subproblems, we need to scale the QP problem over an arbitrary box to the standard BoxQP problem.

We denote the box bounded by  $x^L, x^L$  in the problem (4.1) as  $\hat{B}$ . For any  $x^* \in \hat{B}$ , when  $\hat{B}$  is scaled to a standard box bounded by 1's and  $-1$ 's, its corresponding



coordinate  $y^*$  is

$$y_i^* = \left(x_i^* - \frac{x_i^L + x_i^U}{2}\right) \times \frac{2}{x_i^U - x_i^L}, \quad i = 1, 2, \dots, n. \quad (4.4)$$

Therefore, we have

$$x_i^* = \frac{x_i^U - x_i^L}{2} y_i^* + \frac{x_i^L + x_i^U}{2}, \quad i = 1, 2, \dots, n. \quad (4.5)$$

Denote  $k_i = \frac{x_i^U - x_i^L}{2}$ ,  $t_i = \frac{x_i^L + x_i^U}{2}$ ,  $K = \text{diag}(k_i)$  and  $t = [t_1, t_2, \dots, t_n]^T$ . We have the objective function of the problem (4.1) as

$$\begin{aligned} & x^T A x + 2x^T b + c \\ &= \begin{bmatrix} k_1 y_1 + t_1 \\ k_2 y_2 + t_2 \\ \vdots \\ k_n y_n + t_n \end{bmatrix}^T A \begin{bmatrix} k_1 y_1 + t_1 \\ k_2 y_2 + t_2 \\ \vdots \\ k_n y_n + t_n \end{bmatrix} + 2b^T \begin{bmatrix} k_1 y_1 + t_1 \\ k_2 y_2 + t_2 \\ \vdots \\ k_n y_n + t_n \end{bmatrix} + c \\ &= y^T K A K y + 2y^T K b + t^T A t + 2t^T b + c, \end{aligned} \quad (4.6)$$

Therefore, we have the corresponding BoxQP problem for (4.1) as

$$\begin{aligned} \min \quad & y^T K A K y + 2y^T K b + t^T A t + 2t^T b + c \\ & -1 \leq y_i \leq 1, \quad i = 1, 2, \dots, n. \end{aligned} \quad (4.7)$$

### 4.3 Sensitivity analysis for the four semidefinite relaxations

Adopting sensitivity analysis while branching can speed up the convergence of SDPBB. For multivariate BoxQP problems, the domain of  $x$  is a multi-dimensional unit box. To split the box into two, we need to choose one dimension to split. By carefully choosing the most sensitive dimension to split, we can achieve more improvement on the lower bound within one step of the branch-and-bound method.

In this section, we call the BoxQP problem to be solved “the original problem”.

Once we split the unit box in dimension  $i$ , we will have two subproblems whose ranges of  $x_i$  are changed to the intervals  $[-1, 0]$  and  $[0, 1]$  respectively. We call these two subproblems  $S(i1)$  and  $S(i2)$ . In sensitivity analysis, we examine the improvement of the lower bound achieved by every possible subproblem ( $i = 1, 2, \dots, n$ ) and choose the dimension providing the most improvement. The sensitivity analysis methods for the four SDP relaxations is shown in the subsections below.

### 4.3.1 Sensitivity analysis for diagonal method

From the expression derived from the diagonal method:  $x^T Ax + 2b^T x + c - \gamma \geq \sum_{i=1}^n \lambda_i(1 - x_i^2)$ , we can see that the lower bound  $\gamma$  is tight when the right hand side of this equation is zero, and the improvement of the lower bound  $\gamma$  is equal to the reduction of the right hand side value of this equation. In subproblem  $S(i1)$ , since the range of  $x_i$  has been changed from  $[-1, 1]$  to  $[-1, 0]$ , if  $x_i \in [-1, 0]$ , we have the improvement of the lower bound from the original problem to  $S(i1)$  as

$$\begin{aligned} & \lambda_i(1 - x_i^2) - \lambda_i(0 - x_i)(1 + x_i) \\ & = \lambda_i(1 + x_i) \end{aligned} \tag{4.8}$$

Similarly, the improvement of the lower bound for  $S(i2)$  when  $x_i \in [0, 1]$  is

$$\begin{aligned} & \lambda_i(1 - x_i^2) - \lambda_i(1 - x_i)(0 + x_i) \\ & = \lambda_i(1 - x_i) \end{aligned} \tag{4.9}$$

Since  $x_i$  should be either in  $[-1, 0]$  or in  $[0, 1]$ , only one of the results from equations (4.8) and (4.9) makes sense. It is obvious that the subproblem with the smaller improvement is the one that contains  $x_i$ . Therefore, we have the formulation for the sensitivity analysis as

$$\begin{aligned} & \max_i \min \{ \text{improvement of } S(i1), \text{improvement of } S(i2) \} \\ & = \max_i \min \{ \lambda_i(1 + x_i), \lambda_i(1 - x_i) \}. \end{aligned} \tag{4.10}$$

We can get the most improvement of the lower bound by splitting in the optimal

dimension  $i$ .

### 4.3.2 Sensitivity analysis for full-matrix method

For the full-matrix method, we have the expression:

$$x^T Ax + 2b^T x + c - \gamma \geq \begin{bmatrix} 1+x \\ 1-x \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x \\ 1-x \end{bmatrix}. \quad (4.11)$$

Therefore, for  $S(i1)$ , we have the improvement of the lower bound when  $x \in [-1, 0]$

as

$$\begin{aligned} & \text{improvement of } S(i1) \\ & \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix} - \begin{bmatrix} 1+x_1 \\ \vdots \\ 0+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x_1 \\ \vdots \\ 0+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix} \\ & = 2[\lambda_{i,1}, \lambda_{i,2}, \dots, \lambda_{i,n}] \begin{bmatrix} 1+x \\ 1-x \end{bmatrix} \end{aligned} \quad (4.12)$$

Similarly, we have the improvement of the lower bound for  $S(i2)$  when  $x \in [0, 1]$  as

$$\begin{aligned}
& \text{improvement of } S(i2) \\
& = \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 1-x_i \\ \vdots \\ 1-x_n \end{bmatrix} - \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 0-x_i \\ \vdots \\ 1-x_n \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} 1+x_1 \\ \vdots \\ 1+x_i \\ \vdots \\ 1+x_n \\ 1-x_1 \\ \vdots \\ 0-x_i \\ \vdots \\ 1-x_n \end{bmatrix} \\
& = 2[\lambda_{i+n,1}, \lambda_{i+n,2}, \dots, \lambda_{i+n,n}] \begin{bmatrix} 1+x \\ 1-x \end{bmatrix}
\end{aligned} \tag{4.13}$$

Therefore, we have the formulation to calculate the most sensitive dimension  $i$  as

$$\begin{aligned}
& \max_i \min \{ \text{improvement of } S(i1), \text{improvement of } S(i2) \} \\
& = \max_i \min \left\{ \bar{\lambda}_i \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}, \bar{\lambda}_{i+n} \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \right\},
\end{aligned} \tag{4.14}$$

where  $\bar{\lambda}_i$  is the  $i$ th row of the matrix  $\Lambda_{full}$ .

### 4.3.3 Sensitivity analysis for block-matrix method

We can do the sensitivity analysis for block-matrix method in the same manner as that for the full-matrix method. The formulation to calculate the most sensitive

dimension  $i$  is

$$\begin{aligned} & \max_i \min \{ \text{improvement of } S(i1), \text{improvement of } S(i2) \} \\ & = \max_i \min \left\{ \bar{\lambda}_i [-I \ e] \begin{bmatrix} x \\ 1 \end{bmatrix}, \bar{\lambda}_i [I \ e] \begin{bmatrix} x \\ 1 \end{bmatrix} \right\} \end{aligned} \quad (4.15)$$

where  $\bar{\lambda}_i$  is the  $i$ th row of the matrix  $\Lambda_{block}$ .

### 4.3.4 Sensitivity analysis for tridiagonal method

Similar to the full-matrix method and block-matrix method, we can have the sensitivity analysis formulation for the tridiagonal method as

$$\begin{aligned} & \max_i \min \{ \text{improvement of } S(i1), \text{improvement of } S(i2) \} \\ & = \max_i \min \left\{ [\lambda_{i,i-1} \ \lambda_{i,i} \ \lambda_{i,i+1}] \begin{bmatrix} 1 - x_{i-1} \\ 1 - x_i \\ 1 - x_{i,i+1} \end{bmatrix}, [\lambda_{i,i-1} \ \lambda_{i,i} \ \lambda_{i,i+1}] \begin{bmatrix} 1 - x_{i-1} \\ 1 - x_i \\ 1 - x_{i,i+1} \end{bmatrix} \right\}. \end{aligned} \quad (4.16)$$

## 4.4 Result and comparison

### 4.4.1 Comparison of SDPBB with various SDP relaxation and splitting methods

The performance of SDPBB using different SDP relaxations and different splitting methods is compared in this section. The two instances tested are from the box-constrained QPs introduced by Vandebussche and Nemhauser [23]. The comparison of results for a 20-dimensional instance is shown in Figure 4-2 and Table 4.1. The comparison of results for a 30-dimensional instance is shown in Figure 4-3 and Table 4.2. The four SDP relaxation methods are tested with and without sensitivity analysis. The stopping criteria for the methods are set as the absolute error for the objective value is less than 0.1. The programs are written in Matlab. Yalmip and

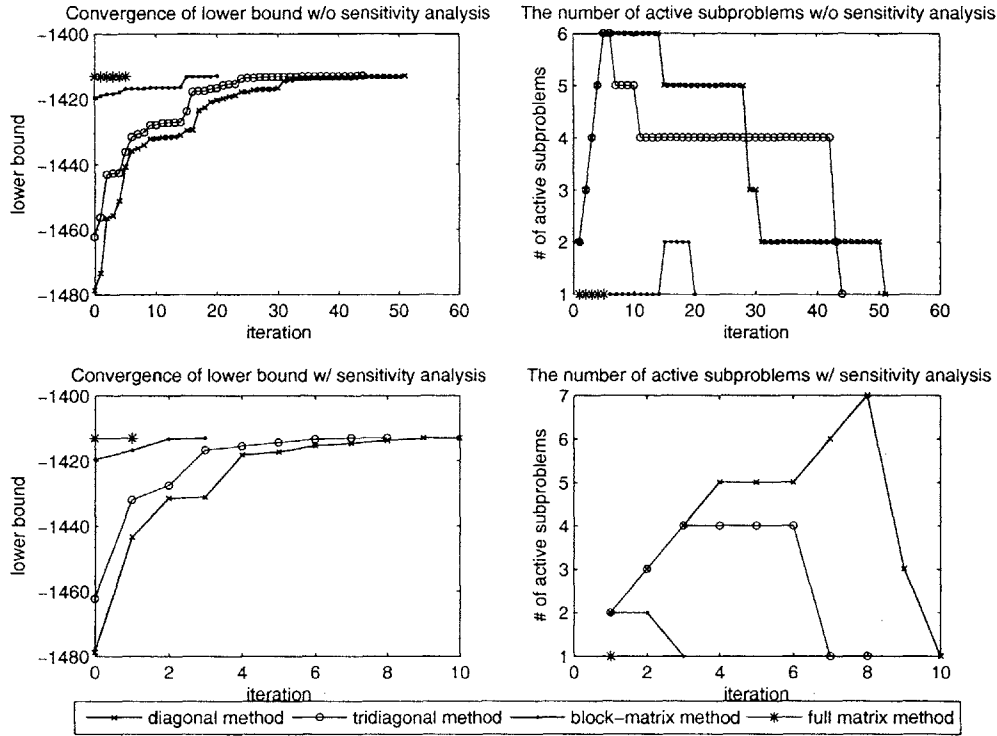


Figure 4-2: Comparison of SDPBB for a 20-dimensional QP problem

SeDuMi are used to solve the SDP relaxations.

	w/o sensitivity analysis		w/sensitivity analysis	
	total time	time for each iteration	total time	time for each iteration
diagonal method	52.12	1.02	10.81	1.08
tridiagonal method	68.34	1.55	13.81	1.72
block-matrix method	90.54	4.52	17.28	5.76
full-matrix method	174.96	34.99	47.34	47.34

Table 4.1: Computation time for the 20-dimensional QP problem using SDPBB

From Figure 4-2 and Figure 4-3, we can observe that convergence speed for the methods with sensitivity analysis is much faster than those without sensitivity analysis. Table 4.1 and Table 4.2 also show that the computation cost for sensitivity analysis is relatively small compared to the cost for calculating bounds. Therefore, splitting using sensitivity analysis is very effective for SDPBB.

Additionally, the SDP relaxation method with better bound quality always makes

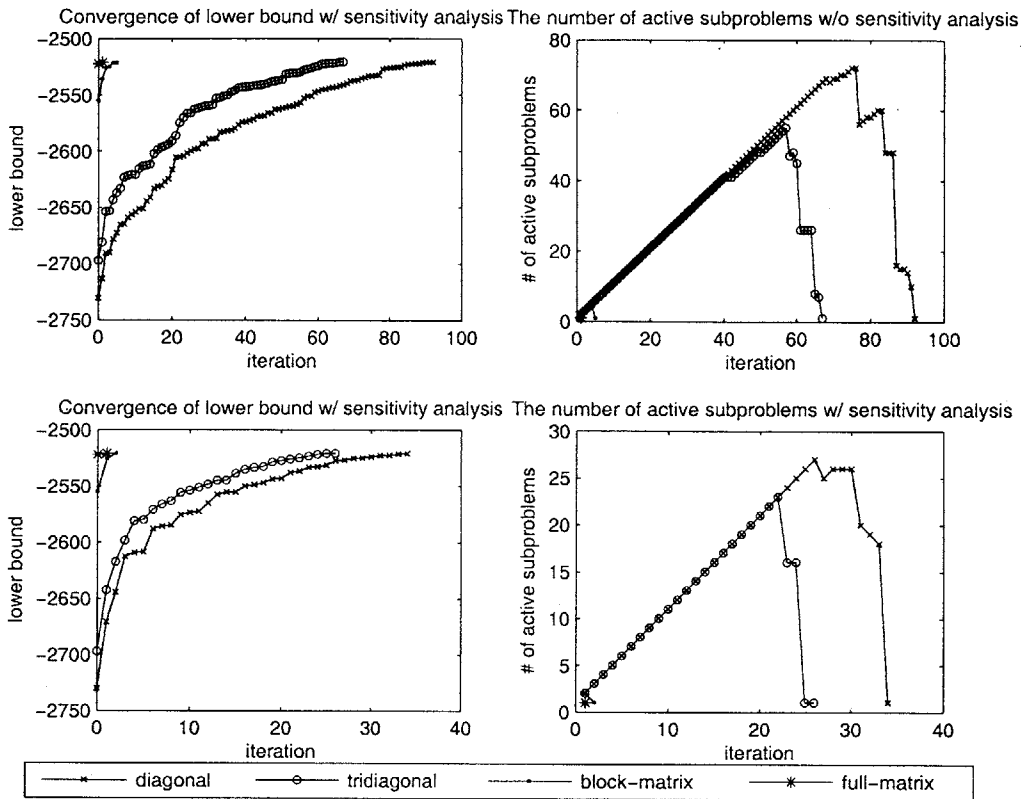


Figure 4-3: Comparison of SDPBB for a 30-dimensional QP problem

	w/o sensitivity analysis		w/sensitivity analysis	
	total time	time for each iteration	total time	time for each iteration
diagonal method	78.31	0.85	29.45	0.86
tridiagonal method	83.96	1.25	34.45	1.32
block-matrix method	182.46	36.49	84.01	42.00
full-matrix method	792.68	792.68	807.29	807.29

Table 4.2: Computation time for the 30-dimensional QP problem using SDPBB

the SDPBB method converge faster. However, the SDPBB method using the SDP relaxation method with better bound quality always takes longer time to finish. From Table 4.1 and Table 4.2, we can observe that the larger the BoxQP problem is, the higher the extra cost of achieving bounds of high quality. Therefore, it is more economic to calculate a large number of bounds of lower quality than to calculate very few bounds of higher quality for SDPBB method.

From the above comparison, we can conclude that SDPBB using diagonal SDP relaxation and sensitivity analysis is most efficient. Figure 4-4 shows the number of active subproblems for each iteration when SDPBB with diagonal method and sensitivity analysis is used. The 20-dimensional and 30-dimensional dense BoxQP instances from Vandembussche and Nemhauser [23] are tested. We can observe that for most of the instances, the number of active subproblems keeps increasing for the first half of the iterations.

#### **4.4.2 Comparison of SDPBB and interval branch-and-bound method**

We use interval branch-and-bound method and SDPBB to test some lower dimensional BoxQP instances. The results are shown in Table 4.3. We can see that the SDPBB method achieves much better bounds in a much shorter time, especially when the dimension of the BoxQP problem is larger.

#### **4.4.3 Comparison of SDPBB and BARON**

We use BARON [18] and SDPBB to test some 10- to 30-dimensional instances. BARON is a global optimization software which uses a branch-and-reduce optimization navigator to solve nonconvex global optimization problems to global optimality. The branch-and-reduce optimization navigator uses the combination of interval analysis and duality for “reduce” and adopts branch-and-bound concepts to accelerate the convergence. We test Baron on NEOS server. For the SDPBB method, diagonal relaxation method with sensitivity analysis is used. The results are shown in Table



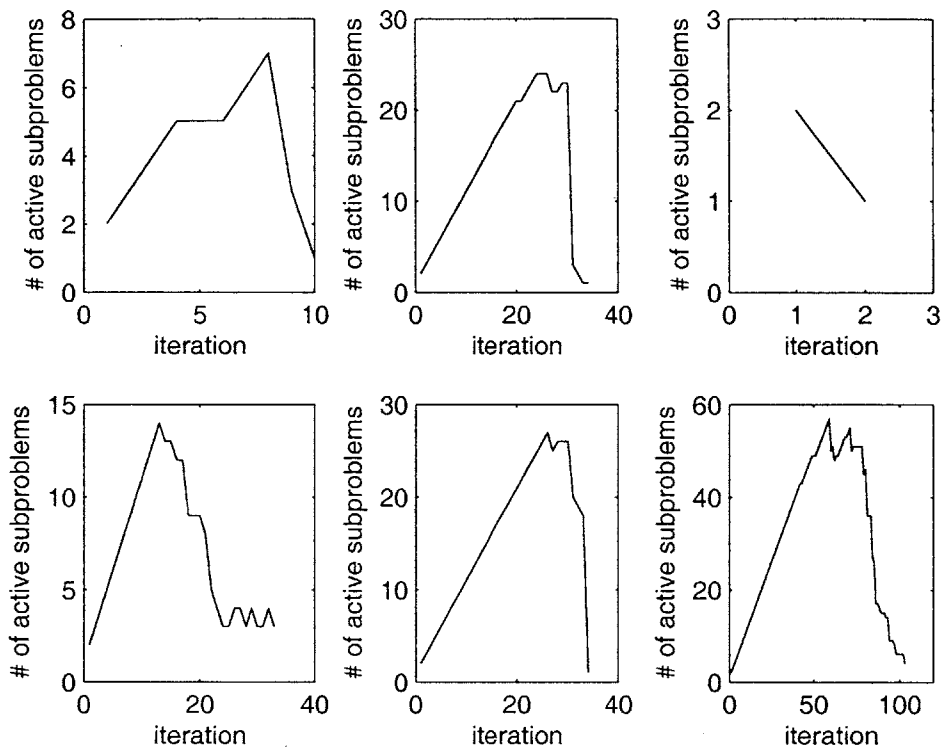


Figure 4-4: Number of active problems for each iteration during convergence process. Top row: three 20-dimensional instances; Bottom row: three 30-dimensional instances.

dimension		SDPBB	Interval branch-and-bound
4	time(s)	0.89	33.24
	lower bound	-66.694	-69.068
	gap	0.000	4.382
4	time(s)	0.9063	57.074
	lower bound	-67.276	-67.427
	gap	0.000	5.130
10	time(s)	2.33	7073.42
	lower bound	-365.105	-383.110
	gap	0.000	113.754
10	time(s)	1.69	5735.33
	lower bound	-233.445	-327.367
	gap	0.000	243.527

Table 4.3: Comparison of SDPBB and interval branch-and-bound method on four randomly generated instances of different sizes

dimension		SDPBB	BARON
10	time(s)	0.068	0.17
	lower bound	-365.104	-405.675
	gap	0.000	40.567
20	time(s)	1.5938	545.71
	lower bound	-866.670	-962.967
	gap	0.000	96.297
20	time(s)	12.34	5.38
	lower bound	-1713.000	-1903.333
	gap	0.001	190.333
30	time(s)	16.06	1000.03
	lower bound	-2454.300	-3875.922
	gap	0.000	1421.672

Table 4.4: Comparison of SDPBB and BARON on four randomly generated instances of different sizes

4.4. We can observe that the SDPBB is more efficient and can achieve much better bounds.

## Chapter 5

# Parallelization of Semidefinite Relaxation based Branch-and-bound Method

The SDPBB method works very well for those BoxQP problem that are not very large, but it takes a long time to solve large dimensional ( $\geq 50$  in dimension) BoxQP problems. In this chapter, we use a parallelization of the SDPBB method to increase the computational speed for large dimensional BoxQP problems. A multisection method that can be used for parallelization is introduced. In the parallel version of the SDPBB method, the SDP solver CSDP [1] is used to solve the SDP relaxations, and the Message Passing Interface(MPI) is used for the communications between different processors.

### 5.1 Multisection in SDPBB

Instead of splitting one box into two and solving the two newly generated problems during one iteration (bisection algorithm), we present a multisection algorithm which splits one box into several subboxes and solves them during one iteration. We use the multisection algorithm to parallelize the SDPBB method, and the parallel version of SDPBB is referred to as Parallel Semidefinite Relaxation Based Branch-and-Bound

(PSDPBB) method.

The idea of multisection on interval branch-and-bound method has been investigated by Csallner et. al. [11] and Casado et. al.[3]. In the multisection algorithm, more than one bisections are made in a single iteration step, which means that larger boxes in the search tree are skipped, and smaller boxes are investigated directly.

We compare the efficiencies of multisection algorithm and bisection algorithm using the following example. Assume the original box is  $B$  and we split according to the longest edge. There are three cases in which the multisection method may have advantages or disadvantages:

- In the first case,  $B$  is split into subboxes  $B_1$  and  $B_2$  in the first iteration of the bisection algorithm. If none of them are pruned in the first iteration,  $B_1$  and  $B_2$  could be split into subboxes  $B_{11}$ ,  $B_{12}$  and  $B_{21}$ ,  $B_{22}$  at the end of third iteration. Six subproblems are evaluated for the bisection algorithm. In the multisection algorithm,  $B$  is split into subboxes  $\hat{B}_{11}$ ,  $\hat{B}_{12}$ ,  $\hat{B}_{21}$ ,  $\hat{B}_{22}$  in the first iteration. While splitting according to the longest edge, the four subproblems obtained from the first splitting of the multisection method are the same as the four subproblem obtained from the third splitting of the bisection method. For the multisection method, only four subproblems are evaluated to achieve the same information as the bisection method. Therefore, the multisection algorithm has advantages in this case.
- In the second case, if one of the subproblems, say  $B_2$ , is pruned within the first two iterations of the bisection method or  $B_2$  is not chosen to be split in the third iteration of the bisection method, there are still four subproblems  $B_1$ ,  $B_2$ ,  $B_{11}$  and  $B_{12}$  to be evaluated in the bisection method. In these two cases, the two algorithms have almost the same efficiency.
- In the third case, if the bisection method stops at the first iteration, the bisection method could have less subproblems to evaluate. The multisection method has disadvantages in this case.

We can see from the previous chapter that for most large dimensional QP problems, none of the generated subproblems are pruned during the first half of the iterations. Therefore, for most large dimensional QP problems, the first case takes place most of the time, and thus the multisection method may have more advantageous than disadvantages.

One additional cost of multisection is to decide on which dimensions to split. For the bisection method, only one dimension needs to be calculated; for the multisection method, multiple dimensions may be required. Figure 5-1 shows two ways of splitting described by Csallner et. al. [11]. One is the multibisection method in which we choose multiple dimensions and then bisect each dimension. The other is the multisplitting method in which we choose one dimension and do multiple splitting on the chosen dimension. The multibisection method requires additional computation for calculating the dimensions to be split. The multisplitting method requires the same splitting information, the dimension to be split, as the bisection method. In our case, we choose the multibisection method because the multisplitting is more likely to generate trivial subproblems. For our method, the additional computation cost for choosing dimensions is trivial, but the cost of computing the subproblem is high.

Multisection method can be easily parallelized. The parallelization scheme of the multisection method is shown in section 5.2. A detailed comparison of the multisection method and the bisection method is shown in section 5.4.

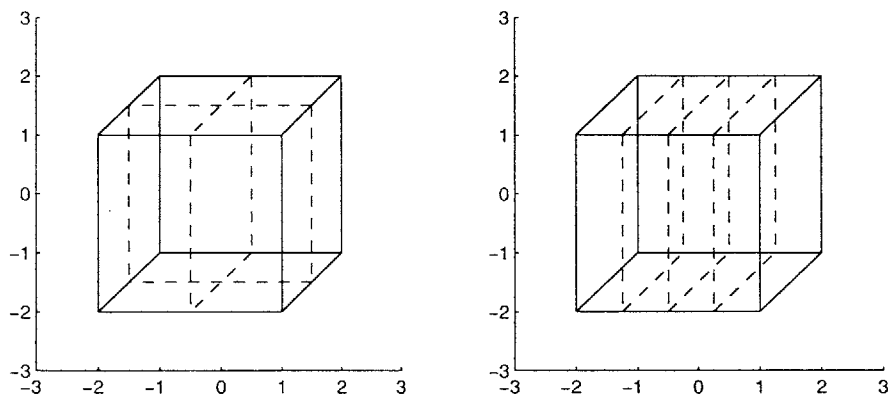


Figure 5-1: Two multisection methods. Left: multibisection; Right: multisplitting

## 5.2 Parallelization scheme

In our parallelization scheme, processor 0 works as a master processor who keeps all the information of active problems. Other processors work as slave processors, who only keep the original coefficients of the QP problem. As shown in Figure 5-2, the master processor initialize the program, sort and prune the list of active problems, and solve one SDP problem during one iteration. The slave processors only work as SDP solvers: they receive interval information from the master processor, calculate the SDP subproblem according to the received interval, and return the result to the master processor. The shaded steps in Figure 5-2 means that the processors are doing communications at that time; otherwise, the processors are doing computation. The number of processors decide the number of subproblems to be generated during one iteration. On one processor, only one SDP problem is solved during one iteration. Therefore, for multisection method with multibisection, the number of processors to be used cannot exceed twice the number of dimension of the QP problem.

In our parallelization scheme, communication cost is low: every slave processor receives  $2n$  double numbers, and send  $2 + n$  double numbers and 5 integer numbers during each iteration. We also know that the time for choosing one problem to split, splitting the intervals and pruning the list of active problems is much shorter than the time to compute a large-dimensional SDP problem. According to the efficiency formulation of parallel computation:  $\text{Efficiency} = \frac{\text{computation time}}{\text{total time}}$ , our parallelization scheme is efficient for large-dimensional SDP problems.

## 5.3 Implementation of the PSDPBB method

We use C/C++ and Message Passing Interface (MPI) to implement the PSDPBB method. We use the class `Actlist` to store the information of the active subproblems. The information includes the interval box, the bounds, the minimizer, and the sensitive dimensions of the subproblem. All the subproblems are stored in a doubly-linked list of `Actlist` objects.

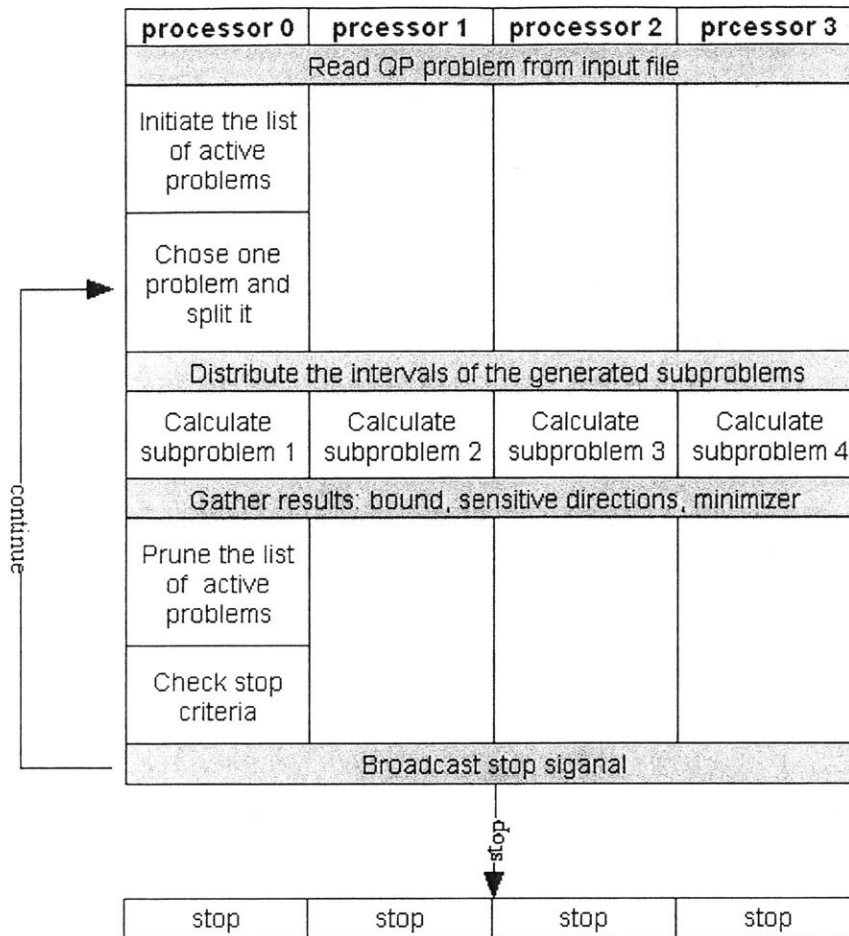


Figure 5-2: Parallelization scheme for PSDPBB

The file `qpdsolver.c` is written to use the CSDP subroutines to solve the diagonal SDP relaxation and the full-matrix SDP relaxation for the BoxQP problem. To solve these SDP relaxations in CSDP, we need to convert these SDP relaxations into standard form. The conversions are shown in Section 5.3.1. The constraint matrices are stored in a two-dimensional linked list structure of sparse matrices. The structure of our constraint matrix is shown in Figure 5-3.

LAPACK, BLAS, math and CSDP libraries are used for our programs. We have written a makefile that compiles the C program using the command `mpicc` and compiles the C++ program using the command `mpicxx`; the object files are linked using the command `mpicxx`. We have written an `sge` file that submits the executable file to

the computer cluster. The number of processors to be used, the name of the input file specifying the coefficients of the QP problem, and the method to be used are specified in the sge file.

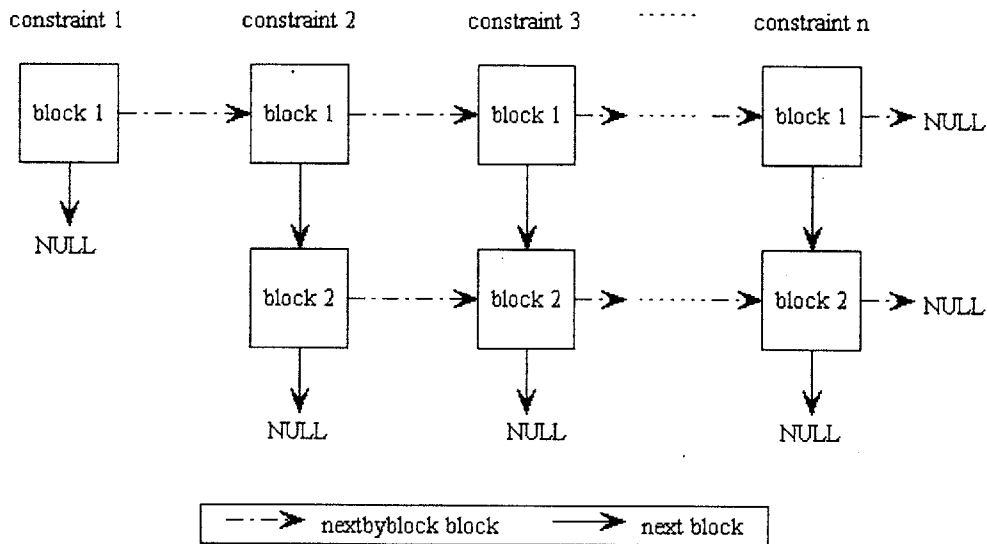


Figure 5-3: Two-dimensional linked list for constraint matrices in both diagonal and full-matrix method

### 5.3.1 Standardized SDP problems for CSDP solver

As we are using MPI, Yalmip and SeDuMi can no longer be used to solve SDP problems. Instead, we use the solver CSDP [1] to solve the SDP problems. CSDP is written in C, and it provides a standalone solver, Matlab routine and subroutine interface. We use the subroutine interface to solve our problem. To solve an SDP problem in CSDP, we need to convert the SDP problem into the standard form:

$$\begin{aligned}
 \text{(P)} \quad & \max \quad \text{tr}(CX) \\
 & X \succeq 0
 \end{aligned} \tag{5.1}$$



where

$$A(X) = \begin{bmatrix} \text{tr}(A_1 X) \\ \text{tr}(A_2 X) \\ \vdots \\ \text{tr}(A_m X) \end{bmatrix} \quad (5.2)$$

$$\begin{aligned} \text{(D)} \quad & \min \quad a^T y \\ & A^T(y) - C = Z \\ & Z \succeq 0 \end{aligned} \quad (5.3)$$

where

$$A^T(y) = \sum_{i=1}^m y_i A_i. \quad (5.4)$$

The standard form of diagonal SDP relaxation method and full-matrix SDP relaxation method are shown below.

For the diagonal method, we have the standard form

$$\begin{aligned} -\max \quad & - \begin{bmatrix} A & b & & \\ b^T & c & & \\ \hline & & 0 & \\ & & & \ddots \\ & & & & 0 \end{bmatrix} \bullet X \\ & A_i \bullet X = 1, i = 1, 2, \dots, n+1 \\ & X \succeq 0, \end{aligned} \quad (5.5)$$

where  $n$  is the dimension of the QP problem,  $A_1$  is a sparse matrix with only the entry  $(n+1, n+1)$  to be 1;  $A_i, i = 2, 3, \dots, n+1$  are the sparse matrices with only the entries  $(i-1, i-1)$  and  $(i+n, i+n)$  to be 1;  $X$  is a  $(2n+1) \times (2n+1)$  variable matrix.

For the full-matrix method, we have the standard form

$$\begin{aligned}
 -\max \quad & - \left[ \begin{array}{cc|c} A & b & \\ b^T & c & \\ \hline & & 0 \\ & & \dots \\ & & 0 \end{array} \right] \bullet X \\
 & A_1 \bullet X = 1 \\
 & A_{i,j} \bullet X = 2, i = 1, 2, \dots, n, j = i + 1, \dots, n \\
 & X \succeq 0,
 \end{aligned} \tag{5.6}$$

where there are in total  $m = 1 + \binom{2n}{2} = 1 + n(2n - 1)$  constraints.  $A_1$  is a  $(n + m) \times (n + m)$  sparse matrix with only the entry  $(n + 1, n + 1)$  to be 1.  $A_{i,j}$  can be calculated in the following manner.

The constraints of the full-matrix method, as shown in 3.7, can be converted as

$$\begin{aligned}
 (1 + x_i)(1 + x_j) &\geq 0 \\
 \Rightarrow -x_i - x_j - x_i x_j + s_{g(i,j)} &= 1 \\
 \Rightarrow -2x_i - 2x_j - 2x_i x_j + s_{g(i,j)} &= 2 \\
 \Rightarrow \hat{A}_{i,j} \bullet \begin{bmatrix} xx^T & x \\ x & 1 \end{bmatrix} + s_{g(i,j)} &= 2, s_{g(i,j)} \geq 0,
 \end{aligned} \tag{5.7}$$

where  $\hat{A}_{i,j}$  is a sparse matrix with the entries  $(i, j)$ ,  $(j, i)$ ,  $(i, n + 1)$ ,  $(n + 1, i)$ ,  $(j, n + 1)$ ,  $(n + 1, j)$  to be  $-1$ ; the entry  $(i, j)$  has the value  $-2$  if  $i$  equals to  $j$ . Therefore, we have

$$A_{i,j} = \begin{bmatrix} \hat{A}_{i,j} & & & & \\ & \dots & & & \\ & & & 1 & \\ & & & & \dots \end{bmatrix}.$$

The entry of the element 1 is  $(g(i, j) + n + 1, g(i, j) + n + 1)$ , where  $g(i, j)$  represents

solver	splitting method	# of processors	iteration	time (sec.)
SDPBB	w/o sensitivity analysis	1	6515	2864.01
	w/ sensitivity analysis	1	1765	764.34
PSDPBB	w/o sensitivity analysis	2	4044	1208.99
		4	2553	778.78
		8	2066	657.10
		16	1595	514.86
	w/ sensitivity analysis	2	1770	547.21
		4	876	217.94
		8	492	322.19
		16	393	127.33

Table 5.1: Convergence comparison for PSDPBB using different number of processors for solving the 50-dimensional instance

the sequence number of  $A_{i,j}$  among all the constraint matrices.

## 5.4 Results and comparison

In this section, we test a series of problems to show the efficiency of our methods. All the problems are tested on a 64-node computer cluster.

To compare the number of processors used and the convergence, we test a randomly generated 50-dimension BoxQP problem using diagonal relaxation method. The precision requirement is set such that the absolute error is to be less than 0.1. The result is shown in Figure 5-4 and Table 5-4. We can see that the more processors used, the faster the method is. Among the multisection methods without sensitivity analysis, multisection on one dimension (2 processors) solves the fewest subproblems. Among the multisection methods with sensitivity analysis, multisection on two dimensions (4 processors) solves the fewest subproblems.

To compare the diagonal method and the full-matrix method, we present the time and iteration comparison in Figure 5-5. Ten randomly generated 10-dimensional instances are tested. From Figure 5-5, we can see that the full-matrix method usually finishes in fewer iterations, but it takes much longer time.

To show the efficiency of the PSDPBB method for problems of different dimen-

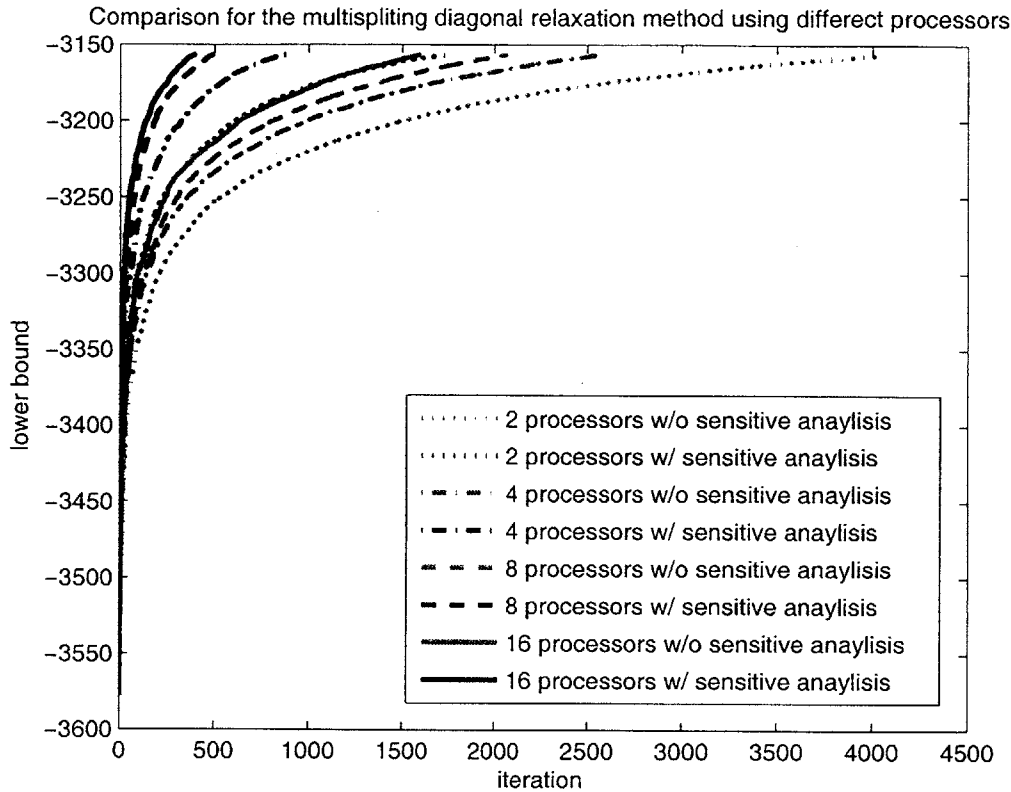


Figure 5-4: Convergence comparison for using different number of processors on a 50-dimensional instance

sions, we tested the PSDPBB method on the BoxQP problems introduced by Vandebussche and Nemhauser [23] and Burer[2]. The dimensions of the instances from Vandebussche and Nemhauser range from 20 to 60, and the dimensions of the instances from Burer range from 70 to 80. For each problem, 16 processors are used. Sensitivity analysis is always used in this section. Figure 5-6 shows the comparison of computation time and number of iterations for instances of different sizes.

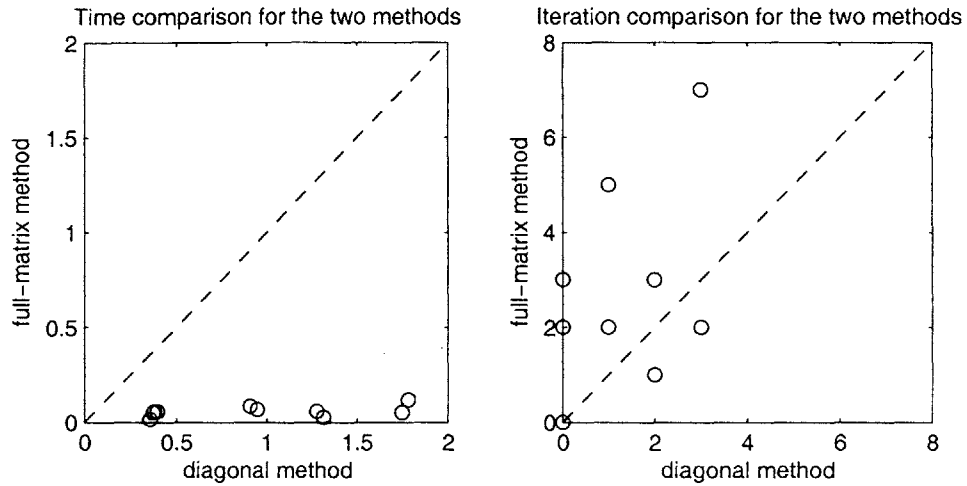


Figure 5-5: Comparison of diagonal and full-matrix methods for 10-dimensional instances

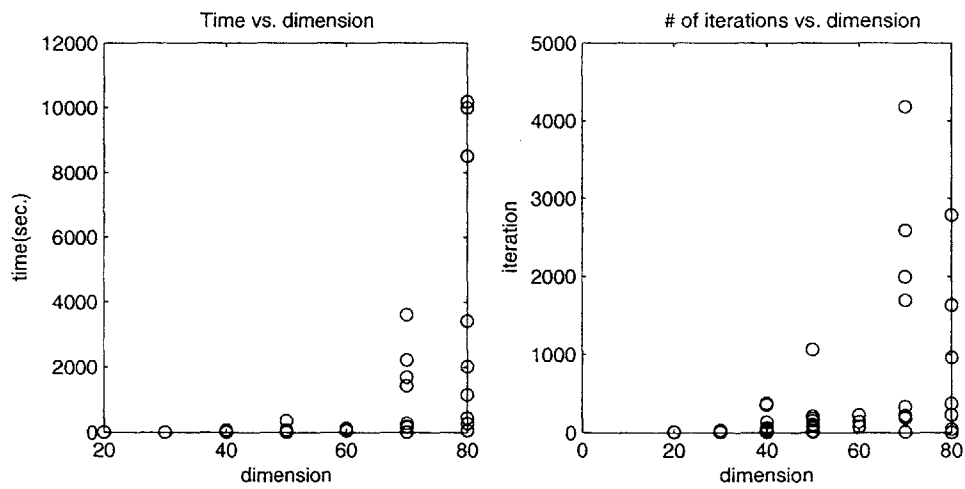


Figure 5-6: Computation result for instances of dimensions 20 to 80

THIS PAGE INTENTIONALLY LEFT BLANK

# Chapter 6

## Conclusions and Future Work

### 6.1 Conclusions

The SDPBB method has been shown to be an accurate and effective method for nonconvex BoxQP problems. By adopting different SDP relaxation methods to the SDPBB method, we have found the best SDP relaxation method for SDPBB: diagonal relaxation method. By comparing different splitting methods, we have found the best splitting method for SDPBB: splitting using sensitivity analysis. The parallel version of SDPBB method, PSDPBB, improves the speed of solving nonconvex BoxQP problems by using multibisection method. Two software packages are developed using SDPBB and PSDPBB methods. The software packages are developed in C/C++ language based on the SDP solver CSDP, and can be easily installed and used as a standalone solver. A user's guide for the software packages is given in Appendix A.

### 6.2 Future work

The speed of the SDPBB method may be improved by dynamic SDP relaxation method, which means that for the full-matrix SDP relaxation method, if we can choose some of the most important entries in the matrix  $\Lambda_{full}$ , we can get bounds of good quality within a relatively short time. One possible way of doing that is the dynamic method described below.

From the expression:

$$\begin{aligned}
& x^T \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} x \succeq x^T \left( \begin{bmatrix} I & e \\ -I & e \end{bmatrix}^T \Lambda \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \right) x \\
& = x^T \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} x \succeq \Lambda \bullet (\tilde{x}\tilde{x}^T),
\end{aligned} \tag{6.1}$$

where

$$\tilde{x} = \begin{bmatrix} I & e \\ -I & e \end{bmatrix} x,$$

we denote

$$\tilde{X} = \tilde{x}\tilde{x}^T.$$

As we have discussed in Section 4.3, the lower bound for the QP problem,  $\gamma$ , will be tight if the right hand side of the equation is very close to zero. Therefore, if the bound is tight, for the entries in  $\tilde{X}$  that have large values, the corresponding entries in  $\Lambda$  should be close to zero. Therefore, these entries in  $\Lambda$  can be ignored. For implementation, we can start with the diagonal relaxation method to get an  $\tilde{X}$ , and then calculate the valuable entries in  $\Lambda$ . After that, we can recalculate the SDP relaxation using the new  $\Lambda$ . We call this SDP relaxation method the dynamic method. We have tested on lower dimensional instances ( $\leq 20$  in dimension). The results show that the lower bound achieved from the dynamic method using  $n^2$  entries in  $\Lambda$  has almost the same quality as the full-matrix method.

Another advantage of the dynamic method is that during the branch-and-bound process, the information of the last iteration can be used when forming  $\Lambda$  in the current iteration. We can calculate a new  $\tilde{x}$  by scaling the  $\tilde{x}$  obtained from last iteration and use the new  $\tilde{x}$  to calculate the important elements of  $\Lambda$ .

The problem of this method is that the  $\tilde{x}$  can not give much information about the importance of the entries in  $\Lambda$  when the the bounds calculated by  $\tilde{x}$  are very loose. More work needs to be done to improve the dynamic method for higher-dimensional QP problems.



# Appendix A

## SDPB and PSDPB User's Guide

### A.1 Introduction

SDPB and PSDPB are two software packages that solve BoxQP problems in the format:

$$\begin{aligned} f_{\text{BoxQP}} = \min_x x^T A x + 2b^T x + c \\ \text{subject to } -1 \leq x_i \leq 1, \quad i = 1, 2, \dots, n, \end{aligned} \tag{A.1}$$

where  $A$ ,  $b$ ,  $c$  are coefficients for the QP problem:  $A$  is an  $n \times n$  matrix,  $b$  is an  $n$ -dimensional vector and  $c$  is a scalar;  $x$  is an  $n$ -dimensional vector of variables; and the dimension of the problem is  $n$ . The algorithm for the two softwares is the semidefinite relaxation based branch-and-bound method. SDPB is a sequential version and PSDPB is a parallel version to be run using MPI. For each version, four methods of solving BoxQP problems are provided, which are

1. diagonal relaxation method without sensitivity analysis
2. diagonal relaxation method with sensitivity analysis
3. full-matrix relaxation method without sensitivity analysis
4. full-matrix relaxation method with sensitivity analysis

### A.1.1 SDP relaxation methods and sensitivity analysis

Both the diagonal relaxation method and full-matrix relaxation method are SDP relaxation methods. The diagonal relaxation method solves the following primal-dual SDP relaxation.

$$\begin{aligned}
 (P) \quad & \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\
 \text{subject to} \quad & X_{n+1,n+1} = 1 \\
 & X_{i,i} \leq 1, \quad i = 1, 2, \dots, n \\
 & X \succeq 0.
 \end{aligned} \tag{A.2}$$

$$\begin{aligned}
 (D) \quad & \max \gamma \\
 \text{subject to} \quad & \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \succeq \begin{bmatrix} -\Lambda & \\ & \text{tr}\Lambda \end{bmatrix} \\
 \text{where} \quad & \Lambda = \text{diag}(\lambda_i), \quad i = 1, 2, \dots, n.
 \end{aligned} \tag{A.3}$$

The full-matrix relaxation method solves the following primal-dual pair:

$$\begin{aligned}
 (P) \quad & \min_X \begin{bmatrix} A & b \\ b^T & c - \gamma \end{bmatrix} \bullet X \\
 \text{subject to} \quad & X_{n+1,n+1} = 1 \\
 & 1 + X_{i,n+1} + X_{j,n+1} + X_{i,j} \geq 0, \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n \\
 & 1 + X_{i,n+1} - X_{j,n+1} - X_{i,j} \geq 0, \quad i, j = 1, 2, \dots, n \\
 & 1 - X_{i,n+1} - X_{j,n+1} + X_{i,j} \geq 0, \quad i = 1, 2, \dots, n, j = i + 1, i + 2, \dots, n \\
 & X \succeq 0
 \end{aligned} \tag{A.4}$$

$$(D) \quad \max \lambda$$

subject to

$$\begin{aligned} \begin{bmatrix} A & b \\ b^T & c - \lambda \end{bmatrix} &\preceq \begin{bmatrix} I & e \\ -I & e \end{bmatrix}^T \Lambda_{full} \begin{bmatrix} I & e \\ -I & e \end{bmatrix} \\ \lambda_{i,j} &\geq 0, \quad i = 1, 2, \dots, 2n, \\ &\quad j = i + 1, \dots, 2n. \end{aligned} \tag{A.5}$$

where

$$\Lambda_{full} = \begin{bmatrix} 0 & \lambda_{1,2} & \cdots & \lambda_{1,2n} \\ \lambda_{1,2} & 0 & & \\ & \ddots & \ddots & \ddots \\ \vdots & & & 0 & \lambda_{2n-1,2n} \\ \lambda_{1,2n} & \cdots & \lambda_{2n-1,2n} & 0 \end{bmatrix}.$$

The diagonal relaxation method takes shorter time for each iteration, and takes more iterations to solve the problem. The full-matrix relaxation method takes longer time for each iteration but it requires fewer iterations. To sum up, the diagonal relaxation method is usually quicker than the full-matrix relaxation method.

The sensitivity analysis decides the splitting dimension for branch-and-bound method. The algorithm splits the box along the most sensitive dimension of the box if sensitivity analysis is used. Without sensitivity analysis, the algorithm splits along the longest edge of the box. Sensitivity analysis requires slightly more computation cost for each iteration, but the method with sensitivity analysis always finishes in much fewer iterations. Therefore, the method with sensitivity analysis is always faster than that without sensitivity analysis.

The rest of this document describes the installation procedure and usage of the two solvers.

## A.2 Installation of SDPBB and PSDPBB

The source codes for SDPBB and PSDPBB can be found in the folder `SDPBBsolver`. The source code for SDPBB can be found in the folder `SDPBBsolver/SDPBB`, while the source code for PSDPBB can be found in the folder `SDPBBsolver/PSDPBB`. The software package CSDP should be installed in advance. The version CSDP5.0 is recommended. A guide for CSDP5.0 can be found in [1].

The following steps show the installation procedure of SDPBB:

1. In the file `Makefile` under directory `SDPBBsolver/SDPBB`, specify the directory of CSDP in the section “environment parameters to be specified during installation”. If LAPACK and BLAS are available, specify the directory of these software in the same section, and uncomment the line for LAPACK and BLAS in the section “The library and the link options”.
2. Use the command `make` under directory `SDPBBsolver/SDPBB` to generate executable file `SDPBB`.
3. Run the test file using command `./SDPBB ../testprob/randQP4.txt -diag -sens`. If the result is printed on the screen, the solver has been successfully installed.

For PSDPBB, the software `mpich` is required. The version `mpich v1.2.7` is recommended. Installation for PSDPBB is the same as SPDBB for the first two steps except that everything is done under folder `SDPBBsolver/PSDPBB`. The third step of installing PSDPBB is

- Run the test file using the command `qsub test_psdpbb_sge.sh`. An id will be given after submitting the SGE file. Use the command `cat psdpbb.o<id>` to see the output file. If no error is displayed, the solver has been successfully installed.

### A.3 Usage of SDPBB

After installation, a standalone SDPBB solver can be used in the following format:

```
./SDPBB inputfile <relaxation method> <sensitivity analysis>
                                [>outputfile]
```

The option <relaxation method> could be

**-diag** diagonal relaxation method

**-full** full-matrix relaxation method

The option <sensitivity analysis> could be

**-noSens** do not use sensitivity analysis

**-sens** use sensitivity analysis

The inputfile specifies the coefficients  $Q$ ,  $b$ ,  $c$  of BoxQP. The format of inputfile is shown in figure A-1.

4		→ dimension of the problem		
-16.359574	-1.943416	-1.910960	-4.029852	} → A
-1.943416	13.542050	9.425036	8.156076	
-1.910960	9.425036	-12.484713	1.502217	
-4.029852	8.156076	1.502217	4.976934	
6.676918	-6.663029	-6.772794	0.967719	→ $b^T$
-1.146471				→ c

Figure A-1: Input file for SDPBB and PSDPBB

User can specify the name of the output file. If no output file is specified, the result will be printed on the screen.

Additionally, the maximum splitting number (`maxsplit`), the accuracy of the result (`relerror/abserror`) and the print level (`printlevel`) can be set in the file `param.sdpbb`. This file should be in the directory `SDPBBsolver/SDPBB`. Two sample `param.sdpbb` files are shown as follows:

maxsplit	5000	maxsplit	5000
abserror	0.1	relerror	0.01
printlevel	1	printlevel	0

For the case on the left, since `abserror` is used, the algorithm stops when the difference between the upper bound and the lower bound is less than 0.1. For the case on the right, since `relerror` is used, the algorithm stops when the difference between the upper bound and the lower bound is less than 1% of the value of lower bound.

If the print level is set to 1, converting the output file to `.m` file and running directly in Matlab generates a vector called `boundRecord` which records the upper bound and lower bound on the best interval (the one with the smallest lower bound) of each iteration. If the print level is set to 0, the bounds information for each iteration will not be printed in the output file. If there is no `param.sdpbb` file, the algorithm will use the default parameters as `maxsplit = 5000`, `abserror = 0.1`, and `printlevel = 1`.

## A.4 Usage of PSDPBB

The usage of PSDPBB is mostly the same as that of SDPBB except that the executable file is submitted to the cluster through an SGE file. A sample SGE file is shown in figure A-2.

The SGE file is submitted to the cluster by the command

```
qsub -N <outputfile> -pe mpich <number of processors to use>
      qpbb_sge.sh.
```

```
#!/bin/sh
### -N psdpbb
#$ -S /bin/sh
#$ -cwd
#$ -V
### -pe mpich 4
progName=psdpbb
machineFileLocation=/home/sma5232/hydra_info/compute_node
mpirun -np $NSLOTS -machinefile $TMPDIR/machines $progName <inputfile> <relaxation
method> <sensitivity analysis>
```

Figure A-2: Sample SGE file: qpbb\_sge.sh

THIS PAGE INTENTIONALLY LEFT BLANK



# Bibliography

- [1] B. Borchers. CSDP 5.0 user's guide. World Wide Web, <http://euler.nmt.edu/~brian/csdpuser.pdf>, 2005.
- [2] S. Burer and D. Vandembussche. A finite branch-and-bound algorithm for non-convex quadratic programming via semidefinite relaxations. Manuscript, Department of Mechanical and Industrial Engineering, University of Illinois Urbana-Champaign, Urbana, IL, USA, June 2005. Revised April 2006 and June 2006. Submitted to *Mathematical Programming*.
- [3] L. G. Casado, I. García, and T. Csendes. A new multisection technique in interval methods for global optimization. *Computing*, 65(3):263–269, 2000.
- [4] C. A. Floudas and V. Visweswaran. Quadratic optimization. In R. Horst and P. M. Pardalos, editors, *Handbook of Global Optimization*, Dordrecht, The Netherlands, 1995. Kluwer Academic Publishers.
- [5] T. Fujie and M. Kojima. Semidefinite programming relaxation for nonconvex quadratic programs. Technical Report Research Report on Information Sciences B-298, Tokyo Institute of Technology, 1995.
- [6] M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.
- [7] N. I. M. Gould and P. L. Toint. Numerical methods for large-scale non-convex quadratic programming. In A. H. Siddiqi and M. Kočvara, editors, *Trends in In-*

- dustrial and Applied Mathematics*, pages 149–179. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2002.
- [8] E. R. Hansen. *Global Optimization Using Interval Analysis*. 2th edition, 2005.
- [9] P. Hansen, B. Jaumard, M. Ruiz, and J. Xiong. Global minimization of indefinite quadratic functions subject to box constraints. Report G-91-54, GERAD, Ecole Polytechnique, Université McGill, Montreal, Quebec, Canada, 1991.
- [10] Y. Huang and S. Zhang. Approximation algorithms for indefinite complex quadratic maximization problems. Technical Report SEEM2005-03, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, 2005.
- [11] M. C. Markót, T. Csendes, and A. E. Csallner. Multisection in interval branch-and-bound methods for global optimization ii. numerical tests. *Journal of Global Optimization*, 16(4):371–392, 2000.
- [12] I. Maros. A repository of convex quadratic programming problems. *Optimization Methods and Software*, 11:671–681, 1999.
- [13] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, 1979.
- [14] A. Nemirovski, C. Roos, and T. Terlaky. On maximization of quadratic form over intersection of ellipsoids with common center. *Mathematical Programming*, 86(3):463–473, 1999.
- [15] Y. E. Nesterov. Semidefinite relaxation and nonconvex quadratic optimization. *Optimization Methods and Software*, 9:141–160, 1998.
- [16] Y. E. Nesterov, H. Wolkowicz, and Y. Ye. Semidefinite programming relaxations of nonconvex quadratic optimization. In H. Wolkowicz, R. Saigal, and L. Vandenberghe, editors, *HANDBOOK OF SEMIDEFINITE PROGRAMMING: Theory, Algorithms, and Applications*, pages 361–420. Kluwer Academic Publishers, Boston, MA, 2000.

- [17] H. Ratschek and J. Rokne. *New computer methods for global optimization*. Halsted Press, New York, NY, USA, 1988.
- [18] N. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8:201–205, 1996.
- [19] S. Skelboe. Computation of rational functions. *BIT*, 14:87–95, 1974.
- [20] M. J. Todd. Semidefinite optimization. *Acta Numerica*, 10:515–560, 2001.
- [21] P. Tseng. Further results on approximating nonconvex quadratic optimization by semidefinite programming relaxation. Technical report, Department of Mathematics, University of Washington, Seattle, WA, 2001.
- [22] L. Vandenberghe and S. P. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, March 1996.
- [23] D. Vandembussche and G. L. Nemhauser. A branch-and-cut algorithm for nonconvex quadratic programs with box constraints. *Mathematical Programming*, 102(3):559–575, 2005.
- [24] D. Vandembussche and G. L. Nemhauser. A polyhedral study of nonconvex quadratic programs with box constraints. *Mathematical Programming*, 102(3):531–557, 2005.
- [25] Y. Ye. Approximating global quadratic optimization with convex quadratic constraints. Technical report, Department of Management Sciences, The University of Iowa, Iowa City, Iowa 52242, July 1998.
- [26] Y. Ye. Approximating quadratic programming with bound and quadratic constraints. *mp*, 84:219–226, 1999.
- [27] S. Zhang. Quadratic minimization and semidefinite relaxation. *Mathematical Programming*, 87(3):453–465, 2000.