低次元分数和計画問題に対する効率的なアルゴリズムに関する研究

Muroran Institute of Technology

Doctoral Dissertation

Efficient Algorithms for Solving the Sum of Linear Ratios Problem with Lower Dimension

Yongwen Hu

Supervisors:
Prof. Jianming Shi
Assoc. Prof. Shinya Watanabe

A dissertation submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy

in the

Computatioanl Intelligence Laboratory
Production Information System Engineering

September 2014

Declaration of Authorship

I, Yongwen Hu, declare that this dissertation titled, 'Efficient Algorithms for Solving the Sum of Linear Ratios Problem with Lower Dimension' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Thanks to my solid academic training, today I can write hundreds of words on virtually any topic without possessing a shred of information, which is how I got a good job in journalism."

Dave Barry

Muroran Institute of Technology

Abstract

School of Engineering
Production Information System Engineering

Doctor of Philosophy

Efficient Algorithms for Solving the Sum of Linear Ratios Problem with Lower Dimension

by Yongwen Hu

The *Sum of Linear Ratios* (SOLR) optimization problem, well know as an $\mathcal{NP}$-hard problem, has wide applications. A variety of problems can be formulated into the SOLR problem, and these problems are characterized by a small number of variables but a large number of ratios. In this dissertation, we are extremely interested in the SOLR problem where the native dimension is lower.

A natural approach to cope with the $\mathcal{NP}$-hardness is to develop an efficiently approximation algorithm that is guaranteed to obtain a better approximation to the optimal solution. The DIRECT algorithm is one of such approximation approach. We evaluate the performance of DIRECT for solving the SOLR problem with lower dimension by conducting numerical experiments. The results show that the DIRECT algorithm could find an optimal solution of the SOLR problem with an extremely high probability (99.99%) and less computational time spent within a given tolerance.

Another approach for designing approximation algorithms is mostly fueled by convex optimization techniques. Recently, Carsslon and Shi (2013) have proposed an algorithm for solving SOLR problem with lower dimension. Carsslon and Shi casted the SOLR problem into its equivalent problem with linear objective and a set of linear and nonconvex quadratic constraints. By dropping the nonconvex quadratic constraints out, they proposed a liner relaxation for the SOLR problem and developed a branch-and-bound algorithm to solve the SOLR problem with lower dimension.

To circumvent the nonconvex quadratic constraints, we make a linear relaxation for the nonconvex constraints by introducing some new auxiliary variables instead of dropping them out. Therefore, this linear relaxation is generally tighter than the previous one. With the help of the new relaxation, we propose a branch and bound algorithm for solving the SOLR problem based on bisection branching rules. We also prove the convergence of the proposed algorithm. The numerical experiments are conducted and the results indicate that our method is much more efficient than the previous one. More precisely, the number of branches is heavily reduced at least to about 2% in average of the previous algorithm.

In addition, we develop a branch and bound algorithm for globally solving the SOLR problem based on new branching rules (advanced branching rules). The advanced branching rules guarantee that, if the rectangle that contains the current best solution is divided into two sub-rectangles, the current best solution will be in the both two sub-rectangles in next iteration. We also conduct numerical experiments to investigate the performance of the proposed algorithm with the new branching rules. The results indicate that the advanced branching rules can accelerate the process for finding approximation solution.

Finally, we reformulate the SOLR problem into an SDP programming, and give comparisons of several SDP relaxations.

Acknowledgements

I owe my deepest gratitude to my supervisors Prof. Jianming Shi and Assoc. Shinya Watanabe. It is their invaluable guidance, advice and constant support and encouragement that enable me to finish this dissertation. I have learned a lot from working with Prof. Shi. He taught me about doing research and presentation. We also had many interesting and stimulating discussions during our weekly meetings. After Prof. Shi moved to Tokyo University of Science, I still have numerous opportunities to discuss the difficulties of my research with him. I am greatly indebted to him. Assoc. Shinya Watanabe also gave me tremendous help on my research. I would like to thank him for interesting research discussion and invaluable comments on my topics.

I am especially grateful to Prof. Hiroyuki Shioya who generously gave me his time to discuss research problems with me.

Many thanks to Prof. Yanjun Wang from Shanghai University of Finance and Economics. She shared valuable ideas with me during her visit in winter 2012 in Muroran, which make me understand further on semidefinite programming.

This dissertation would not be possible without the encouragement and support of many friends. I would like to express my sincerest thanks to them all. My special thanks go to Jun Mao, Woramol Chaowarat, Qunpo Liu, Faqiang Su for their shared experience, assistance and discussions that made my PhD life hassle free. I would also like to thank everyone in computational intelligence lab, for their help and the fun they brought to my life. I will never forget the wonderful times we spent together.

Finally, I would like to express my deepest gratitude to my family, for their love and support over the years. This dissertation is dedicated to them all.

Contents

List of Figures

List of Tables

Dedicated to my family

Chapter 1

Introduction

Optimization problems appear in many practically relevant areas of our life. Typical application areas include project scheduling and staffing, production planning, transportation, investment planning and many more. Optimal solutions in these applications have significant economical and social impact. And better engineering designs often result in lower implementation and maintenance costs, faster execution, and more robust operation under a variety of operating conditions. In this dissertation, we study a convex optimization method for solving a class of non-convex problems. A convex optimization problem, stated in Boyd and Vandenberghe [13], is one of the form

$$\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_i(\boldsymbol{x}) \leq b_i, i = 1, \ldots, m, \end{array} \tag{1.1}$$

where the functions $f_0, \ldots, f_m : \mathbb{R}^n \to \mathbb{R}$ are convex, i.e., satisfy

$$f_i(\alpha \boldsymbol{x} + \beta \boldsymbol{y}) \leq \alpha f_i(\boldsymbol{x}) + \beta f_i(\boldsymbol{y})$$

for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and all positive α, β such that $\alpha + \beta = 1$. $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\top \in \mathbb{R}^n$ is decision variable. Clearly, if $f_0, \ldots, f_m$ are linear, the problem (1.1) is a linear programming (LP) problem which is one of the most well-studied fields within optimization.

If at least one of $f_0, \ldots, f_m$ is not convex, then problem (1.1) becomes non-convex optimization problem, which in general is known to be $\mathcal{NP}$-hard. And

such problems arise naturally in various areas in the real word. The problem of optimizing one or several ratios of functions is called a *fractional programming problem* which was introduced by Charnes and Cooper [19], and the general form of sum of ratios can be formed as following

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) = \sum_{i=1}^{p} \frac{h_i(\boldsymbol{x})}{g_i(\boldsymbol{x})} \\ &\text{subject to} \quad \boldsymbol{x} \in X. \end{aligned} \tag{1.2}$$

Fractional programming is one of the most successful fields today in non-linear optimization. In the special case when g, h are both convex quadratic functions, or both d.c. functions, problem (1.2) was studied by Gotoh and Konno [61] and Jianming Shi [26], respectively. When g, h are both linear functions, problem (1.2) is called the sum of linear ratios (SOLR) problem. It is an important branch of fractional optimization and has attracted many researchers' concern for several decades. In this dissertation, we mainly focus on the SOLR problem with lower dimension.

1.1 The Sum of Linear Ratios Problem

The SOLR problem has attracted the interest of many researchers since 1970s because of its various applications and significant challenges for getting an optimal solution; Many various problems arising in engineering, economics, management science, transportation problems and other disciplines can be stated as the problem of optimizing a sum of linear ratios [80]. In this dissertation, we consider the SOLR problem defined as follows

$$\begin{aligned} &\text{minimize} \quad f(\boldsymbol{x}) = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \\ &\text{subject to} \quad \boldsymbol{x} \in X = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} \leq \boldsymbol{c}, \boldsymbol{x} \geq \boldsymbol{0}\}, \end{aligned} \tag{1.3}$$

where $p \geq 2$, $\boldsymbol{n}_i, \boldsymbol{d}_i$ are both vector in $\mathbb{R}^n$ for $i = 1, \ldots, p$, a_i, b_i are real numbers for $i = 1, \ldots, p$, A is an $m \times n$ matrix, $\boldsymbol{c}$ is a vector in $\mathbb{R}^m$.

The SOLR problem has attracted the interest of practitioners and researchers for more than 40 years. This is because, from a practical point view, the SOLR

problem has a wide variety of applications such as multi-stage stochastic shipping problem [2], multi-objective bond portfolio optimization problem [54, 56], minimum ratio spanning tree problem [79], finance and investment [23], and a number of geometric problems etc. [20]. In addition, from an academic research point of view, the SOLR problem poses some theoretic and computational challenges. It is well know that the SOLR problem generally has multiple locally optimal solutions which are not globally optimal solution even for low dimension [70]. Furthermore, the SOLR problem is $\mathcal{NP}$-hard [63]; actually the 3-partition problem in [64], an $\mathcal{NP}$-hard problem, can be written in the form of problem (1.3).

Since the non-convexity of problem (1.3), traditional convex optimization methods can not be applied directly for it. Fortunately, some complexity results on fractional functions and related problems are known and several algorithms have been proposed and some typically important results have been given for solving for problem (1.3).

For example, solving the SOLR problem with n-variable is $\mathcal{NP}$-complete, even when the sum has only two or few terms [34]. Chandrasekaran [18] reduced the *Minimal Ratio Spanning Trees* (MRST) as the SOLR problem and proved that problem (1.3) is $\mathcal{NP}$-complete problem in the case that the denominator is allowed take on negative values on its feasible region. Schaible [70] showed in 1977 that the SOLR problem is neither quasiconcave nor quasiconvex on X even the objective function is the sum of a linear term and a ratio term, and generally the problem has one or more locally optimal solutions which are not global optimal solution. In such a situation, solving problem (1.3) is quite challenging.

Quite a few algorithms have been proposed for solving it in the past decades, (see [21, 52, 57]). First, some of these algorithms were intended only for the case with $p \leq 2$ [15, 57], and many algorithms were proposed for solving the SOLR problem with only a few ratio degrees (less than 10) over a polytope (see [33, 52, 79]). An efficient simplex-type algorithm were proposed by Charnes and Cooper [19] for the case $p = 1$ in 1962. In fact, the objective function for this case is both quasi-convex and quasi-concave. When $p = 2$, the objective is no longer quasi-convex. Nevertheless, a parametric algorithm [57] can solve it in an efficient manner.

However, problem (1.3) is much more difficult when $p \geq 3$. When $p = 3$, Konno [52] has developed an effective heuristic algorithm which is an extension of the parametric simplex algorithm for $p = 2$. Some algorithms [10, 30, 33, 52, 79, 81] have been proposed for such case. Also, Benson proposed an algorithm for the problem (1.3), and a global solution for the problem (1.3) can be obtained by using the primal-relaxed dual approach. Falk and Palocsay [33] proposed an algorithm to solve problem (1.3) when $p \geq 3$. In this algorithm, each ratio in the original problem turns to be a new variable in the image space, and the optimal solution is easy to obtain in certain directions in the image space.

To minimize the problem (1.3), Wang et al. [88] used a linearization technique twice by the nature of exponential and logarithmic functions to make a linear relaxation programming for its original problem. Jiao et al. [46] found a global optimizer for problem (1.3) using a new pruning technique with linear rexalation.

Konno et al. [55] developed an algorithm that can be used to solve globally the problem obtained from (1.3) by minimizing the objective function of (1.3) in 1994. This algorithm transforms the problem into an equivalent concave minimization problem with $2p$ variables, and the globally optimal solution can be obtained by using outer approximation. Later, Benson [9] proposed a branch-and-bound algorithm for globally solving the problem (1.3) via a branch-and-bound search. Some algorithms use simplex method-like pivoting or the parametric simplex method and apply only when X is polyhedral. The algorithm of Falk and Palocsay [33] searches the nonconvex program on a image space literally until the optimal solution is abstained. In fact, problem (1.3) can not be solved in a reasonable time with a large number of ratios (e.g. , $p = 15$) by using such approach. In addition, Depetrini and Locatelli [30] proposed an algorithm like sampling method to get an approximation optimal solution, but it would take much CPU time for solving problem (1.3) even when $p = 3$. Most of these algorithms based on branch and band have been proposed for solving problem (1.3), see [53].

For an excellent review of the applications, theoretical results, algorithms of the SOLR problem, the reader is reffered to [73]. And some specific applications of the SOLR problem, especially for lower dimension, will be described in next section.

1.2 Applications

In this section, we will give some applications for the SOLR problem. A single-ratio (e.g. [71, 72]) with numerator and denominator can be represented output and input, profit and cost, capital and risk. A multitude of applications of the SOLR problem can be envisioned in this way.

In addition, a variety of problems in application domains which includes *layered manufacturing*, *camera resectioning*, *homography estimation*, *star cover problem*, *triangulation problem* and others, can be appropriately formulated as the SOLR problem with lower dimension. The problems of this class are characterized by a small number of variables and a large number of ratios. The following are some specific examples with detail description.

Camera Resectioning A recursive *Camera Resectioning* algorithm is proposed for solving the problem in [51]. In the algorithm, the estimation of camera motion can be simply computed by

$$E[X_t|Y_t] \approx \sum_{i=1}^{M} w_t^{(i)} x_t^{(i)}.$$

Here M is the number of samples (usually M is large), $w_t^{(i)}$ is the weight of the *i-th* sample, and $x_t^{(i)}$ is the modes of the probability distribution of camera system state at time t for sample i. Clearly, this problem can be formulated into problem (1.3) in $\mathbb{R}^3$ if the probability distribution is linear ratio for each sample.

The star cover problem This problem was introduced by Karen Daniels at the 5th MSI workshop on Computational Geometry in 1995 (see [27]).

Let S be an n-vertex simple polygon. The *star-cover* problem on S is that of computing a star-shaped polygon S' such that S' contains S and the area of S' is minimized. The application can be found in material layout and manufacturing. Arkin et al. [5] reduced the *star-cover* problem to solving $O(n^2)$ problems of optimizing the sum of $O(n)$ fractional polynomials of degrees 3 trigonometric functions constrained $O(n)$ linear inequalities in 2 dimension. Later, Chen et al. [21] converted this problem into $O(n^2)$ number of the SOLR subproblems, where each the SOLR problem has an objective function with $O(n)$ ratios in $\mathbb{R}^2$.

Triangulation Kuno and Masaki [60] built a pinhole camera model for this problem and formulated it into minimizing the following form

$$\sum_{i=1}^{p} \left| \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \right|^q,$$

where $\boldsymbol{x}$ is constrained on a compact convex region in $\mathbb{R}^3$ and $q = 1$ or 2. Obviously, we are be able to transform the *triangulation* problem into the SOLR problem by adding a large enough number for each ratio if we take the L^1 norm criterion. Actually, Charnes-Cooper transformation [19] used to solve problem (1.3) can also be applied to solve this problem even the symbol of value of each ratio can be changed on its feasible region.

Layered Manufactruing In *layered manufacturing*, a physical prototype of a 3D object is built from a (virtual) CAD model by orienting and slicing the model with parallel planes and then manufacturing the slices one by one, each on top of the previous one. *Layered Manufacturing* is the basis of emerging technology called *Rapid Prototyping and Manufacturing*. Readers can read more details of *Layered Manufacturing* in [61]. The objective function $G(x)$ in problem $PR(n)$ (p. 252 in [61]) can be rewritten as the SOLR problem on an interval as follows

$$G(x) := \sum_{i=1}^{p} \frac{d_i}{1 + c_i x},$$

where x is a one-dimensional variable constrained to lie in a line segment. The authors claim that they are not aware of any efficient algorithm that solves problem $PR(n)$.

Several other problems such as optimal penetration problem [22], and so forth, also share the same character mentioned above.

1.3 Dissertation Outline

This dissertation studies on optimizing for the SOLR problem with lower dimension. We rewrite the SOLR problem into its equivalent problem with linear objective and quadratic and linear constraints. We make a linear relaxation for the quadratic constraints instead of dropping them out. Two branch and bound

algorithms have been presented to globally solve the SOLR problem with lower dimension using linear relaxation. The difference between two proposed algorithms is the branching rules— bisection branching rules and advanced branching rules.

The rest of the dissertation is organised as follows. Chapter 2 introduces preliminaries on *linear programming, Lagrangian duality* and *semidefinite programming* and refines the proof on the $\mathcal{NP}$-hardness of the SOLR problem. Chapter 3 reviews DIRECT algorithm and evaluates the performance of DIRECT algorithm for solving the SOLR problem. Chapter 4 presents two branch and bound algorithms for globally solving the SOLR problem with lower dimension using linear relaxation. Chapter 5 reformulates the SOLR problem into an SDP problem. Some existing SDP relaxations that can be applied for solving the reformulated problem. The comparisons between these relaxations are given in this chapter. Finally, we conclude this dissertation and discuss some further research in Chapter 6.

Chapter 2

Preliminaries

Since the long standing popularity of non-convex programming for optimization, there are many theoretical work in this research area. We collect the main technical tools and basic results that will be used throughout this thesis in this chapter. Readers who are familiar with these foundations are encouraged to skip ahead to section 2.5.

2.1 Notations

In this dissertation, we use $\mathbb{R}^n, \mathbb{N}$ to denote n-dimensional Euclidean space, natural numbers, respectively. The cardinality of a finite set S is denoted by $|S|$, and $\overline{co(S)}$ denotes the closure of the convex hull of S.. We let $e_i \in \mathbb{R}$ denote the i-th unit vector. The norm of a vector $v \in \mathbb{R}^n$ is denoted by $||v|| := \sqrt{v^\top v}$.

We always consider an n-dimensional *vector* $\boldsymbol{x}=(x_i)_{i=1,\ldots,n} \in \mathbb{R}^n$ to be a *column vector*, that is,

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}.$$

Additionally, we also use the more convenient notation $\boldsymbol{x}^\top = (x_1, \ldots, x_n)$ to denote the *transpose* of a column vector that is a *row vector*.

If $\boldsymbol{x}, \boldsymbol{y}$ are two vectors in $\mathbb{R}^n$, then the *inner product* of $\boldsymbol{x}, \boldsymbol{y}$ is

$$\boldsymbol{x}^\top \boldsymbol{y} := \sum_{i=1}^{n} x_i y_i.$$

We use $\mathbf{0}$, $\mathbf{1}$ to denote zero vector, $\mathbf{1}$ vector with all components equal to 1, respectively, that is

$$\mathbf{0} = \begin{pmatrix} 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix}, \mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ \vdots \\ \vdots \\ 1 \end{pmatrix}.$$

We denote $\mathbb{R}^{m \times n}$ to be the set of $m \times n$ *matrices* (m rows, n columns) with entries from $\mathbb{R}$. Also, given a matrix $A \in \mathbb{R}^{m \times n}$, A_{ij} is defined as the matrix composed of entry of A with the i-th row and j-th column. We denote by A_i and A^j the i-th row and j-th column of A respectively. The notation $\mathrm{diag}(A)$ is defined as the vector, which is the diagonal of A, while $\mathrm{Diag}(\boldsymbol{v})$ denotes the diagonal matrix with diagonal $\boldsymbol{v}$. The inner product of two matrices $A, B \in \mathbb{R}^{n \times n}$ is defined as $A \bullet B :=\mathrm{trace}(A^\top B)$. Let $\mathcal{S}^n, \mathcal{S}^n_+$ be the space of $n \times n$ symmetric matrices and $n \times n$ positive semedifinite matrices, respectively. Finally, $A \succ 0, A \succeq 0$ denote matrix A is positive definite, positive semidefinite, respectively.

2.2 Linear Programming

Linear programming (LP) has been one of the most fundamental and successful tools for optimization and discrete mathematics. Its applications include exact and approximation algorithms, as well as structural results and estimates.

2.2.1 Duality of Linear Programming

A typical LP has the following from

$$(LP) \quad \left| \begin{array}{ll} \text{minimize} & \boldsymbol{b}^\top \boldsymbol{x} \\ \text{subject to} & A\boldsymbol{x} \geq \boldsymbol{c}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{array} \right. \tag{2.1}$$

where $\boldsymbol{b}, \boldsymbol{c}$ are given vectors in $\mathbb{R}^n, \mathbb{R}^m$, respectively. A is an $m \times n$ matrix, and $\boldsymbol{x} \in \mathbb{R}^n$ is a decision variable. The set $\mathcal{F}_p = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} \geq \boldsymbol{c}, \boldsymbol{x} \geq \boldsymbol{0}\}$ is called a *feasible region*. A point $x \in \mathcal{F}_p$ is called a *feasible point*, and a feasible point x^* is called an optimal solution if $\boldsymbol{b}^\top \boldsymbol{x}^* \leq \boldsymbol{b}^\top \boldsymbol{x}$ for all feasible points $\boldsymbol{x}$. If there is a sequence $\{\boldsymbol{x}^k\}$ such that $\boldsymbol{x}^k$ is feasible and $\boldsymbol{b}^\top \boldsymbol{x}^k \to -\infty$, then the LP problem is said to be *unbounded*.

Given a LP problem we can associate a corresponding dual problem which is given as follows

$$(DP) \quad \left| \begin{array}{ll} \text{maximize} & \boldsymbol{c}^\top \boldsymbol{y} \\ \text{subject to} & A\boldsymbol{y} + \boldsymbol{s} = \boldsymbol{b}, \\ & \boldsymbol{y} \geq \boldsymbol{0}. \end{array} \right. \tag{2.2}$$

The decision variables-called the *dual variables*-form a vector $\boldsymbol{y} \in \mathbb{R}^m$. We denote the feasible region of problem (2.2) by $\mathcal{F}_d$. Linear programs are very efficiently solvable, and have a powerful duality theory showed as follows.

Theorem 2.1 (Weak duality theorem)**.** *Let $\mathcal{F}_p$ and $\mathcal{F}_d$ be non-empty. Then, $\boldsymbol{b}^\top \boldsymbol{x} \geq \boldsymbol{c}^\top \boldsymbol{y}$, where $\boldsymbol{x} \in \mathcal{F}_p$, $\boldsymbol{y} \in \mathcal{F}_d$.*

This theorem shows that a feasible solution to either problem yields a bound on the value of the other problem. We call $\boldsymbol{b}^\top \boldsymbol{x} - \boldsymbol{c}^\top \boldsymbol{y}$ a *duality gap*.

Theorem 2.2 (Strong duality theorem)**.** *Let $\mathcal{F}_p$ and $\mathcal{F}_d$ be non-empty. Then $\boldsymbol{x}^*$is optimal for (2.1) iff the following conditions hold:*
a) $\boldsymbol{x}^* \in \mathcal{F}_p$;
b) there is $\boldsymbol{y} \in \mathcal{F}_d$;
c) $\boldsymbol{b}^\top \boldsymbol{x} - \boldsymbol{c}^\top \boldsymbol{y} = 0$.

Theorem 2.3 (LP duailty)**.** *If (LP) and (DP) both have feasible solutions, then both problems have optimal solutions and the optimal objective values of the objective functions are eauql.*

If one of (LP) or (DP) has no feasible solution, then the other is either unboundsed or has no feasible solution. If one of (LP) or (DP) is unbounded then the other has no feasible solution.

The following theorem plays an important role in analysing interior-point algorithms for solving LP. It gives a unique partition of the LP variables in terms of complementarity.

Theorem 2.4 (Strict complementarity theorem)**.** *If (LP) and (DP) both have feasible solutions then both problems have a pair of strictly complementary solutions* $\boldsymbol{x}^* \geq 0$ *and* $\boldsymbol{s}^* \geq 0$ *meaning*

$$\boldsymbol{x}^*\boldsymbol{s}^* = 0 \text{ and } \boldsymbol{x}^* + \boldsymbol{s}^* > 0.$$

Moreover, the supports

$$P^* = \{j|x_j^* > 0\} \text{ and } Z^* = \{j|s_j^* > 0\}$$

are invariant for all pairs of strictly complementarity solutions.

All the important theorems' proofs mentioned above can be found in [8].

2.2.2 Algorithms for Solving Linear Programming

Various methods are available for solving LP problems, and LP problems are solvable in polynomial time.

The classical, and still very well algorithm to solve LP is the *Simplex Method* [28]. This is practically quite efficient, but can be exponential on some instances. The first polynomial time algorithm to solve linear programming problem was the *Ellipsoid Method* introduced by Khachian [50]. However, it compared poorly to the Simplex method in real life applications.

The most efficient methods known today, both theoretically and practically, are Interior Point Methods proposed by Karmarker [48]. The method starts from

an interior point of the feasible region and then follows a "central" path given by adding a logarithmic barrier function to the objective function towards an optimal point. Newton's method is applied to follow the central path. Finally, when a point sufficiently near an optimal extreme point is found, it can be rounded in a polynomial number of steps to an exact solution. The theoretical worst case running time is given by $O(Ln\ln(n))$, where n is the dimension of the problem space and L is the length of the input data, i.e., A, b, and c.

The basic idea for Karmarker Alogorithm is to use the steepest descent method. It is advisable to move in the direction of steepest descent if the current (approximate) interior point is near the centre of the polytope describing the feasible region, and it is possible to transform the feasible region so as to place the current point near the centre of the polytope, without changing the problem in any essential way.

However, a lot of LPs arising in machine learning and data mining are so large that they cannot be solved by off-the-shelf LP solvers because of memory limitations of computer. As the problem size grows, simplex-like algorithms require a prohibitive number of iterations and interior point methods need a large amount of time and memory to compute a single iteration.

In order to cope with these problems many researchers have investigated decomposition methods, such as Benders' and Dantzig-Wolfe decomposition, and Lagrange relaxations (see [62, 83]). The main idea of these methods is to divide the initial problem into subproblems that are simpler and easier to solve. Researchers also exploit the structure of special LP for dealing with even larger problems, for example, by using problem sparsity. The efficiency analysis of these methods is mostly based on empirical results for specific problem instances. And recently many researchers develop several algorithms to solve large-scale linear programming.

LP solvers are so sophisticated and robust that it is almost always keep the case that a LP can be solved as long as there is sufficient computer memory. LIPSOL , standing for Linear programming Interior-Point SOLvers, is a Matlab-based software developed by Zhang [89] for solving relatively large linear problems. CPLEX [25] optimizer, named for the simplex method as implemented in the C programming language (now it also offer other interfaces such as Matlab, Java,

C++, etc.), is a well know optimizer can solve millions of variables in linear programming [31]. The algorithms used in CPLEX is either primal or dual variants of the simplex method or the barrier interior point method. Gurobi Optimizer [66], a state-of-the-art solver for mathematical programming, is also a powerful solver for large scale programming. The solver we use for solving large-scale linear programming problems in this dissertation is Gurobi.

2.3 Lagrangian Duality

Lagrangian duality is a fundamental tool in optimization, and Lagrangian techniques have been used extensively to solve vary kinds of problems. In this section, we will review some basic definitions and results on Lagrangian duality.

2.3.1 Lagrange Function

We consider the following optimization problem to define Lagrange function.

$$\begin{aligned} \text{minimize} \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \leq 0, i \in \mathcal{I}, \\ & h_j(\boldsymbol{x}) = 0, j \in \mathcal{J}, \end{aligned} \tag{2.3}$$

where $\boldsymbol{x} \in \mathbb{R}^n$. We assume the feasible region of (2.3), $\mathcal{D}$, is nonempty, and denote its optimal value by p^*.

The basic idea in Lagrangian duality is to take some difficult constraints in (2.3) into account by augmenting the objective function with a weighted sum of the constraint function. Next we define the *Lagrange function* L associated with the problem (2.3) as

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\boldsymbol{x}) + \sum_{i \in \mathcal{I}} \lambda_i f_i(\boldsymbol{x}) + \sum_{j \in \mathcal{J}} \nu_j h_j(\boldsymbol{x}). \tag{2.4}$$

We refer to λ_i as the *Lagrange multiplier* associated with the ith inequality constraint $f_i(\boldsymbol{x}) \leq 0$; similarly we refer to ν_j as the Lagrange multiplier associated with the jth equality constraint $h_j(\boldsymbol{x}) = 0$.

2.3.2 Lagrange Dual Function

The key idea of Lagrangian duality is the following observation:

We define the *Lagrange dual function* g as the minimum value of the Lagrange function over $\boldsymbol{x}$ for $\boldsymbol{\lambda} \in \mathbb{R}^{|\mathcal{I}|}, \boldsymbol{\nu} \in \mathbb{R}^{|\mathcal{J}|}$,

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} \left(f_0(\boldsymbol{x}) + \sum_{i \in \mathcal{I}} \lambda_i f_i(\boldsymbol{x}) + \sum_{j \in \mathcal{J}} \nu_j h_j(\boldsymbol{x}) \right). \tag{2.5}$$

It is easy to verify that the dual lagrange function takes on the value $-\infty$ when the lagrange functionis unbounded below in $\boldsymbol{x}$. Since the dual function is the pointwise infimum of a family of affine functions of $(\boldsymbol{\lambda}, \boldsymbol{\nu})$, it is concave, even if the problem (2.3) is not convex.

Proposition 2.5. *For any* $\boldsymbol{\lambda} \geq 0, \boldsymbol{\nu}, g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \leq p^*$.

That is to say, for any $\boldsymbol{\nu}$ and nonnegative $\boldsymbol{\lambda}$, $g(\boldsymbol{\lambda}, \boldsymbol{\nu})$ is a lower-bound of the true optimum. First, for any $\boldsymbol{x}$ in feasible region,

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq f_0(\boldsymbol{x}). \tag{2.6}$$

Inequality (2.6) indicates that the Lagrange function can only work for legal solutions. Thus, in particular, if $\boldsymbol{x}^*$ is the optimum of problem (2.3),

$$g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \leq \inf_{\boldsymbol{x} \in \mathcal{D}} L(\boldsymbol{x}^*, \boldsymbol{\lambda}, \boldsymbol{\nu}) = p^*. \tag{2.7}$$

For illegal solutions, however, this is not true. If $\boldsymbol{x}'$ violates some constraints, then for certain settings of the Lagrange multipliers, $L(\boldsymbol{x}', \boldsymbol{\lambda}, \boldsymbol{\nu}) > f_0(\boldsymbol{x}')$.

2.3.3 Lagrange Dual Problem

We know that the Lagrange dual function gives valid lower bounds for any $\boldsymbol{\nu}, \boldsymbol{\lambda} \geq 0$, it is natural to try to get the best lower bound. A better lower bound can be exploited the following *Lagrange dual problem* associated with problem (2.3)

$$\begin{aligned} \text{maximize} \quad & g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geq 0. \end{aligned} \tag{2.8}$$

The Lagrange dual problem (2.8) is a convex optimization problem, since the objective to be maximized is concave and the constraints are convex.

2.3.4 Weak Duality and Strong Duality

Let d^* be the optimal value of the Lagrange dual problem. In particular, we have the simple but important inequality

$$d^* \leq p^*, \tag{2.9}$$

which holds even if the original problem is not convex. This property is called *weak duality*.

We refer to the difference $p^* - d^*$ as the *optimal duality gap* of problem (2.3). If the equality

$$p^* = d^* \tag{2.10}$$

holds, i.e., the optimal duality gap is zero, then *strong duality* holds, which means that the best bound obtained from the Lagrange dual function is tight. Constraint qualications for $f_i(\boldsymbol{x}_0) < 0$, $i = 1, \ldots, m$ for some $\boldsymbol{x}_0 \in D$. A common usage of the Lagrangian duality is that, when Slater's conditions are satisfied, we can transform the primal problem into the dual problem and apply optimization methods to the dual problem instead of solving the primal problem directly.

2.4 Semidefinite Programming

Semidefinite programming (SDP) , can be solved very efficiently in practice as well as in theory, is the most exciting development in mathematical programming in the 1990's and plays a very useful role in non-convex or combinatorial optimization. And the semidefinite relaxation (SDR) is a powerful, computationally efficient approximation technique for a host of very difficult optimization problems.

In this section, we give some results on semidefinite programming. Particularly, we give some overview of semidefinite relaxation for quadratic optimization problem. Readers are referred to Nemirovski [8] and Helmberg [41] for further background and more details.

A semidefinite programming is an optimization problem of the stand form:

$$\text{(SDP-P)} \quad \left| \begin{array}{ll} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X = b_i, i = 1, \ldots, m, \\ & X \succeq 0, \end{array} \right. \tag{2.11}$$

where $C, A_i \in \mathcal{S}^n, X$ is an $n \times n$ matrix.

2.4.1 Semidefinite Duality

A very important feature of semidefinite programming, from both the theoretical and applied viewpoints, is the associated *duality theory*. For each SDP program in the form (2.11), there is another associated SDP, called the *dual problem* which can be stated as follows

$$\text{(SDP-D)} \quad \left| \begin{array}{ll} \text{maximize} & \boldsymbol{b}^\top \boldsymbol{y} \\ \text{subject to} & \sum_{i=1}^{m} A_i y_i \preceq C, \end{array} \right. \tag{2.12}$$

where $\boldsymbol{b} = (b_1, \ldots, b_m)^\top$ is given, and $\boldsymbol{y} = (y_1, \ldots, y_m)^\top$ is the dual decision variables.

As in the linear programming case, a key relationship between the primal and the dual problems is that feasible solutions of one problem can be used to bound the values of the other, which can be showed by the following proposition.

Proposition 2.6 (SDP weak dulity)**.** *Let X and y be feasible for (SDP-P) and (SDP-D), respectively. Then we have $C \bullet X - \boldsymbol{b}^\top \boldsymbol{y} \geq 0$.*

Proof. Since X and y are feasible for (SDP-P) and (SDP-D), we have

$$C \bullet X - b^\top y \geq \left(\sum_{i=1}^{m} A_i y_i \right) \bullet X - \boldsymbol{b}^\top \boldsymbol{y} = 0.$$

as desired. □

2.4.2 Semidefinite Relaxation for Quadratic Optimization

Let us write the quadratic optimization as follows:

$$\text{(QP)} \quad \left| \begin{array}{ll} \text{minimize} & \boldsymbol{x}^\top C \boldsymbol{x} \\ \text{subject to} & \boldsymbol{x}^\top A_i \boldsymbol{x} \geq b_i, i = 1, \ldots, m, \end{array} \right. \tag{2.13}$$

where $C, A_1, \ldots, A_m \in \mathcal{S}^n$, $b_1, \ldots, b_m \in \mathbb{R}$. To make a relaxation of (2.13), it is easy to observe that

$$\boldsymbol{x}^\top A_i \boldsymbol{x} = \text{trace}(\boldsymbol{x}^\top A_i \boldsymbol{x}) = \text{trace}(A_i \boldsymbol{x}^\top \boldsymbol{x}) = A_i \bullet X,$$

$$\boldsymbol{x}^\top C \boldsymbol{x} = \text{trace}(\boldsymbol{x}^\top C \boldsymbol{x}) = \text{trace}(C \boldsymbol{x}^\top \boldsymbol{x}) = C \bullet X,$$

where $X = \boldsymbol{x}^\top \boldsymbol{x}$. Particularly, both the objective function and constraints in (2.13) are linear with respective to matrix X. Furthermore, $X = \boldsymbol{x}^\top \boldsymbol{x}$ is equivalent to X being a rank one symmetric positive semidefinite matrix. We obtain the following equivalent problem of problem (2.13)

$$\begin{array}{rl} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X \geq b_i, i = 1, \ldots, m, \\ & X \succeq \mathbf{0}, \text{rank}(X) = 1. \end{array} \tag{2.14}$$

Although problem (2.14) is as difficult to solve as problem (2.13), the formulation in (2.14) allows us to identify the fundamentally difficulty in solving problem (2.13). Indeed, the constraints in problem (2.14) are all convex constraints except the constraint $\text{rank}(X) = 1$. Thus, we can obtain a relaxed version which is given as bellow for problem (2.14) by dropping the non-convex constraint as follows

$$\begin{array}{rl} \text{minimize} & C \bullet X \\ \text{subject to} & A_i \bullet X = b_i, i = 1, \ldots, m, \\ & X \succeq 0. \end{array} \tag{2.15}$$

Indeed, problem (2.15), the standard form of semidefinite programming, can be solved conveniently and effectively by many optimization tools such as CVX [39] or Sedumi [69].

Recently, several SDP relaxations have been proposed for problem (2.13) [3, 4, 11, 12, 14, 35]. Particularly, we rewrite the following main theoretical results for different SDP relaxations for (2.13), which is reported by Bao [6].

$$v^L = v^{Shor} = v^{DShor} \leq v^{SD} \leq v^{SC} \leq v^{SRLT} \leq f^*,$$

where

- v^L : the value of a standard Lagrangian relaxation [85] of (2.13),
- v^{Shor} : the value of the Shor relaxation [78],
- v^{DShor} : the value of dual of the Shor relaxation [78],
- v^{SD} : the value of the Shor relaxation enhanced with convex/concave envelopes for diagonal elements of the matrix variables [78],
- v^{SC} : the value of the Shor relaxation [78] enhanced with convex/concave envelopes for matrix variables,
- v^{SRLT} : the value of the Shor relaxation enhanced with a partial first-order RLT [76],
- f^*: the optimal value of (2.13).

2.5 $\mathcal{NP}$-hardness of the SOLR Problem

In order to propose effective solution approaches for an optimization problem, it is essential to understand the problem's complexity [38]. This chapter reviews the known complexity results [63, 67] for some linear multiplicative programming and proves that the SOLR problem with linear constrains can be reduced to a kind of linearly multiplicative problem which is proved to be an $\mathcal{NP}$-hard problem [63]. These results enhance our understanding of what makes the SOLR problem so difficult to solve.

2.5.1 Prelininaries

Let us define some classes of problems as follows:

Definition 2.7. Any problem for which the answer is either zero or one is called a *decision problem*. An algorithm for a decision problem is termed a *decision algorithm*.

Definition 2.8. Any problem that involves the identification of an optimal (either minimum or maximum) value of a given cost function is known as an *optimization problem*. An *optimization algorithm* is used to solve an optimization problem.

Definition 2.9. $\mathcal{P}$ is a set of all decision problems solvable by deterministic algorithms in polynomial time. $\mathcal{NP}$ is a set of all decision problems solvable by nondeterministic algorithms in polynomial time.

Clearly, if a problem is in $\mathcal{P}$, then it is also in $\mathcal{NP}$.

Definition 2.10. A problem $\mathcal{A}$ is $\mathcal{NP}$-hard if and only if satisfiability reduces to $\mathcal{A}$ (satisfiability $\propto \mathcal{A}$). A problem $\mathcal{A}$ is $\mathcal{NP}$-complete if and only if $\mathcal{A}$ is $\mathcal{NP}$-hard and $\mathcal{A} \in \mathcal{NP}$.

Definition 2.11. Let $\mathcal{A}$ and $\mathcal{B}$ be problems. Problem $\mathcal{A}$ reduces to $\mathcal{B}$ ($\mathcal{A} \propto \mathcal{B}$) if and only if there is a way to solve $\mathcal{A}$ by a deterministic polynomial time algorithm using a deterministic algorithm that solves $\mathcal{B}$ in polynomial time.

The reader is referred to the classic text by Garey and Johnson [49] for more details on computational complexity and the theories of $\mathcal{NP}$-hard and $\mathcal{NP}$-complete.

In order to prove the $\mathcal{NP}$-hardness of the SOLR problem, let us consider the following problem which is given in [63] :

$$\begin{aligned} &\text{maxmize} && \tfrac{1}{x_1} + \tfrac{1}{x_2} \\ &\text{subject to} && \boldsymbol{x} \in X = \{\boldsymbol{x} \in \mathbb{R}^n | A\boldsymbol{x} \leq c, \boldsymbol{x} \geq \boldsymbol{0}\} \end{aligned} \tag{2.16}$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$. Clearly, the SOLR problem is $\mathcal{NP}$-hard if (2.16) $\propto$ the SOLR problem under the condition that (2.16) is $\mathcal{NP}$-hard. Next we will make a construction to show that (2.16) is $\mathcal{NP}$-hard.

2.5.2 Main Construction

Pardalos and Vavasis [67] showed that the following problem of minimizing a quadratic concave problem with linear constraints is $\mathcal{NP}$-hard.

$$\begin{aligned} &\text{maxmize} && x_1 - x_2^2 \\ &\text{subject to} && \boldsymbol{x} \in X = \{\boldsymbol{x} | A\boldsymbol{x} \leq c, \boldsymbol{x} \geq \boldsymbol{0}\}. \end{aligned} \tag{2.17}$$

In this section, we review an excellent proof given by Matsui [63] who refined the proof described in [67]. Let us denote the value $rx_1+r^2x_2+\cdots+r^nx_n$ be w, where $\boldsymbol{x} \in [0,1]^n$ and $r > 0$. And for $\forall \boldsymbol{x} \in [0,1]^n$,

$$w^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} x_i x_j.$$

Since $\forall \boldsymbol{x} \in [0,1]^n$, we can make a relaxation for $x_i x_j$ by the following linear inequalities if we replace $x_i x_j$ by y_{ij}.

$$Y := \left\{ y_{ij} \;\middle|\; \begin{array}{l} 0 \le y_{ij} \le 1, \\ y_{ij} \le x_i, \\ y_{ij} \le x_j, \\ y_{ij} \ge x_i + x_j - 1, \\ i \ne j. \end{array} \right\} \tag{2.18}$$

For the case $i = j$, we replace the term $x_i x_i$ by y_{ii}. Thus,

$$y_{ii} = x_i. \tag{2.19}$$

In addition, w^2 is approximated by the following value with the definition of y_{ij}.

$$w^2 = \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} x_i x_j.$$

Clearly, according to (2.18), $y_{ij} = x_i x_j$ if either x_i or x_j is 0-1 valued. And $y_{ii} = x_i$ if and only if $x_i \in [0,1]^n$ is 0-1 valued for all i. Therefore, for $x_i \in [0,1]$ valued 0-1 for all i, the equality $y_{ij} = x_i x_j$ always holds. However, if $\boldsymbol{x} \in [0,1]^n$ is not valued 0-1, the equality $y_{ij} = x_i x_j$ does not hold in general. Next we consider the gap between w^2 and $\sum_{i=1}^{n} r^{i+j} y_{ij}$ when $\boldsymbol{x}$ is not 0-1 valued.

Let $k = \arg\max(x_i), 0 < x_i < 1$. Therefore, for any $i > k$, $y_{ij} = x_i x_j$ for all j because x_i is 0-1 valued. Then we have

$$\begin{aligned}
& \sum_{i=1}^{n} r^{i+j} y_{ij} - w^2 \\
= & \sum_{i=1}^{k}\sum_{j=1}^{k} r^{i+j} y_{ij} + \sum_{i=k+1}^{n}\sum_{j=1}^{k} r^{i+j} y_{ij} \\
& + \sum_{i=1}^{k}\sum_{j=k+1}^{n} r^{i+j} y_{ij} + \sum_{i=k+1}^{n}\sum_{j=k+1}^{n} r^{i+j} y_{ij} - \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} x_i x_j \\
= & \sum_{i=1}^{k}\sum_{j=1}^{k} r^{i+j} y_{ij} + \sum_{i=k+1}^{n}\sum_{j=1}^{k} r^{i+j} x_i x_j \\
& + \sum_{i=1}^{k}\sum_{j=k+1}^{n} r^{i+j} x_i x_j + \sum_{i=k+1}^{n}\sum_{j=k+1}^{n} r^{i+j} x_i x_j - \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} x_i x_j \\
= & \sum_{i=1}^{k}\sum_{j=1}^{k} r^{i+j} y_{ij} - \sum_{i=1}^{k}\sum_{j=1}^{k} r^{i+j} x_i x_j \\
\geq & \; r^{2k} y_{kk} - (r^{2k}(x_k)^2 + r^{2k-1}(k^2 - 1)) \\
= & \; r^{2k}(x_k - (x_k)^2) - r^{2k-1}(k^2 - 1) \\
> & \; r^2(x_k - (x_k)^2) - rn^2.
\end{aligned}$$

Since $0 < x_k < 1$, $x_k - (x_k)^2 \geq \frac{1}{4}$. Let $\sigma \in (0, \frac{1}{2})$, and we have

$$\sum_{i=1}^{n} r^{i+j} y_{ij} - w^2 \geq r^2\sigma/2 - rn^2. \tag{2.20}$$

Lemma 2.12. *Let $\boldsymbol{x} \in \mathbb{R}^n$, and $\boldsymbol{y} \in \mathbb{R}^{n\times n}$ such that:*

$$\begin{aligned}
& 0 \leq x_i \leq 1 && (for\,all\,i), \\
& 0 \leq y_{ij} \leq 1 && (for\,all\,i,\,j), \\
& y_{ij} \leq x_i,\; y_{ij} \leq x_j,\; y_{ij} \geq x_i + x_j - 1 && (for\,all\,i,\,j\,such\,that\,i \neq j), \\
& y_{ii} = x_i && (for\,all\,i).
\end{aligned} \tag{2.21}$$

Assume that r is a positive integer, $\boldsymbol{x}$ is not 0-1 valued and $\exists\sigma \in (0, 1/2)$ such that each element x_i is either $x_i = 0, 1$ or $\sigma < x_i < 1 - \sigma$. Then the inequality $\sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y_{ij} - w^2 \geq r^2\sigma/2 - rn^2$ holds.

Lemma 2.12 implies that $\sum_{i=1}^{n} r^{i+j} y_{ij} - w^2 > 0$ if r is sufficiently large. Thus,

$\sum_{i=1}^{n} r^{i+j} y_{ij} - w^2 \leq 0$ if and only if $\boldsymbol{x} \in [0,1]^n$ is 0-1 valued if r is sufficiently large, which give us an idea to show the $\mathcal{NP}$-hardness of problem (2.17). We transform the $\mathcal{NP}$-hard problem, *Set partition* [49, 64], to a decision version of problem (2.17) to show $\mathcal{NP}$-hardness of (2.17).

Set partition

- DATA: A 0-1 matrix $M \in \mathbb{R}^{m \times n}$ such that $n > m$.
- QUESTION: Is there a 0-1 vector $\boldsymbol{x}$ such that $M\boldsymbol{x} = \mathbf{1}$?

Let us consider the following optimization problem:

$$\begin{aligned} f^* = \text{minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y_{ij} - \left(\sum_{i=1}^{n} r^i x_i\right)^2 \\ \text{subjcet to} \quad & (2.21)\, \textit{and}\, M\boldsymbol{x} = \mathbf{1}. \end{aligned} \tag{2.22}$$

Suppose $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution. Without loss of generality, let $\mathcal{I}$ is set of the index such that $x_i = 0, \forall i \in \mathcal{I}$. And $\mathcal{J}$ is the set of index that $x_j = 0, \forall j \in \mathcal{J}$. Since y_{ij} is constrained to the set linear inequalities of (2.21), $y_{ij} = 0$ if and only if $x_i = 0$. Similarly, $y_{ij} = 1$ if and only if $\mathcal{I} = \emptyset$. Therefore, the optimal value of (2.22) is non-positive if the system $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution.

Now we focus on the case that the system $M\boldsymbol{x} = \mathbf{1}$ dose not have a 0-1 valued solution. Let $\mathcal{F}_{P_3}$ be the feasible region of problem (2.22), and, of course, $\mathcal{F}_{P_3}$ is a bounded polytope. The number of constrains of (2.22) is equal to $n + n^2 + 4(n^2 - n) + n + m$ and so the number of constrains is less than n^3 when $n > 5$. Let $(\boldsymbol{x}', \boldsymbol{y}')$ be a vertex of polytope $\mathcal{F}_{P_3}$. Since the element of the constraint matrix of (2.22) is $-1, 0$ or 1, Cramer's rule imply that each element of $(\boldsymbol{x}', \boldsymbol{y}')$ is 0-1 or contained in the interval $[1/(n^3)^{n^3}, 1 - 1/(n^3)^{n^3}]$, which can derive the following theorem if we let $\sigma' = 1/(n^3)^{n^3}$.

Theorem 2.13. *Let $M \in \mathbb{R}^{m \times n}$ be a 0-1 matrix with $n > m$ and $n \geq 5$. Assume that $r = n^{n^4}$. The equality system $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution if and only if the optimal value of problem (2.22) is non-positive. If $M\boldsymbol{x} = \mathbf{1}$ does not have a 0-1 valued solution, the optimal value of problem (2.22) is greater than r.*

Proof. We have mentioned that the optimal value of problem (2.22) is non-positive if $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution.

Now we consider the case that $M\boldsymbol{x} = \mathbf{1}$ does not have a 0-1 valued solution. For any vertex $(\boldsymbol{x}', \boldsymbol{y}')$ of polytope $\mathcal{F}_{P_3}$, each element of $(\boldsymbol{x}', \boldsymbol{y}')$ is 0-1 valued or contained in the interval $[1/(n^3)^{n^3}, 1 - 1/(n^3)^{n^3}]$. Since $M\boldsymbol{x} = \mathbf{1}$does not have a 0-1 valued solution, any element of x' is not 0-1 valued. According to lemma 2.12, we have

$$\sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y'_{ij} - \left(\sum_{i=1}^{n} r^i x'_i\right)^2 \geq r^2\sigma'/2 - rn^2 > p(n^{n^4}/(2n^{3n^3}) - n^2) > r.$$

Because $(\boldsymbol{x}', \boldsymbol{y}')$ is the any vertex of polytope $\mathcal{F}_{P_3}$, any feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ can be represented by a convex combination of vertices of $\mathcal{F}_{P_3}$. Since the objective function of (2.22) is concave, any feasible solution $(\boldsymbol{x}, \boldsymbol{y})$ satisfied that $\sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y_{ij} - \left(\sum_{i=1}^{n} r^i x_i\right)^2 > p$, which completes the proof. □

Theorem 2.23 implies that we can decide the answer to the *Set partition* by solving the optimization problem (2.22). The largest coefficient of (2.22) is r^{2n}, and its input size is $\lceil \log(r^{2n}) \rceil + 1 = \lceil \log(n^{n^4})^{2n}) \rceil + 1 = \lceil 2n^5 \log n \rceil + 1$, which indicates that the input size of problem (2.22) is bounded by a polynomial of n. Therefor, problem (2.22) is $\mathcal{NP}$-hard.

In addition, we can extend the results of theorem 2.23 to a general optimization.

Corollary 2.14. *Let n be a positive integer such that $n \geq 5$ and $r = n^{n^4}$. Assume that $g(x_0, y_0)$ is a function satisfying the following condition:*

1. $\forall x_0 \in [0, nr^n]$, $\forall y_0 \in [0, n^2 r^{2n}]$, *if* $y_0 - x_0^2 \leq 0$ *then* $g(x_0, y_0) \leq 0$,
2. $\forall x_0 \in [0, nr^n]$, $\forall y_0 \in [0, n^2 r^{2n}]$, *if* $y_0 - x_0^2 > r$ *then* $g(x_0, y_0) > 0$,

Let $M \in \mathbb{R}^{m\times n}$ be a 0-1 matrix with $n > m$ and $n \geq 5$, and we define the following problem:

$$\begin{aligned} f^* = \text{minimize} \quad & g(x_0, y_0) \\ \text{subject to} \quad & (2.21) \text{ and } M\boldsymbol{x} = \mathbf{1}, \\ & x_0 = \sum_{i=1}^{n} r^i x_i, \\ & y_0 = \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y_{ij}. \end{aligned} \tag{2.23}$$

Then the optimal value of problem (2.23) is non-positive if and only if the equality the system $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution.

Corollary 2.14 give us a way to construct optimization problems which can be reduced to problem (2.23). Let $n \geq 5$ and $r = n^{n^4}$, considering the following special function :

$$\begin{aligned} g_1(x_0, y_0) &= (y_0 - r + 2r^{4n})^2 - 4r^{4n}x_0^2 - 4r^{8n} \\ &= (y_0 - r + 2r^{4n} + 2r^{2n}x_0)(y_0 - r + 2r^{4n} - 2r^{2n}x_0) - 4r^{8n}. \end{aligned}$$

Next we show that $g_1(x_0, y_0)$ satisfies the two conditions in Corollary 2.14.

1. If $x_0 \in [0, nr^n]$, $y_0 \in [0, n^2r^{2n}]$ and $y_0 - x_0^2 \leq 0$, then

$$\begin{aligned} g_1(x_0, y_0) &= (y_0 - r + 2r^{4n})^2 - 4r^{4n}x_0^2 - 4r^{8n} \\ &\leq (y_0 - r)^2 + 2(y_0 - r)(2r^{4n}) + 4r^{8n} - 4r^{4n}y_0 - 4r^{8n} \\ &= (y_0 - r)^2 - 4r^{4n}(y_0 + r - y_0) \\ &= (y_0 - r)^2 - 4r^{4n+1} \\ &\leq (y_0)^2 + r^2 - 4r^{4n+1} \\ &\leq (x_0)^4 + r^2 - 4r^{4n+1} \\ &\leq n^4r^{4n} + r^2 - 4r^{4n} \\ &\leq 0 \,(\text{since } p \text{ is sufficient large}). \end{aligned}$$

2. If $x_0 \in [0, nr^n]$, $y_0 \in [0, n^2 r^{2n}]$ and $y_0 - x_0^2 > p$, then

$$\begin{aligned} g_1(x_0, y_0) &= (y_0 - r + 2r^{4n})^2 - 4r^{4n}x_0^2 - 4r^{8n} \\ &> (x_0^2 + 2r^{4n})^2 - 4r^{4n}x_0^2 - 4r^{8n} \\ &= x_0^2 \\ &\geq 0. \end{aligned}$$

Then it is natural consider the optimization problems as follows

$$\begin{aligned} f_1^* = \text{minimize} \quad & z_1 z_2 \\ \text{subject to} \quad & (2.21)\, and\, M\boldsymbol{x} = \mathbf{1}, \\ & x_0 = \sum_{i=1}^{n} r^i x_i, \\ & y_0 = \sum_{i=1}^{n}\sum_{j=1}^{n} r^{i+j} y_{ij}, \\ & z_1 = y_0 - r + 2r^{4n} + 2r^{2n}x_0, \\ & z_2 = y_0 - r + 2r^{4n} - 2r^{2n}x_0. \end{aligned} \tag{2.24}$$

Clearly, the optimal value of problem (2.24) is less than or equal to $4r^{8n}$ if and only if $M\boldsymbol{x} = \mathbf{1}$ has a 0-1 valued solution. Thus, we can decide the answer of *set partition* by solving problem (2.24). In addition, the largest coefficient in problem (2.24) is $2r^{4n} = 2(n^{n^4})^{4n} = 2n^{4n^5}$, and the threshold value is $4r^{8n} = 4(n^{n^4})^{8n^5} = 4n^{8n^5}$. Thus, the input size of problem the largest coefficient and the threshold value are $4n^5 \log(2n)$, $8n^5 \log(4n)$, respectively. Of course, they are both bounded by a polynomial of n. Further more, problem (2.24) is a special case of problem (2.22). Therefore, problem (2.22) can be reduced into problem (2.24), which derives the following lemma.

Lemma 2.15. *Problem (2.24) is $\mathcal{NP}$-hard.*

Note that z_1, z_2, defined in problem (2.24), are both positive. Since $r = n^{n^4}$ is an efficiently large number when $n \geq 5$. Therefore

$$z_1 = y_0 - r + 2r^{4n} + 2r^{2n}x_0 > -r + 2r^{4n} > 0;$$

$$z_2 = y_0 - r + 2r^{4n} - 2r^{2n}x_0 > -r + 2r^{4n} - 2r^{2n}nr^n = -r + 2r^{4n} - 2nr^{3n} > 0.$$

Since problem (2.24) implies that $f_1^* \leq (2r^{4n})^2$ if and only if $M\boldsymbol{x} = \mathbf{1}$ has a 0—1 valued solution. In addition, $z_1 z_2 \leq (2r^{4n})^2$ if and only if

$$\frac{1}{z_1 + 2r^{4n}} + \frac{1}{z_2 + 2r^{4n}} \geq \frac{1}{2r^{4n}}.$$

Therefore, if we let $z_3 = z_1 + 2r^{4n}$ and $z_4 = z_2 + 2r^{4n}$, $1/z_3 + 1/z_4 \geq 1/2r^{4n}$. Thus, we can decide the answer to *set partition* by optimizing the following problem:

$$\begin{aligned}
\text{minimize} \quad & \tfrac{1}{z_3} + \tfrac{1}{z_4} \\
\text{subject to} \quad & (2.21)\, \textit{and}\, M\boldsymbol{x} = \mathbf{1}, \\
& x_0 = \sum_{i=1}^{n} r^i x_i, \\
& y_0 = \sum_{i=1}^{n} \sum_{j=1}^{n} r^{i+j} y_{ij}, \\
& z_1 = y_0 - r + 2r^{4n} + 2r^{2n} x_0, \\
& z_2 = y_0 - r + 2r^{4n} - 2r^{2n} x_0, \\
& z_3 = z_1 + 2r^{4n}, \\
& z_4 = z_2 + 2r^{4n}.
\end{aligned} \tag{2.25}$$

Similarly, the input size of the largest coefficient and its threshold value of problem (2.25) are both bounded by a polynomial of n, and problem (2.24) is a special case of problem (2.25), which implies that the $\mathcal{NP}$-hard problem (2.24) can be reduced to problem (2.25). Then we have following thorem:

Theorem 2.16. *Problem* (2.16) *is* $\mathcal{NP}$*-hard.*

2.5.3 An Example-Traveling Salesman Problem

In section 2.5.2, we showed that the *set partition* problem can be reduced. It is well know that the *traveling salesman problem* (TSP) is an $\mathcal{NP}$-hard problem. Gao, Mishra and Shi [37] showed that TSP can be written in a form of the SOLR problem. In this section, we will review it to show how to reduce the TSP problem to the SOLR problem.

The TSP problem consists of a salesman and a set l cities. The salesman has to visit each one of the n cities once and only once, starting from a certain one and returning to the same city. What route, or tour, should be choose in order to

minimise the total distance traveled? Other notions such as time, cost, etc., can be considered as well instead of distance.

Mathematically, the TSP problem can be described as follows:

(1) Given a "cost matrix" $D = (d_{ij})$, where d_{ij} is the cost of going from city i to city j, $(i, j, \ldots, n)$. determine x_{ij} which minimises the quantity $Q = \sum_{i,j} d_{ij}x_{ij}$ subject to

(a) $x_{ii} = 0$;

(b) $x_{ij} = 0, 1$;

(c) $\sum_i x_{ij} = \sum_j x_{ij}$;

(d) for any subset $S = \{i_1, i_2, \ldots, i_r\}$ of the integers from 1 to n,

$$x_{i_1 i_2} + x_{i_2 i_3} + \cdots + x_{i_{r-1} i_r} + x_{i_r i_1} \begin{cases} < r & for\, r < n \\ \leq n & for\, r = n \end{cases}$$

Let O, O^* be the city order index set and the optimal order from city 1 to city n is $O^* = \{o_1^*, o_2^*, \ldots, o_n^*, o_1^*, \}$, respectively. Then solving the TSP problem is equivalent to optimise the following problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{n} d_{o_i o_{i+1}} x_{o_i o_{i+1}} + d_{o_n o_1} x_{o_n o_1} \\ \text{subject to} \quad & (a)\, to\, (d). \end{aligned} \tag{2.26}$$

We denote that

$$\alpha_j := \sum_{i=1}^{n} d_{o_i^j o_{i+1}^j} x_{o_i^j o_{i+1}^j} + d_{o_n^j o_1^j} x_{o_n^j o_1^j},$$

and $L := \min\{d_{i_1,i_2} | i_1, i_2, \in I\}, \beta_j := \frac{L}{2n!\alpha_j}$ for each order O^j with $1 \leq j \leq (n-1)!/2$, where I is the city index set. Therefore, problem (2.26) is equivalent to finding and order O^j such that β_j is maximized for $1 \leq j \leq (n-1)!/2$. Denote that

$$\rho_j := \beta_{j+1}(j+1) - (j+1)^2,\ \sigma_j := (\beta_j - j)j,$$

and

$$g_j(y) := y + \frac{\rho_j(y-j) + \sigma((j+1)-y)}{y},\ y \in \mathbb{R}.$$

Clearly, $g_j(j) = \beta_j$, $g_j(j+1) = \beta_{j+1}$. Furthermore, $g_j(y)$ is convex becasuse $\sigma_j(j+1) - \rho_j j > 0$, for all j. Since $\rho_j(j-j) + \sigma_j((j+1)-j) = \sigma_j = (\beta_j - j)j$, and $\rho_j((j+1)-j) + \sigma_j((j+1)-(j+1)) = \rho_j = (j+1)(\beta_{j+1} - (j+1))$, and $\rho_{j+1} - \sigma_{j+1} < \rho_j - \sigma_j$, which implies that the function values of the numerator of the second term of $g_j(\cdot)$ at point j and point $j+1$ are the same, and the slope of the numerator is decreasing with j growing. According to these properties above, we can solve the following problem instead of problem (2.26).

$$\begin{aligned} \text{maximize} \quad & y + \tfrac{t}{y} \\ \text{subject to} \quad & t \le \rho_j(y-j) + \sigma_j((j+1)-y), j = 1, \ldots, (n-1)!/2, \\ & 1 \le y \le (n-1)!/2. \end{aligned} \tag{2.27}$$

Therefore, the TSP problem can be reduced to a problem (2.27) which is a special case of sum of linear ratios, which indicates that problem (2.27) is $\mathcal{NP}$-hard problem.

Chapter 3

The DIRECT Algorithm for Solving the SOLR Problem

The SOLR problem, mentioned in Chapter 1, is a non-convex and $\mathcal{NP}$-hard problem. Furthermore, the derivative information of objective function is difficult to get and therefore only objective function values can be used. There are some methods used to solve this kind of problem, and one of them is so called the *DIRECT algorithm* [36, 47] which is the abbreviation of Dividing RECTangles. DIRECT does not exploit any a prior knowledge on the objective function, which is suitable for our problems. In this chapter, we apply the DIRECT algorithm for solving SOLR with lower dimension within a given tolerance.

3.1 Lipschitz Optimization

For easier understand the princle of the DIRECT algorithm, we give the following *Lipschitz continuous functions*.

Definition 3.1. Let $M \subset \mathbb{R}^n$ and $f : M \to \mathbb{R}$. The function f is called Lipschitz continuous on M with Lipschitz constant γ if

$$|f(x) - f(x')| \leq \gamma \|x - x'\| \quad \forall x, x' \in M, \tag{3.1}$$

where $\|\cdot\|$ is any norm on $\mathbb{R}^n$. The following theorem shows that the class of Lipschitz continuous functions is very general.

Theorem 3.2. *Let $M \subset \mathbb{R}^n$ be a bounded , closed set , and let f be a continuously differentiable function on an open convex set $M_0 \supseteq M$. Then f is Lipschitz continuous on M.*

Proof. Let $\boldsymbol{x}^0 \in M$ be an arbitrary point. Then the first order Taylor-expansion of f near $\boldsymbol{x}^0$ is

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^0) + g(\boldsymbol{x}^r)^\top(\boldsymbol{x} - \boldsymbol{x}^0), \boldsymbol{x} \in M, \tag{3.2}$$

where g is the gradient of f, and $\boldsymbol{x}^r$ is a suitably chosen point of the interval $[\boldsymbol{x}^0, \boldsymbol{x}]$. Applying the Cauchy-inequality to equation (3.2), we have

$$\left|f(\boldsymbol{x}) - f(\boldsymbol{x}^0)\right| = \left|g(\boldsymbol{x}^r)^\top(\boldsymbol{x} - \boldsymbol{x}^0)\right|, \boldsymbol{x} \in M. \tag{3.3}$$

Then the relationship $\overline{co(M)} = co(M) \subseteq M_0$ holds [68], and therefore $\overline{co(M)}$ is compact and the norm of the gradient g is bounded on $\overline{co(M)}$. Then $\max_{\boldsymbol{x}\in\overline{co(M)}} g(\boldsymbol{x})$ is an appropriate Lipschitz constant, and f is Lipschitz continuous. □

According to the Definition 3.1 and the Theorem 3.2, we have the following theorem which gives the Lipschitz continuousity of the objective function in (1.3).

Theorem 3.3. *The function $f(\boldsymbol{x})$ in (1.3) is Lipschitz continuous on X.*

Proof. We show $f(\boldsymbol{x})$ in (1.3) is continuous differentiable on open convex set $X_0 \supseteq X$ first. Let $r_i(\boldsymbol{x}) = \boldsymbol{n}_i^\top + a_i$, $q_i(\boldsymbol{x}) = \boldsymbol{d}_i^\top + b_i$, $h(\boldsymbol{x}) = \frac{r_i(\boldsymbol{x})}{q(\boldsymbol{x})}$. Since $q_i(\boldsymbol{x}) \neq 0$ on X for all $i = 1, \ldots, p$, we have the following formular for each $j \in 1, \ldots, n$

$$\frac{\partial h(\boldsymbol{x})}{\partial \boldsymbol{x}_j} = \frac{\frac{\partial r(\boldsymbol{x})}{\partial \boldsymbol{x}_j} q(\boldsymbol{x}) - r(\boldsymbol{x}) \frac{\partial q(\boldsymbol{x})}{\partial \boldsymbol{x}_j}}{q(\boldsymbol{x})^2}$$

Clearly, X is a bounded set, therefore $f(\boldsymbol{x})$ in (1.3) is Lipschitz continuous on X according to the Theorem 3.3, □

3.2 Outline of the DIRECT Algorithm

In this section, we give a brief overview of the DIRECT algorithm proposed by Jones at al. [47]. The DIRECT algorithm, created in order to solve difficult global optimization problems with bound constraints and Lipschitz-continuous objective

function, is a sampling algorithm and has been proven to be effective in a wide range of application domains. The algorithm centers around a space-partitioning scheme that divides large hyperrectangles into smaller ones. And the center of each hyperrectangle, considered as a representative of the hyperrectangle, is evaluated by objective function to compare with other ones.

The key of the algorithm are identifying potentially optimal intervals or hyperrectangles and its strategy of division. We will give a brief description as follows.

3.2.1 Dividing hyperrectangles

The DIRECT algorithm begins by scaling the domain, Ω, to an n dimensional unit hypercube. The domain Ω is identified as the first potentially optimal hyperrectangle and DIRECT initiates its search by evaluating objective function at the center point of the original domain Ω, $c_e = (1/2, \ldots, 1/2)$. Then it continues search process by evaluating the objective function at the sampled points $c_e \pm \delta e_i$ which are determined as equidistant to the center c_e. Where δ is the one third of the distance to the hypercube, and e_i is the i^{th} unit vector. Then the algorithm moves to the next phase of the iteration by dividing the potentially optimal hyperrectangle. The dividing process is done by trisecting in all directions and the trisection is based on the directions with the smallest objective function value. For two-dimensional case, two dividing scheme are illustrated in Figure 3.1.

If we need to subdivide one of the hyperrectangles, we only divide them along their longest side, which ensure that we are able to get a decrease in the maximal length of the hyperrectangle.

3.2.2 Identifying Potentially Optimal Hyperrectangles

Another phase of the algorithm is identifying potentially optimal hyperrectangles which is defined below.

Definition 3.4. Assuming that the unit hypercube with center c_i is divided into m hyperrectangles, a hyperrectangle j is said to be potentially optimal if there

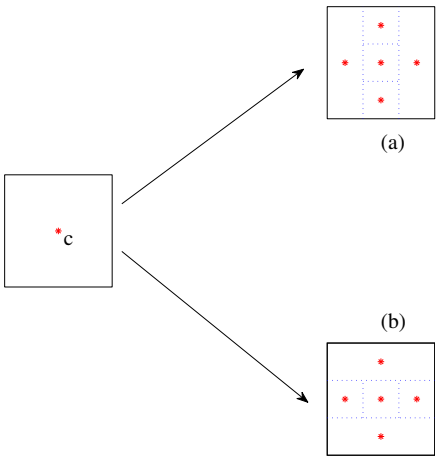

FIGURE 3.1: Different division strategies for hyperrectangles for two dimensions.

exists rate-of-change constant $\widetilde{K}$ such that

$$f(c_j) - \widetilde{K}\sigma_j \leq f(c_i) - \widetilde{K}\sigma_j, \forall i = 1, \ldots, m, \tag{3.4}$$

$$f(c_j) - \widetilde{K}\sigma_j \leq f_{min} - \varepsilon|f_{min}|, \tag{3.5}$$

where f_{min} is the current best function value, and σ_j is the distance from the center of hyperrectangle j to its vertices in [47]. $f(c_j)$ is the function value at the center point of hyperrectangle j, and the positive ε , a parameter that balances between global and local search, ensures that we have the possibility of a sufficient decrease in the potentially optimal hyperrectangle.

It is not clear from the definition (3.4) to identify potentially optimal heperrectangles. The following lemma will show an easy way to identify potentially optimal hyperrectangles.

Lemma 3.5. *Let $\varepsilon > 0$ be a positive constant and let f_{min} be the current best function value, I be the set of all indices of all intervals. Let $I_1 = \{i \in I : \sigma_i < \sigma_j\}$, $I_2 = \{i \in I : \sigma_i > \sigma_j\}$ and $I_3 = \{i \in I : \sigma_i = \sigma_j\}$. Hyperrectangle j is potentially optimal if*

$$f(c_j) \leq f(c_i), \forall i \in I_3, \tag{3.6}$$

there $\exists \widetilde{K} > 0$ *such that*

$$\max_{i \in I_1} \frac{f(c_j) - f(c_i)}{\sigma_j - \sigma_i} \leq \widetilde{K} \leq \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{\sigma_i - \sigma_j} \tag{3.7}$$

Proof. According to the definition (3.1) and I_1, I_2, I_3, (3.6) becomes

$$\widetilde{K} \geq \frac{f(c_j) - f(c_i)}{\sigma_j - \sigma_i}, \forall i \in I_1, \tag{3.8}$$

$$\widetilde{K} \leq \frac{f(c_i) - f(c_j)}{\sigma_i - \sigma_j}, \forall i \in I_2, \tag{3.9}$$

and

$$f(c_j) \leq f(c_i), \forall i \in I_3. \tag{3.10}$$

From (3.10), it follows that hyperrectangle j can only be potentially optimal if $f(c_j) = \min_{i \in I_3} f(c_i)$. If j satisfies this condition, then equations (3.8) and (3.9) give

$$\max_{i \in I_1} \frac{f(c_j) - f(c_i)}{\sigma_j - \sigma_i} \leq \widetilde{K} \leq \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{\sigma_i - \sigma_j} \tag{3.11}$$

Now we go back to inequality (3.5), hyperrectangle j can only be potentially optimal if $\exists \widetilde{K} > 0$ satisfying inequality (3.7). We want to make $\widetilde{K}$ as large as possible. Therefroe the hyperrectangle with index j is potentially optimal if

$$\varepsilon \leq \frac{f_{min} - f(c_j)}{|f_{min}|} + \frac{\sigma_j}{|f_{min}|} \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{\sigma_i - \sigma_j}, f_{min} \neq 0, \tag{3.12}$$

or

$$f(c_j) \leq d_j \min_{i \in I_2} \frac{f(c_i) - f(c_j)}{\sigma_i - \sigma_j}, f_{min} = 0. \tag{3.13}$$

which completes the proof. □

Readers are reffered to some other potentially optimal identification in [36].

The phase of identifying potentially optimal hyperrectangles combines the purposes of global and local search. The potentially optimal hyperrectangle(s)

with smallest function value will be divided into sub-hyperrectangles along the longest side, which is essential for proving convergence and can also ensure that hyperrectangles shrink on every dimension. This strategy increases the attractiveness of searching optimal function value with fewer number of local optima.

We claim that the DIRECT algorithm will converge to the global optimum because the maximum size, $\max \sigma_j, \forall j$, is able to decrease to zero and the entire search space can be thoroughly explored if needed.

More details on the DIRECT algorithm and its modification can be found in [36, 47].

3.2.3 Algorithm Description

We now have the main parts of the DIRECT algorithm. Each iteration begins by identifying the set of potentially optimal hyperrectangles which are then sampled and subdivided again. The process continues until a prespecified iteration limit $numit$ (or a prespecified number of function evaluations limit, $numfunc$) is reached. The formal statement of the DIRECT algorithm is described as bellow.

Algorithm 1 DIRECT algorithm

1: Normalize the search space to be the unit hypercube with center point c_1.
2: Evalute $f(c_1)$, $f_{min} = f(c_1), t = 0, m = 1$.
3: **while** $t \leq numit$ and $m \leq numfunc$ **do**
4: Identify the set S of potentially optimal hyperrectangles
5: **while** $S \neq \emptyset$ **do**
6: Take $j \in S$.
7: Sample new points, evaluate f at the new points and divide the hyperrectangle.
8: Upate $f_{min}, m = m + \Delta m$
9: Set $S = S \setminus \{j\}$
10: **end while**
11: $t = t + 1$.
12: **end while**

The DIRECT algorithm is suitable for black-box optimization problem since it dose not exploit and a priori knowledge on the objective function. The idea is to carry out simultaneous searches in all directions using all possible constant and to regard the Lipschitz constant as a weighting parameter to balance global and local search. And DIRECT performs a sampling of feasible domain on a set of points

that become dense in the limit, which guarantees strong theoretical convergence properties [47].

Usually the optimal function value is not known, therefore we are not able to get a function value within the suitable number of maximum iterations or function evaluation for a certain accuracy. Fortunately, the algorithm is effective for the typical dimension (less than 30) of a problem as cited in [47]. Next, we will give some statistic results to show the performance of DIRECT for optimizing the SOLR problem.

3.3 Performance of the DIRECT Algorithm for Lower Dimension

In this section, we give some results on solving some specific examples with DIRECT. As we mentioned above, DIRECT can be converged to a global optimum after a large number of iteration. However, it will cost much computational time. In the case that we only need to get an approximate solution, the early iteration of DIRECT can provide a rapid rate of improvement as it showed in Figure 3.2 . Figure 3.2 is on a typical example of the function evaluations VS. the best value found by DIRECT for solving the SOLR problem in [21]. As the figure shows, DIRECT provide a rapid rate of improvement at early iterations, while the later iterations do not improve the solution very much.

Example 1 (see [21])

$$(EX_1)\quad \left|\begin{array}{ll} \text{minimize} & \frac{-x_1+2x_2+2}{3x_1-4x_2+5}+\frac{4x_1-3x_2+4}{-2x_1+x_2+3} \\ \text{subject to} & x_1+x_2\le 1.5, \\ & x_1\le x_2, \\ & 0\le x_1\le 1, 0\le x_2\le 1. \end{array}\right. \tag{3.14}$$

For Example 1, we also test the cost of function evaluations for improving the tolerance of the optimal value showed in Table 3.1.

In Table 3.1, f_{min} is the result found by DIRECT with given tolerance, and $f(x^*)$ is the optimal solution of example 1. As it showed in Figure 3.2 and Table 3.1, the DIRECT can get a high approximation solution (e. g. tolerance $\varepsilon = 10^{-5}$)

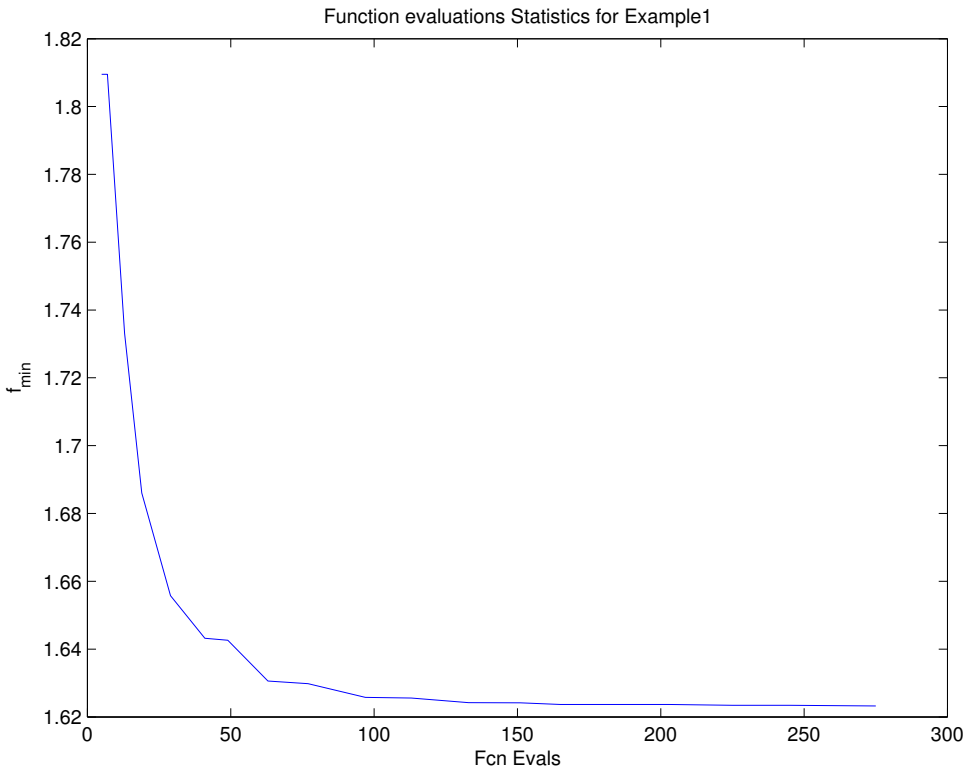

FIGURE 3.2: Results of DIRECT for Example 1.

$\lvert f_{min} - f(x*)\rvert$	$f - eval.$	CPU time (s)
0.1	13	0.1617
0.01	63	0.3073
10^{-3}	133	0.3991
10^{-4}	245	0.4404
10^{-5}	15963	5.2781

TABLE 3.1: Convergence of DIRECT for example 1 with different torlerance.

in a reasonable CPU time. In Section 3.4, we will give more results on DIRECT for solving SOLR.

3.4 Numerical Results

Now we give some more results and a statistic results on DIRECT for solving SOLR with lower dimension. The algorithm is coded in MATLAB® and implemented on an iMac with a Core i5 and 8GB of RAM. The results are summarized in Table 3.2 for a tolerance $\varepsilon = 10^{-4}$.

Example 2 (see [87])

$$(EX_2)\quad \left|\begin{array}{ll} \text{minimize} & \frac{2x_1+x_2}{x_1}+\frac{2}{x_2} \\ \text{subject to} & 2x_1+x_2 \le 6, \\ & 3x_1+x_2 \le 8, \\ & x_1-x_2 \le 1, \\ & x_1 \ge 1, \\ & 1 \le x_2 \le 6. \end{array}\right.$$

Example 3 (see [59])

$$(EX_3)\quad \left|\begin{array}{ll} \text{minimize} & (-\frac{-3x_1+5x_2+3x_3+50}{3x_1+4x_2+5x_3+50})+(-\frac{3x_1+4x_2+50}{4x_1+3x_2+2x_3+50})+(-\frac{4x_1+2x_2+4x_3+50}{5x_1+4x_2+3x_3+50}) \\ \text{subject to} & 6x_1+3x_2+3x_3 \le 10, \\ & 10x_1+3x_2+8x_3 \le 10, \\ & x_1,x_2,x_3 \ge 0. \end{array}\right.$$

The dimension of the 3 examples are less than 4, and the DIRECT algorithm

Problem	optimal value	CPU time (s)
P1	1.62326168	0.3456
P2	-6.4994	0.3498
P3	-3.0021	0.7763

TABLE 3.2: Results on 3 examples with tolerance $\varepsilon = 10^{-4}$.

is to be effective. Jones et al. [47] terminate DIRECT when it has finished a given number of iterations. And they also report numerical results of DIRECT for different test problems in [47] .

For the case the global minimum of the test problem is known, we can terminate DIRECT for a given tolerance mentioned in [47]. However, as mentioned above, most of the global minimal function value is unknown in real applications. Therefore, the termination criteria by Jones cannot be used. Instead, we use a more realistic termination criteria with given, fixed function budgets showed in Tabel 3.3. For a test problem, we let a large number of function evaluations (e.g. the number of function evaluations is 10^7) as a termination criteria. We can pick another objective function value at some function evaluations such that the difference between the larger objective function value and the smaller one is within a

given tolerance ε. We did experiments for 1000 instances for different dimensions and the statistic results is showed as follows.

N	$f-eval.$	Rate(%)
2	8100	99.99
3	15400	99.99
4	45800	99.99

TABLE 3.3: Numerical results with a budget of different function evaluations for low dimensions.

Tabel 3.3 shows the statistic results for DIRECT with fixed function evaluations budget. We accept a point as a solution if its tolerance ε is lower than 10^{-4}. In the column labeled "Rate", we report how often DIRECT was able to find a solution within the given budget of function evaluations. We also did a numerical experiment for solving a more general SOLR problem using DIRECT, and the data of the test problems are set as follows: n_{ij}, d_{ij} are i.i.d. generated in the ranges of $-5 \leq n_{ij} \leq 5, -5 \leq d_{ij} \leq 5$, for $i = 1, \cdots, p$ and $j = 1, 2, 3, 4$. The a_i and b_i are fixed to 50, 90, respectively. The constraints and $x \in$ [0,5] used in numerical experiments for dimention 2, 3, 4 are

$$A_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, c_2 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; A_3 = \begin{pmatrix} 6 & 3 & 3 \\ 10 & 3 & 8 \end{pmatrix}, c_3 = \begin{pmatrix} 10 \\ 10 \end{pmatrix};$$

$$A_4 = \begin{pmatrix} 2 & 2 & 5 & 3 \\ 1 & 6 & 3 & 4 \\ 5 & 9 & 2 & 8 \\ 9 & 3 & 7 & 1 \end{pmatrix}, c_4 = \begin{pmatrix} 10 \\ 10 \\ 10 \\ 10 \end{pmatrix}.$$

The results are shown in Table 3.4. We also did the experiment for solving SOLR with only one ratio for 5 dimensions with DIRECT algorithm, and it will take much more time (more than 2 hours) to get a solution within tolerance $\varepsilon = 10^{-4}$.

3.5 Conclusions

For an algorithm to be truly global, some effort must be allocated to the global search which ensures that the global optima will not be overlooked in feasible region. On the other hand, some effort must be on local search for efficiency, and

	p											
dim.	10	20	30	40	50	60	70	80	100	200	300	400
2	2.3	2.7	3.2	3.4	3.8	4.0	4.4	4.8	5.5	8.4	12.3	15.5
3	4.0	5.5	5.1	6.5	7.2	7.7	8.3	9.3	10.5	16.5	23.4	29.7
4	23.6	25.5	27.3	29.5	30.1	32.5	34.1	35.8	39.2	57.6	79.1	96.6

TABLE 3.4: The average CPU time (seconds) of DIRECT algorithm for dimension 2, 3, 4, with $p = 10$ through 400, and the budget of function evaluations are 8100, 15400, 45800, respectively.

local search ensures that the current best solution will be achieved in a local area. Generally, most of algorithms strike a balance between local and global search using one or two approaches. One is to start with a large emphasis on global search and then shift to the emphasis on local search to get optimal solution (see eg. [29, 58]) . The other is to combine a local optimization techniques with some other technique that give a global aspect to the search (see eg. [40, 75]).

The DIRECT algorithm balances global and local search, and it does a little of both on each iteration. In every iteration, it selects rectangles that contain the potentially optimal and evaluate functional value on each sampling points to update the current best solution. Furthermore, the DIRECT algorithm carries out simultaneous searches with all possible Lipschitzian constant. The DIRECT algorithm does not require derivatives, therefore, it is suitable to some "difficult" problems.

We applied DIRECT to solve SOLR for different dimensions. We got the number of function evaluations for different dimensions with given tolerance $\varepsilon = 10^{-4}$. And the results show that we have a high probability (99.99%) to get a "good" solution in an efficient CPU time, which will help us to design some algorithm using the "good" starting point.

Chapter 4

A linear Relaxation Algorithm for Solving the SOLR Problem

4.1 Introduction

In this chapter, we focus on a revision of the linear relaxation algorithm [16] with lower dimension and we also develop an new branch and bound algorithm for solving the SOLR problem with lower dimension based on new branch rule. The new branch and bound algorithm based on bisection branching rules has been published in [43, 45], and the branch and bound algorithm [44] based on a sophisticated branching rules mentioned in section 4.4 is to be submitted for publication.

Carlsson and Shi [16] casted the SOLR problem into an equivalent problem with linear objective and a set of linear and nonconvex quadratic constraints. By dropping out the nonconvex quadratic constraints, they proposed a linear relaxation for the SOLR problem and designed a branch-and-bound algorithm to solve the SOLR problem with lower dimension. In addition, Kuno and Masaki [60] also focused on the problem with lower dimension and proposed an algorithm for solving a kind of the SOLR problem with different branching rules.

To circumvent the nonconvex quadratic constraints in [16], we do not drop the nonconvex constraints out but make a linear relaxation for them with some extra variables. Therefore, this linear relaxation is generally tighter than the previous one. In addition, we implemented numerical results to evaluate performance of

the proposed algorithm, and the results show that CPU times and the number of iterations, branches are sharply decreased when we use the proposed algorithm compared the original algorithm. These features push us to develop a new branch rule for solving the SOLR problem.

4.2 Equivalent Transformation and Linear Relaxation

4.2.1 Equivalent Transformation

We rewrite the SOLR problem as follows

$$(P_0) \quad \left| \begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = \sum_{i=1}^{p} \dfrac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \\ \text{subject to} & \boldsymbol{x} \in X = \{\boldsymbol{x} \in \mathbb{R}^n \mid A\boldsymbol{x} \leq \boldsymbol{c}, \boldsymbol{x} \geq \boldsymbol{0}\}, \end{array} \right.$$

where $p \geq 2$, $\boldsymbol{n}_i, \boldsymbol{d}_i$ are vectors in $\mathbb{R}^n$ and a_i, b_i are real numbers for all $i = 1, \ldots, p$, A is an $m \times n$ matrix, $\boldsymbol{c}$ is a vector in $\mathbb{R}^m$. We assume that $\boldsymbol{d}_i^\top \boldsymbol{x} + b_i \neq 0$ on X for all $i = 1, 2, \ldots, p$. Without loss of generality, we suppose that $\boldsymbol{d}_i^\top \boldsymbol{x} + b_i > 0$ in this chapter based on the following proposition

Proposition 4.1. *Assume* $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^1 + b_i\right) \neq 0, \forall \boldsymbol{x} \in X$, *for* $\forall \boldsymbol{x}^1, \boldsymbol{x}^2 \in X$, *we have* $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^1 + b_i\right)\left(\boldsymbol{d}_i^\top \boldsymbol{x}^2 + b_i\right) > 0$.

Proof. Without loss of generality, we suppose that $\exists\, \boldsymbol{x}^1, \boldsymbol{x}^2 \in X$ such that $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^1 + b_i\right) > 0$ and $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^2 + b_i\right) < 0$. Then there $\exists\, \boldsymbol{x}^\sigma = \boldsymbol{x}^1 + \sigma(\boldsymbol{x}^2 - \boldsymbol{x}^1) \in X$ such that $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^\sigma + b_i\right) = 0$, where $\sigma \in (0, 1)$. It contradicts the assumption of $\left(\boldsymbol{d}_i^\top \boldsymbol{x}^1 + b_i\right) \neq 0, \forall \boldsymbol{x} \in X$. So the conclusion is valid. □

Because the set X is nonempty and bounded, we can construct a rectangle $B = [\boldsymbol{l}, \boldsymbol{u}]$ which contains the feasible region X. We denote that $\boldsymbol{l} = (l_1, l_2, \ldots, l_n)^\top$, $\boldsymbol{u} = (u_1, u_2, \ldots, u_n)^\top$, where l_j, u_j are the optimal values of the linear programming

problem (4.1) and (4.2), respectively.

$$\begin{aligned} l_j := \text{minimize} \quad & x_j \\ \text{subject to} \quad & \boldsymbol{x} \in X. \end{aligned} \tag{4.1}$$

$$\begin{aligned} u_j := \text{maximize} \quad & x_j \\ \text{subject to} \quad & \boldsymbol{x} \in X. \end{aligned} \tag{4.2}$$

Consider problem of (P_0) with box $B = [\boldsymbol{l}, \boldsymbol{u}]$ as follows

$$(P_1) \quad \left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \\ \text{subject to} & A\boldsymbol{x} \leq \boldsymbol{c}, \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \end{array} \right. \tag{4.3}$$

where $\mathbf{0} \leq \boldsymbol{l} \leq \boldsymbol{u}$. Becasuse $X \subseteq B$ from (4.1) and (4.2), problem (P_0) is equivalent to problem (P_1). Let us apply the Charnes-Cooper transformation [19] to (4.3), by introducing $2p$ variables:

$$\boldsymbol{y}^i := z_i \boldsymbol{x}, \;\; z_i := \frac{1}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i}, i = 1, 2, \ldots, p.$$

It is easy to see that $A\boldsymbol{x} \leq \boldsymbol{c}$ if any only if $A\boldsymbol{y} - \boldsymbol{c}z \leq \mathbf{0}$ and that $\boldsymbol{x} \in [\boldsymbol{l}, \boldsymbol{u}]$ if and only if $-\boldsymbol{y} + \boldsymbol{l}z \leq \mathbf{0}$ and $\boldsymbol{y} - \boldsymbol{u}z \leq \mathbf{0}$ in the sense that $\boldsymbol{y} = \boldsymbol{x}/(\boldsymbol{d}^\top \boldsymbol{x} + b), z = 1/(\boldsymbol{d}^\top \boldsymbol{x} + b)$.

Denote that $\alpha_i := \min \left\{ \boldsymbol{d}_i^\top \boldsymbol{x} + b_i \,\middle|\, \boldsymbol{x} \in X \right\}$ and $\beta_i := \max \left\{ \boldsymbol{d}_i^\top \boldsymbol{x} + b_i \,\middle|\, \boldsymbol{x} \in X \right\}$. Then we have

$$(P_2) \quad \left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i \right) \\ \text{subject to} & \left. \begin{array}{l} \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ A\boldsymbol{y}^i - \boldsymbol{c}z_i \leq \mathbf{0}, \\ \dfrac{1}{\beta_i} \leq z_i \leq \dfrac{1}{\alpha_i}, \\ \boldsymbol{y}^i z_j = \boldsymbol{y}^j z_i, \\ z_i \boldsymbol{l} \leq \boldsymbol{y}^i \leq z_i \boldsymbol{u}. \end{array} \right\} i, j = 1, 2, \ldots, p. \end{array} \right. \tag{4.4}$$

We will see that problem (P_2) is equivalent to problem (P_1) from the following theorem.

Theorem 4.2. *Problem (P_2) is equivalent to problem (P_1).*

Proof. The following proof is similar to Theorem 1 in [16]. For self-contained we give a proof below.

Let $\boldsymbol{x}^*$ be an optimal solution to problem (P_1). Define that $(\boldsymbol{y}^i)^* := \boldsymbol{x}^*/(\boldsymbol{d}_i^\top \boldsymbol{x}^* + b_i)$, $z_i^* =: 1/(\boldsymbol{d}_i^\top \boldsymbol{x}^* + b_i)$ then we see that $((\boldsymbol{y}^i)^*, z_i^*)$ are feasible for problem (P_2). Suppose there is a feasible solution $((\boldsymbol{y}^i)', z_i')$ such that

$$\sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top (\boldsymbol{y}^i)' + a_i z_i'\right) < \sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top (\boldsymbol{y}^i)^* + a_i z_i^*\right). \tag{4.5}$$

Thus, we see that with $\boldsymbol{x}' = (\boldsymbol{y}^i)'/z_i'$,

$$\sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top (\boldsymbol{y}^i)' + a_i z_i'\right) = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}' + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}' + b_i}, \quad \sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top (\boldsymbol{y}^i)^* + a_i z_i^*\right) = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^* + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^* + b_i},$$

and $\boldsymbol{x}'$ is also a feasible solution to (P_1). Therefore, the inequality (4.5) contradicts the optimality of x^* for P_1. And similar to vice versa. □

4.2.2 Linear Ralaxation

From Theorem 4.2, in order to globally solve (4.3), we may solve (4.4) instead because all (P_0), (P_1) and (P_2) are equivalent. However, the constraints $\boldsymbol{y}^i z_j = \boldsymbol{y}^j z_i$ for $i, j = 1, 2, \ldots, p$ in (4.4) are quadratic and nonconvex. Therefore, this problem can be solved in the category of nonconvex quadratic programming, which is general, of course, $\mathcal{N}P$-hard.

It is trivial that we can make a relaxation for problem (P_2) by discarding $\boldsymbol{y}^i z_j = \boldsymbol{y}^j z_i$ for all $i, j = 1, 2, \ldots, p$ as follows:

$$Q_0(\boldsymbol{l}, \boldsymbol{u})) \left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i\right) \\ \text{subject to} & \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ & A\boldsymbol{y}^i - \boldsymbol{c} z_i \leq \boldsymbol{0}, \\ & \dfrac{1}{\beta_i} \leq z_i \leq \dfrac{1}{\alpha_i}, \\ & z_i \boldsymbol{l} \leq \boldsymbol{y^i} \leq z_i \boldsymbol{u}. \end{array} \right\} i = 1, 2, \ldots, p. \tag{4.6}$$

Let $(\boldsymbol{y}^i, z_i), i = 1, 2, \ldots, p$ be the optimal solution of problem Q_0, then the following claim holds

Proposition 4.3. *Let $(\boldsymbol{y}^i, z_i), i = 1, 2, \ldots, p$ be the optimal solution of problem Q_0, and*

$$\frac{\boldsymbol{y}^i}{z_i} = \frac{\boldsymbol{y}^j}{z_j}, \forall i,\ j = 1, 2, \ldots, p. \tag{4.7}$$

Then $\boldsymbol{x}^* = \frac{\boldsymbol{y}^i}{z_i}, i = 1, 2, \ldots, p$ is an optimal solution of problem (4.3).

Proof. If (4.7) holds, which implies that $\boldsymbol{y}^i z_j = \boldsymbol{y}^j z_i$, $\forall i,\ j = 1, 2, \ldots, p$. Therefore, $(\boldsymbol{y}^i, z_i)$, $i = 1, 2, \ldots, p$ satisfies all constraints in (4.4). Then $\boldsymbol{x}^* = \frac{\boldsymbol{y}^i}{z_i}, i = 1, 2, \ldots, p$ is an optimal solution of problem (4.3) followed by Theorem 4.2. □

The problem (4.6) is a linear programming problem. In their paper [16], Carlsson and Shi fundamentally exploited this linear form (4.6) to obtain the lower bounds of (4.4) and design a branch and bound algorithm for solving the SOLR problem.

In this study, we devise a linear relaxation of $\boldsymbol{y}^i z_j = \boldsymbol{y}^j z_i$ with $-\boldsymbol{y}^i + \boldsymbol{l} z_i \leq \boldsymbol{0}$ and $\boldsymbol{y}_i - \boldsymbol{u} z_i \leq \boldsymbol{0}$. Note that $1/\beta_i \leq z_i \leq 1/\alpha_i$ and $z_i > 0$ for all i. Thus we see that for each i

$$\boldsymbol{l}/\beta_i \leq \boldsymbol{y}^i \text{ and } \boldsymbol{y}^i \leq \boldsymbol{u}/\alpha_i \text{ for all } i = 1, \ldots, p.$$

We consider two sets B_{curv} and B_{tria} that are defined below.

$$B_{\text{curv}} := \{(\boldsymbol{t}_{ij}, \boldsymbol{y}^i, z_j) \,|\, \boldsymbol{t}_{ij} = \boldsymbol{y}^i z_j, 1/\beta_j \leq z_j \leq 1/\alpha_j, \boldsymbol{l}/\beta_i \leq y^i \leq \boldsymbol{u}/\alpha_i\}$$

and

$$B_{\text{tria}} := \left\{ (\boldsymbol{t}_{ij}, \boldsymbol{y}^i, z_j) \;\middle|\; \begin{array}{l} \boldsymbol{t}_{ij} \le \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\alpha_j \beta_i}, \\ \boldsymbol{t}_{ij} \le \frac{1}{\beta_j}\boldsymbol{y}^i + \frac{\boldsymbol{u}}{\alpha_i} z_j - \frac{\boldsymbol{u}}{\alpha_i \beta_j}, \\ \boldsymbol{t}_{ij} \ge \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{u}}{\alpha_i} z_j - \frac{\boldsymbol{u}}{\alpha_i \alpha_j}, \\ \boldsymbol{t}_{ij} \ge \frac{1}{\beta_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\beta_i \beta_j}, \\ 1/\beta_j \le z_j \le 1/\alpha_j, \\ \boldsymbol{l}/\beta_i \le \boldsymbol{y}^i \le \boldsymbol{u}/\alpha_i. \end{array} \right\}$$

It is easy to see that

Lemma 4.4. *The relation $B_{curv} \subseteq B_{tria}$ holds.*

Proof. We first prove the case that $\boldsymbol{t}_{ij} \le \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\alpha_j \beta_i}$. For $\forall (\boldsymbol{t}_{ij}, \boldsymbol{y}^i, z_j) \in B_{\text{curv}}, \forall i, j = 1, 2, \ldots, p$.

$$\begin{aligned} & \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\alpha_j \beta_i} - \boldsymbol{t}_{ij} \\ = \; & \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\alpha_j \beta_i} - \boldsymbol{y}^i z_j \\ = \; & \left(\frac{1}{\alpha_j} - z_j\right)\left(\boldsymbol{y}^i - \frac{\boldsymbol{l}}{\beta_i}\right). \end{aligned}$$

Since $z_j \le 1/\alpha_j$ and $\boldsymbol{y}^i \ge \boldsymbol{l}/\beta_i$, we have $\frac{1}{\alpha_j}\boldsymbol{y}^i_k + \frac{\boldsymbol{l}}{\beta_i} z_j - \frac{\boldsymbol{l}}{\alpha_j \beta_i} - \boldsymbol{t}_{ij} \ge \boldsymbol{0}$. Using a similar way, we can easily prove that the other cases in the lemma hold. □

Lemma 1 indicates that the following problem $Q_1(\boldsymbol{l}, \boldsymbol{u})$ can be exploited to find a lower bound for problem $Q_0(\boldsymbol{l}, \boldsymbol{u})$ with a box $[\boldsymbol{l}, \boldsymbol{u}]$:

$$
Q_1(\boldsymbol{l}, \boldsymbol{u}) \left| \begin{array}{ll} \text{minimize} & \displaystyle\sum_{i=1}^{p} \left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i\right) \\ \text{subject to} & \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ & A\boldsymbol{y}^i - \boldsymbol{c} z_i \leq \boldsymbol{0}, \\ & \dfrac{1}{\beta_i} \leq z_i \leq \dfrac{1}{\alpha_i}, \\ & -\boldsymbol{y}^i + \boldsymbol{l} z_i \leq \boldsymbol{0}, \boldsymbol{y}^i - \boldsymbol{u} z_i \leq \boldsymbol{0}, \\ & \boldsymbol{t}_{ij} \leq \dfrac{1}{\alpha_j}\boldsymbol{y}^i + \dfrac{\boldsymbol{l}}{\beta_i} z_j - \dfrac{\boldsymbol{l}}{\alpha_j \beta_i}, \\ & \boldsymbol{t}_{ij} \leq \dfrac{1}{\beta_j}\boldsymbol{y}^i + \dfrac{\boldsymbol{u}}{\alpha_i} z_j - \dfrac{\boldsymbol{u}}{\alpha_i \beta_j}, \\ & \boldsymbol{t}_{ij} \geq \dfrac{1}{\alpha_j}\boldsymbol{y}^i + \dfrac{\boldsymbol{u}}{\alpha_i} z_j - \dfrac{\boldsymbol{u}}{\alpha_i \alpha_j}, \\ & \boldsymbol{t}_{ij} \geq \dfrac{1}{\beta_j}\boldsymbol{y}^i + \dfrac{\boldsymbol{l}}{\beta_i} z_j - \dfrac{\boldsymbol{l}}{\beta_i \beta_j}, \\ & \boldsymbol{t}_{ij} = \boldsymbol{t}_{ji}. \end{array} \right\} i, j = 1, 2, \ldots, p. \tag{4.8}
$$

Next we will develop a branch and bound algorithm to find an optimal solution to (4.3) by solving a series of the linear programming problem (4.8).

4.3 A Branch and Bound Algorithm Based on Bisection Branching Rules

4.3.1 Branch and Bound Algorithm Review

It is well know that Branch and Bound (B&B) algorithms have been applied successfully on various $\mathcal{NP}$-hard problems, such as *Traveling Salesman Problems* [17, 65], *Schedule Problems* [7], *Mixed Integer Programming Problems* [1]. It is an intelligent search heuristic for finding a global optimum to problems of the form $\min_{\boldsymbol{x} \in X} f(\boldsymbol{x})$. Basic B&B searches feasible region X by iteratively subdividing the feasible region and recursively searching each piece for an optimal feasible region. Also, B&B algorithms are able to solve large instance for optimality, especially for lower dimension, because it can eliminate regions which provably do not contain an optimal solution. Readers are refereed to [24] for details of B&B.

The following phenomenon frequently occurs in various optimization problems: a problem can be easily solved if we make some relaxation (e.g., remove some constraints that are hard to solve) for the original problem, see examples in [74]. Different relaxation techniques [32] are used to solve in numerous $\mathcal{NP}$ hard problems.

A B&B algorithm for a minimization problem hence consists of the follow 3 main components:

Bounds *Upper bound* is the value of the best solution obtained currently, and *lower bound* of a subproblem should be smaller than or equal to the optimum function value got so far. If a lower bound exceeds the valued of the current unbound for some subproblems, the corresponded subproblems will be discarded.

Search strategy The search strategy prescribes how the B&B algorithm should proceed trough the search tree. The two most common methods are *Depth First Search*, which solves the most recently generated subproblem first, and *Best First Search*, which solves the most promising subproblem first. The search strategy selected in this section is the *Best First Search*.

Branching rule At each node of a B&B search tree, the branching rule prescribes how the current problem should be divided into new subproblems. The branch rule in this section is bisection rules described bellow.

Suppose $B^k = \{\boldsymbol{x} \in \mathbb{R}| : \underline{x}_i \leq x_i \leq \overline{x}_i, i = 1, 2, \ldots, n\}$ is a rectangle that contains an optimal solution in the process. Then B^k is divided into two subrectangles B^{2k}, B^{2k+1} according to the following rules $R1$-$R3$:

$R1$. Let $i_0 \in \arg\max\{\overline{x}_i - \underline{x}_i \,|\, i = 1, 2, \ldots, n\}$.

$R2$. Let $\gamma_{i_0} = \frac{1}{2}(\underline{x}_{i_0} + \overline{x}_{i_0})$.

$R3$. Let $B^{2k} = \{\ \boldsymbol{x} \in \mathbb{R}^n \,|\, \underline{x}_i \leq x_i \leq \overline{x}_i, i \neq i_0,\ \underline{x}_{i_0} \leq x_{i_0} \leq \gamma_{i_0}\}$,

$$B^{2k+1} = \{\boldsymbol{x} \in \mathbb{R}^n \,|\, \underline{x}_i \leq x_i \leq \overline{x}_i, i \neq i_0,\ \gamma_{i_0} \leq x_{i_0} \leq \overline{x}_{i_0}\}.$$

4.3.2 Algorithm Description

In our algorithm, the branching process is executed in the space of $\mathbb{R}^n$. Starting from $B^1 = \{[\boldsymbol{l}^1, \boldsymbol{u}^1] = [\boldsymbol{l}, \boldsymbol{u}]\}$ that is generated by solving problem (4.1) and (4.2),

we solve $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ for $k = 1, 2, \ldots$. The rectangle B^k will be discarded at the k-th iteration if the optimal value of $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ is greater than the current best value at the solution $\boldsymbol{x}^{i_0} = \boldsymbol{y}^{i_0}/z_{i_0}$, $i_0 \in \arg\min\{f(\boldsymbol{x}^{i_k}), i_k = 1, 2, \ldots, p\}$. Now the algorithm is ready to be described in detail as follows:

Algorithm 2 Branch and bound algorithm for the SOLR problem based on bisection rule

1: Initial settings: Set $k = 1$. Let $\mathcal{B}^k = \{[\boldsymbol{l}^k, \boldsymbol{u}^k]\} = \{[\boldsymbol{l}, \boldsymbol{u}]\}$ be the initial rectangle. $L = -\infty$, $U = \infty$.
2: Solve $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ with $B^j \in \mathcal{B}^k$. If $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ is feasible then obtain the optimal value L^k and $\boldsymbol{x}^{i_k} = \boldsymbol{y}^{i_k}/z_{i_k}$, and $U^k = \min\{f(\boldsymbol{x}^{i_k}), i_k = 1, \ldots, p\}$. $\mathcal{L}^k = L^k, U = U^k$. If $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ is infeasible then terminate.
3: Set a tolerance $\varepsilon \geqq 0$.
4: **while** $U - \mathcal{L}^k > \varepsilon$ **do**
5: Discard rectangles $B^j \in \mathcal{B}^k$ such that the value of $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j) > U^k$
6: Select $B^{j_0} \in \mathcal{B}^k$ with $L^{j_0} = \mathcal{L}^k$.
7: Divide B^{j_0} into two subrectangles B^{2k}, B^{2k+1} according branching rules $R1$-$R3$, and update $\mathcal{B}^k = (\mathcal{B}^k \setminus \{B^{j_0}\}) \cup \{B^{2k}\} \cup \{B^{2k+1}\}$
8: Solve $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ for $j = k, 2k+1$. If $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ is not feasible, then discard B^j. Otherwise, obtain $L^k, U^k, L^{2k+1}, U^{2k+1}$ and keep the corresponding incumbent best solution $\boldsymbol{x}^{j_k} = \boldsymbol{y}^{j_k}/z_{j_k}$.
9: Update $\mathcal{L}^k = \min\{L^k, L^{2k+1}\}$ if $\mathcal{L}^k < \min\{L^k, L^{2k+1}\}$, $U = \min\{U^k, U^{2k+1}\}$ if $U > \min\{U^k, U^{2k+1}\}$ and update the incumbent best solution.
10: Set $k = k + 1$
11: **end while**
12: Return U as an ε-minimum of (P_0) with a minimizer $\boldsymbol{x}^{j_k} = \boldsymbol{y}^{j_k}/z_{j_k} \in [\boldsymbol{l}, \boldsymbol{u}]$.

4.3.3 Algorithm's Convergent Proof

Let $\delta([\boldsymbol{l}^k, \boldsymbol{u}^k]) := \max\{u_i^k - l_i^k \mid i = 1, 2, \ldots, n\}$, which denotes the size (or diameter) of box $[\boldsymbol{l}^k, \boldsymbol{u}^k]$. Hereafter, we also use δ to denote the size of a box. The convergence of **Algorithm 1** can be shown by the following Lemma 4.5 and Theorem 4.6.

Lemma 4.5. *Suppose that* $(\boldsymbol{y}^i, z_i)$ *for* $i = 1, \ldots, p$ *are obtained from solving (4.8) and* $\boldsymbol{x} = \boldsymbol{y}^{i_0}/z_{i_0}$ *for any given* $i_0 \in \{1, \ldots, p\}$. *Then*

$$\left| \sum_{i=1}^{p} (\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i) - \sum_{i=1}^{p} \frac{\boldsymbol{n}_{i_0}^\top \boldsymbol{x} + a_{i_0}}{\boldsymbol{d}_{i_0}^\top \boldsymbol{x} + b_{i_0}} \right| \to 0 \text{ as } \delta([\boldsymbol{l}^k, \boldsymbol{u}^k]) \to 0. \tag{4.9}$$

Proof. This proof is almost as the same as Theorem 1 in [16]. For self-contained we give a proof below.

Without loss of generality, we suppose that $i_0 = 1$. Let $\boldsymbol{x}^i = \boldsymbol{y}^i/z_i$. Since $\boldsymbol{y}^i, z_i$ are feasible solutions of (4.8), it follows that $\boldsymbol{x}^i$ for $i = 1, \ldots, p$ are feasible for (4.3) . We see that for each i,

$$\left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i\right) \;=\; \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i} \,.$$

Let $\boldsymbol{x}_\delta = \boldsymbol{x} - \boldsymbol{x}^i$, then $||\boldsymbol{x}_\delta|| \leqq \delta([\boldsymbol{l}^k, \boldsymbol{u}^k])$. Denote that

$$\begin{aligned}
\tau_i &:= \text{maximize}\left\{\left|\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i\right| \,|\, \boldsymbol{x}^i \in X\right\} \\
\zeta &:= \text{maximize}\left\{||\boldsymbol{d}_i|| \,|\, i = 1, \ldots, p\right\} \\
\mu &:= \text{maximize}\left\{||\boldsymbol{n}_i|| \,|\, i = 1, \ldots, p\right\}
\end{aligned}$$

Suppose $\boldsymbol{x}^i = \boldsymbol{y}^i/z_i$ for all $i = 1, 2, \ldots, p$ and $\boldsymbol{x}$ are points in $[\boldsymbol{l}^k, \boldsymbol{u}^k]$.

$$\begin{aligned}
& \left|\sum_{i=1}^{p}(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i) - \sum_{i=1}^{p} \frac{\boldsymbol{n}_{i_0}^\top \boldsymbol{x} + a_{i_0}}{\boldsymbol{d}_{i_0}^\top \boldsymbol{x} + b_{i_0}}\right| \\
= & \left|\sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i} - \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^+ b_i}\right| \\
= & \left|\sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i} - \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i + \boldsymbol{n}_i^\top \boldsymbol{x}_\delta}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i + \boldsymbol{d}_i^\top \boldsymbol{x}_\delta}\right| \\
\leq & \left|\sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i} - \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i + \boldsymbol{d}_i^\top \boldsymbol{x}_\delta}\right| + \left|\sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}_\delta}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i + \boldsymbol{d}_i^\top \boldsymbol{x}_\delta}\right| \\
\leq & \sum_{i=1}^{p} \left|\frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i} \cdot \frac{\boldsymbol{d}_i^\top \boldsymbol{x}_\delta}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i + \boldsymbol{d}_i^\top \boldsymbol{x}_\delta}\right| + \sum_{i=1}^{p} \left|\frac{\boldsymbol{n}_i^\top \boldsymbol{x}_\delta}{\boldsymbol{d}_i^\top \boldsymbol{x}^i + b_i + \boldsymbol{d}_i^\top \boldsymbol{x}_\delta}\right| \\
\leq & \sum_{i=1}^{p} \left|\frac{\boldsymbol{n}_i^\top \boldsymbol{x}^i + a_i}{\alpha_i^2}\right| \left|\boldsymbol{d}_i^\top \boldsymbol{x}_\delta\right| + \sum_{i=1}^{p} \left|\frac{\boldsymbol{n}_i^\top \boldsymbol{x}_\delta}{\alpha_i}\right| \\
\leq & \delta \sum_{i=1}^{p} \left(\frac{\tau_i \zeta}{\alpha_i^2} + \frac{\mu}{\alpha_i}\right).
\end{aligned}$$

Therefore, the assertion holds as $\delta \to 0$. □

Theorem 4.6. *For a given tolerance ε, a global ε-minimizer of problem (P_0) can be found within finitely many iterations.*

Proof. A sufficient condition for a global optimization to be convergent to the global minimum, for instance, stated in Horst and Tuy [42], is that the bounding operation must be consistent and the selection operation must be bound improving. A bounding operation is called consistent if at every step any unfathomed partition can be further refined, and if any infinitely decreasing sequence of successively refined partition elements satisfies:

$$\lim_{k \to +\infty} (U - \mathcal{L}^k) = 0, \tag{4.10}$$

where U and $\mathcal{L}^k$ are the computed upper bound and the current best lower bound at iteration k, respectively.

Since the subdivision process is bisection, the process is exhaustive. Clearly, Lemma 4.5 keeps (4.10) holding, which implies that the employed bounding operation in our algorithm is consistent.

A selection operation is called bound improving if at least one partition element where the actual lower bound is attained will be selected for further partition after a finite number of refinements. Clearly, the partition element at step 6 in Algorithm 1 where the current lower bound is attained will be selected for further partition at the next iteration in our algorithm. Therefore, the employed selection operation improves the bound. We complete the proof of the convergence. □

4.3.4 Numerical Experiments

In this section, we give a numerical example to compare the efficiency between the revision and its previous algorithm. We also report the results of the numerical experiments which were conducted using randomly generated data sets to verify the performance of the revised algorithm. The algorithm is coded in MATLAB® and implemented on an iMac with a quad-core Core(i5) and 8GB of RAM in Muroran Institute of Technology.

We use the following Example 1 to see the empirical evidence that the new algorithm works efficiently well.

Example 4.1 ([21], *p.*78)

$$\begin{aligned} \text{minimize} \quad & \frac{-x_1 + 2x_2 + 2}{3x_1 - 4x_2 + 5} + \frac{4x_1 - 3x_2 + 4}{-2x_1 + x_2 + 3} \\ \text{subject to} \quad & x_1 + x_2 \leq 1.5, \\ & x_1 \leq x_2, \\ & 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1. \end{aligned}$$

Example 4.1 was solved by Algorithm SOLiRat in [16] and its revision Algorithm 2 that is proposed in this study, respectively. That is, Example 4.1 was solved based on two different linear relaxations Q_0 in (2.5) and Q_1 in (2.6) with $\varepsilon = 0.05$. The minimum of Example 4.1 is `1.62318`. In Figure 1, the horizontal and vertical axises are the number of iterations in the process and the lower & upper bounds the algorithms obtained at the iteration, respectively. The blue line depicts the lower and the red is for upper bounds which were calculated by the revised algorithm. We see that the gaps between the red and blue lines become very small after about only 5 iterations in this example. The gaps, in contrast, between the sky-blue and black lines keep a relative wider range even after 60 iterations. Figure 1 indicates that

- The new linear relaxation Q_1 is likely to be more efficient than Q_0.

To investigate the difference in the ability to produce a tighter lower bound between Q_0 and Q_1, we solve problem (P_0) with a variety of size (diameter) δ of the box constraints $\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$. The problem used in this numerical experiment is with $p = 5, n = 3$. Figure 2 indicates that

- The new linear relaxation Q_1 makes a better lower bound for any size δ of the box constraints.
- For both Q_0 and Q_1, the larger the size of the box constraints is, the worse the lower bounds become.
- The larger the size of box constraints is, the larger the difference in the lower bounds obtained from Q_0 and Q_1 is.
- For the sam size of box constraints, the larger number of ratios, the larger the difference in the lower bounds obtained from Q_0 and Q_1 is.

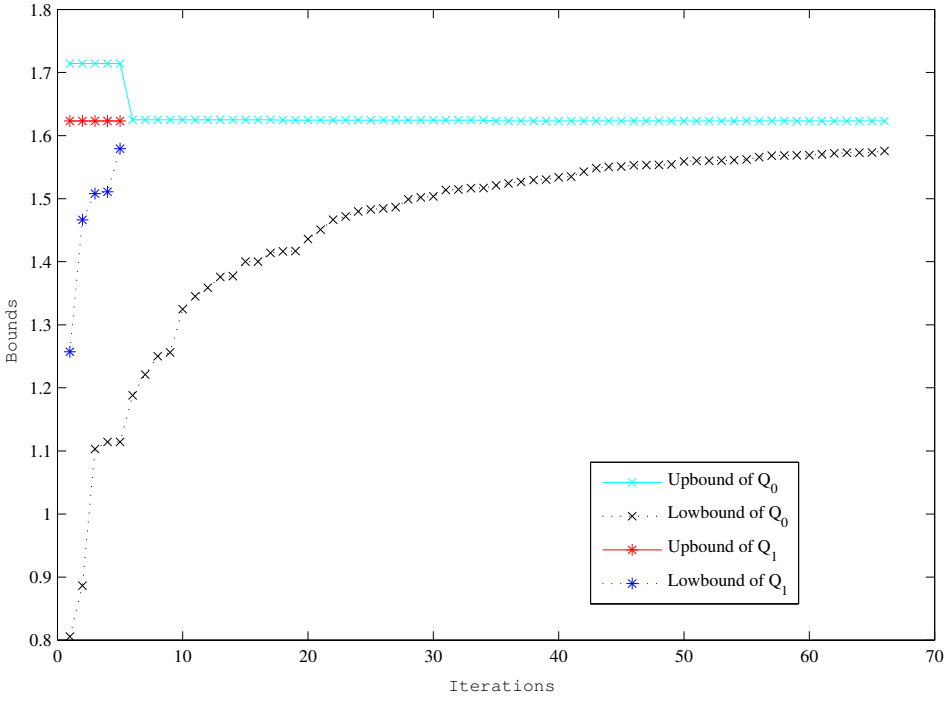

FIGURE 4.1: Bounds and iterations with $\varepsilon = 0.05$ for solving Example 4.1.

We also conducted the numerical experiments to evaluate the general behavior of the revised algorithm with lower dimension problems.

The datasets of the test problems used in this study are set as follows: The dimension of variables is 3 and the tolerance $\varepsilon = 0.05$. The coefficients $\boldsymbol{n}_i, \boldsymbol{d}_i$ are *i.i.d.* generated in the ranges of $-5 \leq n_{ij}, d_{ij} \leq 5$, for $i = 1, \ldots, p$ and $j = 1, 2, 3$. The constants a_i $(i = 1, \ldots, p)$ are randomly chosen from $[0, 60]$ and b_i $(i = 1, 2, \ldots, p)$ are fixed to 60.

Problem (P_0) with a variety of p from 5 to 80 was solved. For a fixed p, a set of 15 instances of the problem was solved by model Q_1 and model Q_0 using bisection branching rules, respectively.

The recorded CPU times in second, the number of iterations and the number of branches in the execution are displayed in Table 4.1, Table 4.2 and Table 4.3, respectively. The columns titled AVERG. provide the information about the average value of CPU time, the number of iterations and the number of branches out of the 10 runs in the execution, respectively.

As their names indicate, the rows $\mathcal{Q}_{1b}$ delivers the results that obtained from model Q_1 using bisection branching rules, while rows $\mathcal{Q}_{0b}$ for the results obtained

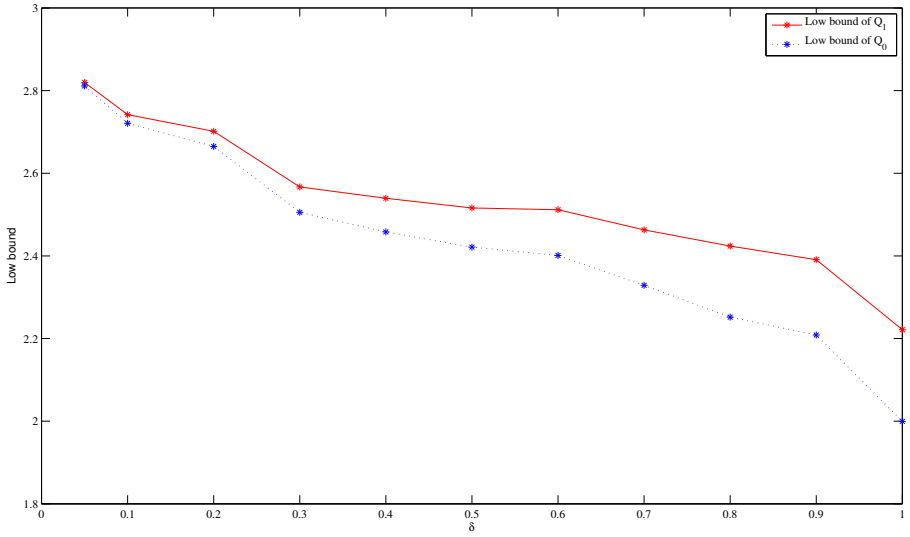

FIGURE 4.2: Average lowboud of two models with different size of box $p = 5$, $n = 3$.

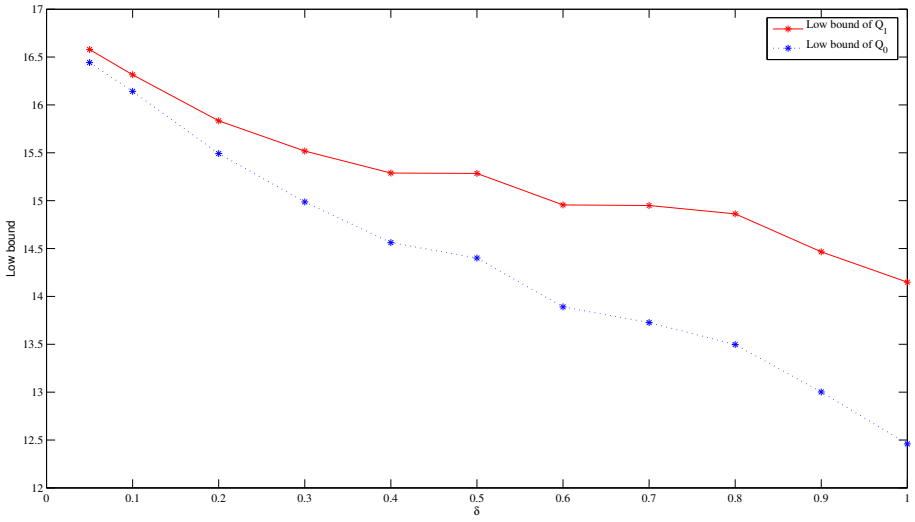

FIGURE 4.3: Average lowboud of two models with different size of box $p = 30$, $n = 3$.

from model Q_0 using bisection branching rules. Table 4.1, Table 4.2, Table 4.3 indicates the following observations.

- The number of branches is heavily reduced at least about to only 2% of the previous algorithm. A smaller number of branches in a branch-and-bound algorithm usually results in less time-consuming in implementation. In this study, the proposed algorithm consists of two parts: bounding based on a

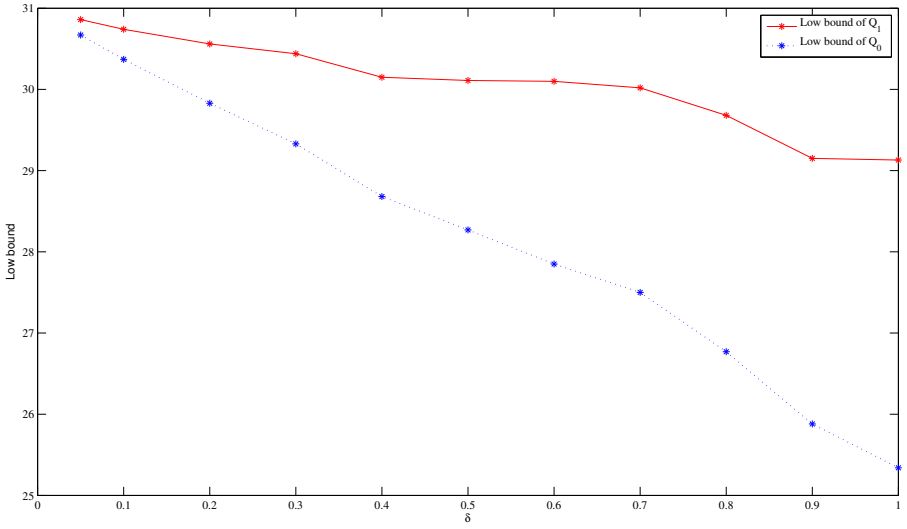

FIGURE 4.4: Average lowboud of two models with different size of box $p = 60$, $n = 3$.

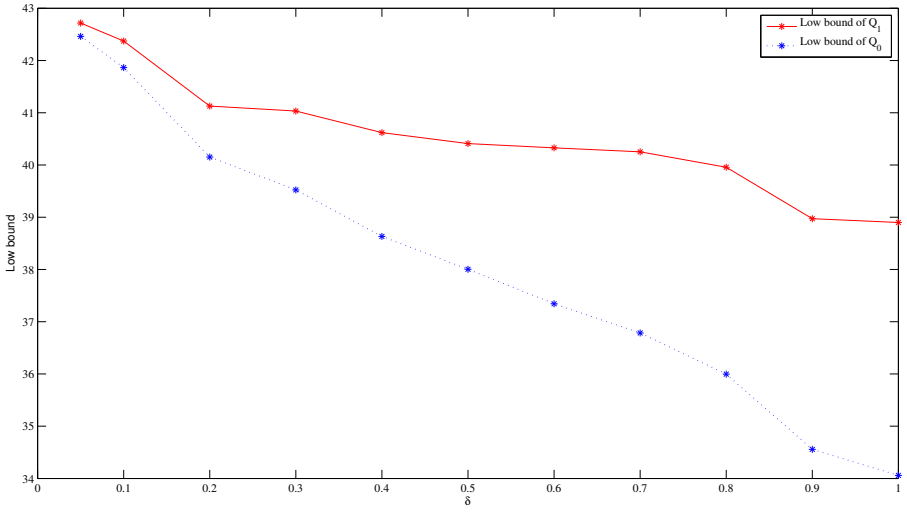

FIGURE 4.5: Average lowboud of two models with different size of box $p = 80$, $n = 3$.

LP relaxation and branching. The LP can be solved in polynomial time, so the reduction of the number of branches is fundamentally important to design an algorithm.

- The proposed LP relaxation is very efficient for solving problem (P_0) with various p from 5 to 80. The proposed algorithm achieves superiority over

Model	# p	CPU time(s)		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	0.1	0.3	0.6
$\mathcal{Q}_{0b}$	5	0.7	12.7	77.8
$\mathcal{Q}_{1b}$	10	0.4	0.8	2.0
$\mathcal{Q}_{0b}$	10	4.3	66.5	170.4
$\mathcal{Q}_{1b}$	20	2.7	8.7	14.4
$\mathcal{Q}_{0b}$	20	31.9	865.1	3295.4
$\mathcal{Q}_{1b}$	30	14.3	34.2	54.6
$\mathcal{Q}_{0b}$	30	242.1	3206.0	11544.8
$\mathcal{Q}_{1b}$	40	49.4	111.3	223.9
$\mathcal{Q}_{0b}$	40	425.6	10798.1	12774.5
$\mathcal{Q}_{1b}$	50	95.0	242.5	500.9
$\mathcal{Q}_{0b}$	50	993.1	11108.1	30781.0
$\mathcal{Q}_{b1}$	60	296.3	650.7	1782.9
$\mathcal{Q}_{0b}$	60	3446.0	16592.3	40149.1
$\mathcal{Q}_{1b}$	70	469.3	1879.4	7502.5
$\mathcal{Q}_{0b}$	70	6955.2	27989.7	59016.8
$\mathcal{Q}_{1b}$	80	1182.3	2177.5	6161.9
$\mathcal{Q}_{0b}$	80	8907.0	44494.3	98020.6

TABLE 4.1: Numerical results on CPU times for solving problem (P_0) with Q_1 and Q_0 with various p and $n = 3$.

the previous algorithm in CPU time, number of iterations and numbers of branches on average, as well as max and min values.

Because that Q_1 is a linear programming with a number of $(p^2n + pn + p)$ variables, Q_1 has a large number of variables when p is large. We see that such a large number has two sides:

1) Theoretically, the size of $(p^2n + pn + p)$ is a polynomial of the size of input data, and LP can be solved by a polynomial time of the size of input data, therefore we strongly expect that the proposed algorithm keeps its good efficiency even for a larger scale problem if we use a sophisticated software to solve the involved LP problems.

2) In reality, in our numerical experiments solving the problems with $p > 80$ the implementation reached the maximum memory of our computer. To take full advantage of this LP relaxation developed in this study the software

Model	# p	# Iterations		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	1	1.8	4
$\mathcal{Q}_{0b}$	5	6	114.7	714
$\mathcal{Q}_{1b}$	10	1	2.2	6
$\mathcal{Q}_{0b}$	10	18	283.2	735
$\mathcal{Q}_{1b}$	20	2	4.8	12
$\mathcal{Q}_{0b}$	20	73	1406.7	4150
$\mathcal{Q}_{1b}$	30	3	7.4	12
$\mathcal{Q}_{0b}$	30	198	3853.5	15851
$\mathcal{Q}_{1b}$	40	4	9.4	19
$\mathcal{Q}_{0b}$	40	266	6749.2	16377
$\mathcal{Q}_{1b}$	50	3	14.5	17
$\mathcal{Q}_{0b}$	50	863	8480.9	23497
$\mathcal{Q}_{1b}$	60	1	8.8	27
$\mathcal{Q}_{0b}$	60	1843	8873.7	21356
$\mathcal{Q}_{b1}$	70	3	10.7	21
$\mathcal{Q}_{0b}$	70	2557	10033.1	21362
$\mathcal{Q}_{1b}$	80	5	8.8	25
$\mathcal{Q}_{0b}$	80	3007	15042.0	32457

TABLE 4.2: Numerical results on iterations for solving problem (P_0) with Q_1 and Q_0 with various p and $n = 3$.

products that can solve large scale linear programming efficiently are highly recommended to use.

4.4 A Branch and Bound Algorithm Based on Advanced Branching Rules

The bisection branching rules, mentioned in section 4.3.1, is the standard bisection via the longest edge. It recursively divides the hyperrectangle containing a current best solution into sub-hyperrectangles, and discards the rectangles that the optimal solution can not be included. In this section, we propose a branch and bound algorithm to solve problem (4.8) based on sophisticated branching rules.

Model	# p	# Branches		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	1	2.13	5
$\mathcal{Q}_{0b}$	5	10	114.9	346
$\mathcal{Q}_{1b}$	10	1	3.9	10
$\mathcal{Q}_{0b}$	10	23	500.8	1351
$\mathcal{Q}_{1b}$	20	3	8.3	24
$\mathcal{Q}_{0b}$	20	106	2582.4	7818
$\mathcal{Q}_{1b}$	30	5	12.6	23
$\mathcal{Q}_{0b}$	30	282	6918.6	29523
$\mathcal{Q}_{1b}$	40	7	16.5	34
$\mathcal{Q}_{0b}$	40	513	12958.1	31012
$\mathcal{Q}_{1b}$	50	5	25.9	32
$\mathcal{Q}_{0b}$	50	1277	15640.1	37584
$\mathcal{Q}_{1b}$	60	2	15.8	46
$\mathcal{Q}_{0b}$	60	2745	15058.8	38440
$\mathcal{Q}_{1b}$	70	5	16.9	27
$\mathcal{Q}_{0b}$	70	3886.4	17495.3	32555
$\mathcal{Q}_{1b}$	80	7	16.3	48
$\mathcal{Q}_{0b}$	80	4794	24067.2	37560

TABLE 4.3: Numerical results on branches for solving problem (P_0) with Q_1 and Q_0 with various p and $n = 3$.

4.4.1 Feature of Best Solution in Native Dimension

It is well know that the feasible region for a set of linear inequalities is a polytope, and the optimal solution to LP problem constrained by a polytope is attained at some vertex(s) of the polytope. We consider the following 3 polytopes for problem (4.3), (4.6) and (4.8), $\forall i, j = 1, 2, \ldots, p$

$$M_1 = \left\{ \boldsymbol{x} \;\middle|\; \begin{array}{l} A\boldsymbol{x} \leq \boldsymbol{c}, \\ \boldsymbol{l}^k \leq \boldsymbol{x} \leq \boldsymbol{u}^k. \end{array} \right\} \tag{4.11}$$

$$M_{Q_0} = \left\{ (\boldsymbol{y}^i, z_i) \;\middle|\; \begin{array}{l} \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ A\boldsymbol{y}^i - \boldsymbol{c} z_i \leq \boldsymbol{0}, \\ \dfrac{1}{\beta_i} \leq z_i \leq \dfrac{1}{\alpha_i}, \\ z_i \boldsymbol{l}^k \leq \boldsymbol{y}^i \leq z_i \boldsymbol{u}^k, \end{array} \right\} \tag{4.12}$$

$$M_{Q_1} = \left\{ (\boldsymbol{t_{ij}}, \boldsymbol{y}^i, z_i) \;\middle|\; \begin{array}{l} \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ A\boldsymbol{y}^i - \boldsymbol{c} z_i \le \boldsymbol{0}, \\ \frac{1}{\beta_i} \le z_i \le \frac{1}{\alpha_i}, \\ -\boldsymbol{y}^i + \boldsymbol{l}^k z_i \le \boldsymbol{0}, \boldsymbol{y}^i - \boldsymbol{u}^k z_i \le \boldsymbol{0}, \\ \boldsymbol{t}_{ij} \le \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}^k}{\beta_i} z_j - \frac{\boldsymbol{l}^k}{\alpha_j \beta_i}, \\ \boldsymbol{t}_{ij} \le \frac{1}{\beta_j}\boldsymbol{y}^i + \frac{\boldsymbol{u}^k}{\alpha_i} z_j - \frac{\boldsymbol{u}^k}{\alpha_i \beta_j}, \\ \boldsymbol{t}_{ij} \ge \frac{1}{\alpha_j}\boldsymbol{y}^i + \frac{\boldsymbol{u}^k}{\alpha_i} z_j - \frac{\boldsymbol{u}^k}{\alpha_i \alpha_j}, \\ \boldsymbol{t}_{ij} \ge \frac{1}{\beta_j}\boldsymbol{y}^i + \frac{\boldsymbol{l}^k}{\beta_i} z_j - \frac{\boldsymbol{l}^k}{\beta_i \beta_j}, \\ \boldsymbol{t}_{ij} = \boldsymbol{t}_{ji}. \end{array} \right\} \tag{4.13}$$

Let vert_{M_1}, $\mathrm{vert}_{M_{Q_0}}$, and $\mathrm{vert}_{M_{Q_1}}$ denote the vertex set of M_1, M_{Q_0} and M_{Q_1}, respectively. Then we have

Proposition 4.7. *$\exists\, \boldsymbol{x}' \in vert_{M_1}$ such that $((\boldsymbol{y}')^i, (z')^i) \in vert_{M_{Q_0}}$, where $(\boldsymbol{y}')^i := (z')^i \boldsymbol{x}'$, $(z')^i := 1/(\boldsymbol{d}_i^\top \boldsymbol{x}' + b_i), i \in \{1, 2, \ldots, p\}$.*

Proof. Without loss of generality, let $\boldsymbol{x}' \in \mathrm{vert}_{M_1}$ such that $A\boldsymbol{x}' = \boldsymbol{c}$ and $\alpha_i = \min\left\{ \boldsymbol{d}_i^\top \boldsymbol{x}' + b_i \,\middle|\, \boldsymbol{x}' \in M_1 \right\}$. Then $A(\boldsymbol{y}')^i - \boldsymbol{c}(z')^i = \frac{1}{\alpha_i}(A\boldsymbol{x}' - \boldsymbol{c}) = \boldsymbol{0}$, $(z')^i = \frac{1}{\alpha_i}$, and $(z')^i \boldsymbol{l}^k \le (\boldsymbol{y}')^i \le (z')^i \boldsymbol{u}^k$ if $\boldsymbol{l}^k \le \boldsymbol{x}' \le \boldsymbol{u}^k$. Similarly, we can prove the equalities in (4.12) are strictly held if $\beta_i = \min\left\{ \boldsymbol{d}_i^\top \boldsymbol{x}' + b_i \,\middle|\, \boldsymbol{x}' \in M_1 \right\}$. Therefore, $((\boldsymbol{y}')^i, (z')^i) \in \mathrm{vert}_{M_{Q_0}}$, which completes the proof. □

Proposition 4.8. *$\exists\, ((\boldsymbol{y}')^i, (z')^i) \in vert_{M_{Q_0}}$ such that $((\boldsymbol{t}_{ij})', (\boldsymbol{y}')^i, (z')^i) \in vert_{M_{Q_1}}, i \in \{1, 2, \ldots, p\}$.*

Proof. Let $((\boldsymbol{y}')^i, (z')^i) \in \mathrm{vert}_{M_{Q_0}}$ such that

$$(z')^i = \frac{1}{\alpha_i} = \frac{1}{\min\left\{ \boldsymbol{d}_i^\top \boldsymbol{x}' + b_i \,\middle|\, \boldsymbol{x}' \in M_1 \right\}}, i = 1, 2, \ldots, p. \tag{4.14}$$

Then we will see that

$$\boldsymbol{t}_{ij} \le \frac{1}{\alpha_j}(\boldsymbol{y}')^i + \frac{\boldsymbol{l}^k}{\beta_i}(z')^j - \frac{\boldsymbol{l}^k}{\alpha_j \beta_i} = \frac{1}{\alpha_j}(\boldsymbol{y}')^i. \tag{4.15}$$

$$\boldsymbol{t}_{ij} \ge \frac{1}{\alpha_j}(\boldsymbol{y}')^i + \frac{\boldsymbol{u}^k}{\alpha_i}(z')^j - \frac{\boldsymbol{u}^k}{\alpha_i \alpha_j} = \frac{1}{\alpha_j}(\boldsymbol{y}')^i. \tag{4.16}$$

(4.15) and (4.16) imply that $\boldsymbol{t}_{ij} = \frac{1}{\alpha_j}(\boldsymbol{y}')^i$. Similarly, if we let

$$(z')^i = \frac{1}{\beta_i} = \frac{1}{\max\left\{\boldsymbol{d}_i^\top \boldsymbol{x}' + b_i \,\middle|\, \boldsymbol{x}' \in M_1\right\}}, i = 1, 2, \ldots, p. \tag{4.17}$$

Then

$$\boldsymbol{t}_{ij} \leq \frac{1}{\beta_j}(\boldsymbol{y}')^i + \frac{\boldsymbol{u}^k}{\alpha_i}(z')^j - \frac{\boldsymbol{u}^k}{\alpha_i\beta_j} = \frac{1}{\beta_j}(\boldsymbol{y}')^i. \tag{4.18}$$

$$\boldsymbol{t}_{ij} \geq \frac{1}{\beta_j}(\boldsymbol{y}')^i + \frac{\boldsymbol{l}^k}{\beta_i}(z')^j - \frac{\boldsymbol{l}^k}{\beta_i\beta_j} = \frac{1}{\beta_j}(\boldsymbol{y}')^i. \tag{4.19}$$

Therefore, $\boldsymbol{t}_{ij} = \frac{1}{\beta_j}(\boldsymbol{y}')^i$ according to (4.18) and (4.19). Thus, if we take $(z')^i$ satisfied (4.14) or (4.17), $((\boldsymbol{t}_{ij})', (\boldsymbol{y}')^i, (z')^i) \in \text{vert}_{M_{Q_1}}$, which is the desired proof. □

From the proof mentioned above, it is easy to see that

Proposition 4.9. *$\exists\, \boldsymbol{x}' \in vert_{M_1}$ such that $((\boldsymbol{t}_{ij})', (\boldsymbol{y}')^i, (z')^i) \in vert_{M_{Q_1}}$, where $(\boldsymbol{y}')^i := (z')^i\boldsymbol{x}'$, $(z')^i := \frac{1}{\boldsymbol{d}_i^\top \boldsymbol{x}' + b_i}, i \in \{1, 2, \ldots, p\}$.*

Proposition 4.9 implies that if the optimal solution of problem Q_1 with box $[\boldsymbol{l}^k, \boldsymbol{u}^k]$, denoted by $((\boldsymbol{t}_{ij}), (\boldsymbol{y}')^i, (z')^i), i, j = 1, \ldots, p$, such that

$$(z')^{i_*} = \min\left\{\frac{1}{\boldsymbol{d}_{i_*}^\top \boldsymbol{x}' + b_{i_*}}\right\},$$

or

$$(z')^{i_*} = \max\left\{\frac{1}{\boldsymbol{d}_{i_*}^\top \boldsymbol{x}' + b_{i_*}}\right\}.$$

then the optimal solution of problem (4.3) is on the vertex of M_1, where $i_* = \arg\min\{f(\frac{(\boldsymbol{y}')^i}{(z')^i}) | i = 1, \ldots, p\}$.

Furthermore, the number of iterations and branches showed in table 4.2, and table 4.3 are decreased sharply, which implies that we are able to improve the lower-bound quickly if we solve problem (4.8) recursively with branch and bound method. These features, together the proposition 4.9, give us an idea to develop another branch strategy.

4.4.2 Advanced Branching Rules

To globally solve problem (4.3), we can solve problem (4.8) instead by using branch and bound method. Instead of bisection branching rules, mentioned in section 4.3.1, we develop a more sophisticated branching strategies.

Let $((\boldsymbol{t}_{ij})^k, \boldsymbol{y}_i^k, z_i^k)$ be the optimal solution of problem Q_1 at iteration k with box $B^k = [\boldsymbol{l}^k, \boldsymbol{u}^k]$. Denote $\boldsymbol{x}^{i_k} = \boldsymbol{y}_i^k / z_i^k$ and $k^* = \arg\min \{f(\boldsymbol{x}^{i_k}) | i = 1, 2, \ldots, p\}$. It is clear that $\boldsymbol{x}^{i_k}$ is the optimal solution of problem (4.3) if $\boldsymbol{x}^{1_k} = \boldsymbol{x}^{2_k} = \cdots = \boldsymbol{x}^{p_k}$ for $\forall i = 1, 2, \ldots, p$. However, usually the relation $\boldsymbol{x}^{1_k} = \boldsymbol{x}^{2_k} = \cdots = \boldsymbol{x}^{p_k}$ does not hold. Therefore, we need to push it into next iteration.

Let

$$\varpi_j = \min \left\{ u_j^k - x_j^{i_k}, x_j^{i_k} - l_j^k \right\}, j = 1, 2, \ldots, n. \tag{4.20}$$

and

$$j_0 = \arg\min \{\varpi_j | j = 1, 2, \ldots, n.\} \tag{4.21}$$

Then the rectangle $B^k = [\boldsymbol{l}^k, \boldsymbol{u}^k]$ with the interior point $\boldsymbol{x}^{k^*}$ are to be divided into two sub-rectangles B^{2k}, B^{2k+1} by the following rules:

Ra Get $\varpi_j, j = 1, 2, \ldots, p$ by the rule of (4.20)

Rb Select j_0 by the rule of (4.21).

Rc Define $B^{2k} = \left\{ \boldsymbol{x} \in \mathbb{R}^n | l_j^k \le x_j \le u_j^k, j \ne j_0, x_{j_0}^{k^*} \le x_j \le u_j^k \right\}$,
$B^{2k+1} = \left\{ \boldsymbol{x} \in \mathbb{R}^n | l_j^k \le x_j \le u_j^k, j \ne j_0, l_j^k \le x_j \le x_{j_0}^{k^*} \right\}$.

The difference between bisection branching rules and advanced branching rules are showed by Figure 4.6 and Figure 4.7.

According to the branch strategy, clearly, we have
Remark: $x_{j_0}^{k^*} = l_{j_0}^{k+1}$ *or* $u_{j_0}^{k+1}$.

In addition, if we divide further for rectangle B^{2k}, B^{2k+1}, the following relation holds

$$\boldsymbol{l}^k \le \boldsymbol{l}^{k+1} \le \boldsymbol{x}^{(k+1)^*} \le \boldsymbol{u}^{k+1} \le \boldsymbol{u}^k, k = 1, 2, \ldots \tag{4.22}$$

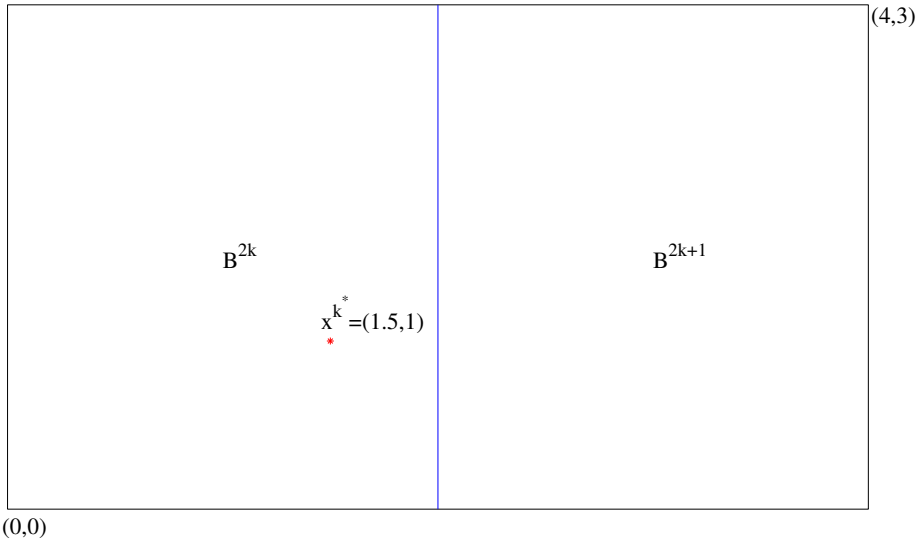

FIGURE 4.6: An example of bisection branching rules at iteration k with a current best solution x^{k^*} in box B^k

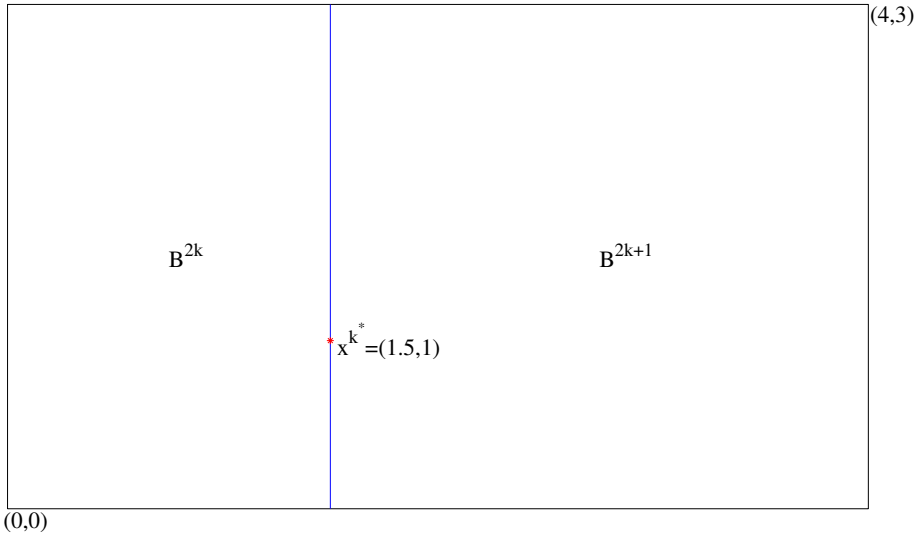

FIGURE 4.7: An example of advanced branching rules at iteration k with a current best solution x^{k^*} in box B^k

These relation mentioned above imply that

Lemma 4.10. *$\exists \tilde{\boldsymbol{l}}, \tilde{\boldsymbol{u}} \in [\boldsymbol{l}, \boldsymbol{u}]$ such that $\tilde{\boldsymbol{l}} \leq \tilde{\boldsymbol{u}}$, $\lim_{k\to\infty} \boldsymbol{l}^k = \tilde{\boldsymbol{l}}$ and $\lim_{k\to\infty} \boldsymbol{u}^k = \tilde{\boldsymbol{u}}$. Furthermore, the sequence $\{\boldsymbol{x}^{k^*}\}$ has accumulation points, and each of which lies on a corner of the limit rectangle $[\tilde{\boldsymbol{l}}, \tilde{\boldsymbol{u}}]$.*

Proof. For each element of $\boldsymbol{l}^k$, $\boldsymbol{u}^k$, denoted by l_j^k, u_j^k, both sequence $\{l_j^k\}$ and $\{u_j^k\}$ are monotonic, bounded followed by (4.22). Hence, $\{l_j^k\}$ and $\{u_j^k\}$ have limits denoted by $\tilde{l}_j$ and $\tilde{u}_j$, respectively. Of course, $\tilde{l}_j \leq \tilde{u}_j$.

Since $\boldsymbol{x}^{k^*}$ is generated in the compact set $[\boldsymbol{l}, \boldsymbol{u}]$, there exists at least one accumulation points. Without loss of generality, let $\hat{\boldsymbol{x}}^{k^*}$ be an arbitrary accumulation point and $\{\boldsymbol{x}^{k_s{}^*}\}$ be a subsequence converging to $\hat{\boldsymbol{x}}^{k^*}$. Since the set $\{1, 2, \ldots, n\}$ is finite, there $\exists t \in \{1, 2, \ldots, n\}$ such that $t = j^{k_s{}^*}$ when $s \to \infty$. Note that $x_t^{k^*} = \{l_t^{k+1}, u_t^{k+1}\}$, therefore we have $\lim\limits_{s\to\infty} x_t^{k_s{}^*} = \hat{x}_t^{k^*} = \{\tilde{l}_j, \tilde{u}_j\}$. Since $x_t^{k_s{}^*}$ are satisfied (4.20) and (4.21), we have

$$\min\left\{u_t^{k_s^*} - x_t^{k_s{}^*}, x_t^{k_s{}^*} - l_t^{k_s^*}\right\} \geq \min\left\{u_j^{k_s^*} - x_j^{k_s{}^*}, x_j^{k_s{}^*} - l_j^{k_s^*}\right\}, j = 1, \ldots, n. \quad (4.23)$$

Since $\min\left\{u_t^{k_s^*} - x_t^{k_s{}^*}, x_t^{k_s{}^*} - l_t^{k_s^*}\right\} = 0$, then $\min\left\{u_j^{k_s^*} - x_j^{k_s{}^*}, x_j^{k_s{}^*} - l_j^{k_s^*}\right\} = 0, j = 1, \ldots, n$, which implies that $\{\hat{\boldsymbol{x}}^{k^*}\}$ lies on a corner of rectangle $[\tilde{\boldsymbol{l}}, \tilde{\boldsymbol{u}}]$. □

Lemma 4.11. *Let $\hat{\boldsymbol{x}}^{k^*}$ be an arbitrary accumulation point of $\{\boldsymbol{x}^{k^*}\}$, and $\{\boldsymbol{x}^{k_s{}^*}\}$ be a subsequence converging to $\hat{\boldsymbol{x}}^{k^*}$. Then we have*

$$\lim_{s\to\infty} \boldsymbol{x}_i^{k_s} = \hat{\boldsymbol{x}}^{k^*}, \boldsymbol{x}_i = \frac{\boldsymbol{y}_i}{z_i}, i = 1, 2, \ldots, p. \quad (4.24)$$

Proof. From lemma 4.10, we know that $\exists\, j$ such that $\hat{x}_j^{k^*} = \{\tilde{l}_j, \tilde{u}_j\}$, for $j = 1, 2, \ldots, n$. Clearly, $l_j^{k_s} \leq x_{ij}{}^{k_s}$ for $i = 1, 2, \ldots, p$, which implies that $\boldsymbol{x}_{ij}{}^{k_s} \to l_j^{k_s}$ if $\hat{x}_j^{k^*} = \tilde{l}_j$. In the similar way, we can prove $x_{ij}{}^{k_s} \to u_j^{k_s}$ if $\hat{x}_j^{k^*} = \tilde{u}_j$. Therefore (4.24) holds. □

Lemma 4.11 indicates that if we can find an arbitrary accumulation point, denoted by $\hat{\boldsymbol{x}}^{k^*}$, then $\hat{\boldsymbol{x}}^{k^*}$ is also an optimal solution of problem (4.3). Furthermore, lemma 4.10 and lemma 4.11 keep the advanced branch rules convergent.

4.4.3 Advanced Algorithm Description

As it described in section 4.3.2, we starting from $B^1 = \{[\boldsymbol{l}^1, \boldsymbol{u}^1] = [\boldsymbol{l}, \boldsymbol{u}]\}$ that is generated by solving problem (4.1) and (4.2), we solve $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ for $k = 1, 2, \ldots$. At each iteration $k = 1, 2, \ldots$, we will get an optimal value of $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ at $(\boldsymbol{t}^{ij_k}, \boldsymbol{y}^{i_k}, z_{i_k})$, $i, j = 1, 2, \ldots, p$. And let $\boldsymbol{x}^{k^*} = \boldsymbol{y}^{k^*}/z_{k^*}$, $k^* \in \arg\min\{f(\boldsymbol{x}^{i_k}), i_k =$

$1, 2, \ldots, p\}$. Then the rectangle B^k that the smallest value of $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ take will be divided into two rectangles B^{2k}, B^{2k+1} by the advanced branching rules. And rectangle B^k will be discarded at the k-th iteration if the optimal value of $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ is greater than the current best vale. Now the algorithm is ready to be described in detail as follows:

Algorithm 3 Advanced Branch and bound algorithm for SOLR based on bisection rule

1: Initial settings: Set $k = 1$. Let $\mathcal{B}^k = \{[\boldsymbol{l}^k, \boldsymbol{u}^k]\} = \{[\boldsymbol{l}, \boldsymbol{u}]\}$ be the initial rectangle. $L = -\infty$, $U = \infty$.
2: Solve $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ with $B^j \in \mathcal{B}^k$. If $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ is feasible then obtain the optimal value L^k and $\boldsymbol{x}^{i_k} = \boldsymbol{y}^{i_k}/z_{i_k}$, and $U^k = \min\{f(\boldsymbol{x}^{i_k}), i_k = 1, \ldots, p\}$. $\mathcal{L}^k = L^k, U = U^k$. If $Q_1(\boldsymbol{l}^k, \boldsymbol{u}^k)$ is infeasible then terminate.
3: Set a tolerance $\varepsilon \geq 0$.
4: **while** $U - \mathcal{L}^k > \varepsilon$ **do**
5: Discard rectangles $B^j \in \mathcal{B}^k$ such that the value of $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j) > U^k$
6: Select $B^{j_0} \in \mathcal{B}^k$ with $L^{j_0} = \mathcal{L}^k$.
7: Divide B^{j_0} into two subrectangles B^{2k}, B^{2k+1} according to the advanced branching rules *Ra-Rc*, and update $\mathcal{B}^k = (\mathcal{B}^k \setminus \{B^{j_0}\}) \cup \{B^{2k}\} \cup \{B^{2k+1}\}$
8: Solve $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ for $j = k, 2k+1$. If $Q_1(\boldsymbol{l}^j, \boldsymbol{u}^j)$ is not feasible, then discard B^j. Otherwise, obtain $L^k, U^k, L^{2k+1}, U^{2k+1}$ and keep the corresponding incumbent best solution $\boldsymbol{x}^{j_k} = \boldsymbol{y}^{j_k}/z_{j_k}$.
9: Update $\mathcal{L}^k = \min\{L^k, L^{2k+1}\}$ if $\mathcal{L}^k < \min\{L^k, L^{2k+1}\}$, $U = \min\{U^k, U^{2k+1}\}$ if $U > \min\{U^k, U^{2k+1}\}$ and update the incumbent best solution.
10: Set $k = k + 1$
11: **end while**
12: Return U as an ε-minimum of (P_0) with a minimizer $\boldsymbol{x}^{j_k} = \boldsymbol{y}^{j_k}/z_{j_k} \in [\boldsymbol{l}, \boldsymbol{u}]$.

It is easy to see that the Algorithm 3 are the same as the Algorithm 2 except the branch rules, and the branch rules used in Algorithm 3 are proven to be convergent. Therefore, for a given $\varepsilon > 0$, Algorithm 3 can terminate in finitely many iterations; If the algorithm can not terminate for a given $\varepsilon = 0$, any accumulation point of the sequence $\{\boldsymbol{x}^{k^*} | k = 1, 2 \ldots\}$ is an optimal solution followed by the proposition 4.3 and lemma 4.11.

4.4.4 Numerical Experiments

In this section, in order to compare the performance of the two different branch strategies used in branch and bound for solving the SOLR problem with lower

dimension, we implemented some numerical experiments using randomly generated data sets. The algorithm is coded in MATLAB® and implemented on an iMac with a quad-core Core(i5) and 8GB of RAM in Muroran Institute of Technology.

The datasets of the test problems used in this study are set as follows: The dimension of variables is 3 and the tolerance $\varepsilon = 0.05$. The coefficients $\boldsymbol{n}_i, \boldsymbol{d}_i$ are *i.i.d.* generated in the ranges of $-5 \leq n_{ij}, d_{ij} \leq 5$, for $i = 1, \ldots, p$ and $j = 1, 2, 3$. The constants a_i $(i = 1, \ldots, p)$ are randomly chosen from $[0, 60]$ and b_i $(i = 1, 2, \ldots, p)$ are fixed to 60.

Problem (P_0) was solved for various $p = 5, 10, 20, 30, 40, 50, 60, 70, 80$. For a fixed p, a set of 15 instances of the problem was solved with model Q_1 using bisection rules and advanced rules, respectively. The column titled MIN., AVERG., MAX. provide the information on the minimal The recorded minimal CPU times in second, average CPU times in second, maximal CPU times in second are provided in Tabel 4.4 of the column titled "MIN.", "AVERG." and "MAX.", respectively. And the related information on the number of iterations and the number of branches in the exception are showed Table 4.5 and Table 4.6, respectively. The rows $\mathcal{Q}_{1b}$ delivers the results that obtained by solving model Q_1 with bisection rules, while the rows $\mathcal{Q}_{1a}$ delivers the results that obtained by solving model Q_1 with advanced rules.

Tabel 4.4, Table 4.5 and Table 4.6 indicate that

- The average number of branches is decreased about 4% of bisection branching rules when $p \geq 40$. Usually, small number of branches will result in less computational time.

- The average CPU time solved by branch and bound method using advanced branch rules is less than it solved by bisection branching rules when $p \geq 30$. And the decreased CPU time will be larger with p growing. That is because the average number of branches becomes smaller when $p \geq 30$ and the computational time for solving model Q_1 will be more with the p growing. Furthermore, the minimal CPU time, iterations and branches for advanced branch rules will be no more than those from bisection rules, which implies that the advanced branching rules will result in less number of branches for

those problem that can be solved in a few number of branches when we use bisection branching rules (e.g. the number of branches is less than 5).

- For a smaller p (e.g. $p \leq 5$), the average CPU time and number of branches is larger for advanced branching rules compared those from bisection branching rules. That is because the number of branches will become larger for advanced branch rules compared the bisection branching rules, and the computational time for solving model Q_1 once is tiny.

- For some cases, the maximal number of iterations for advanced branching rules is larger compared the bisection branching rules. The reason is based on twofold. First, when the optimal solution obtained within a given tolerance ε at last iteration, the selected rectangle will be divided into two sub-rectangles at the previous iteration. For the bisection branching rules, one of them may be discarded when the lower bound of the one is greater than the current best value. The second reason is related to the advanced branching rules mentioned in section 4.4.2. For the advanced branching rules, since the current best solution will be on the edge that the both two sub-rectangles share if the algorithm goes to the next iteration, both of the two sub-rectangles will not be discarded if the optimal solution is on the side that is belonged to the both rectangle. Therefore, though the number of branches will be larger, the CPU time will not be increased because there is no more computational time for other branches. However, one of them can be discarded if we using the bisection branching rules.

4.5 Conclusions

In this chapter, we have made a new linear relaxation for the SOLR problem and designed a branch and bound algorithm for solving the SOLR problem with lower dimension using two branching rules. One is the standard bisection via longest edge, and the other is the advanced branching rules. The proposed two algorithms share the similarities to the previous algorithm [16] that its branching process works on a space with dimensions n, the dimensions of native variables while its bounding process works on a space with dimensions of $(p^2n + pn + p)$, p

Model	# p	CPU time(s)		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	0.1	0.3	0.6
$\mathcal{Q}_{1a}$	5	0.1	0.42	0.66
$\mathcal{Q}_{1b}$	10	0.4	0.8	2.0
$\mathcal{Q}_{1a}$	10	0.4	0.8	1.8
$\mathcal{Q}_{1b}$	20	2.7	8.7	14.4
$\mathcal{Q}_{1a}$	20	2.4	8.7	19.4
$\mathcal{Q}_{1b}$	30	14.3	34.2	54.6
$\mathcal{Q}_{1a}$	30	15.3	30.5	45.5
$\mathcal{Q}_{1b}$	40	49.4	111.3	223.9
$\mathcal{Q}_{1a}$	40	44.7	98.8	215.3
$\mathcal{Q}_{1b}$	50	95.0	242.5	500.9
$\mathcal{Q}_{1a}$	50	92.4	204.3	504.2
$\mathcal{Q}_{1b}$	60	296.3	650.7	1782.9
$\mathcal{Q}_{1a}$	60	192.4	567.9	1689.2
$\mathcal{Q}_{1b}$	70	469.3	1879.4	7502.5
$\mathcal{Q}_{1a}$	70	303.1	1770.7	6984.3
$\mathcal{Q}_{1b}$	80	1182.3	2177.5	6161.9
$\mathcal{Q}_{1a}$	80	863.4	1937.5	5786.0

TABLE 4.4: Numerical results on CPU times for solving problem (P_0) with various p and $n = 3$. The rows $\mathcal{Q}_{b1}$ for the results obtained from model Q_1 using bisection branching rules, while rows $\mathcal{Q}_{a1}$ for the results obtained from model Q_1 using advanced branching rules.

is the number of terms of ratios. Theoretically, the proposed algorithms finds an ε-minimizer for any pre-given $\varepsilon > 0$ within finitely many iterations.

We conducted the numerical experiments to investigate the behavior of the proposed algorithm using bisection branching rules. The results obtained from the numerical experiments indicate that the proposed algorithm is superior to the previous algorithm in CPU time, number of iterations and numbers of branches. The numerical experiments shows that the CPU time is at most about 7% of the algorithm in [16] on average.

Since the proposed algorithm using bisection rules can find an optimal solution within a given tolerance efficiently, thus the relaxed model can enforce a strong approximation for the quadratic constraints of the original problem. Thus we developed another branch and bound algorithm for solving the SOLR problem. The advanced branching rules are proven to make the proposed algorithm convergent. And the numerical results show that the average CPU time solved by

Model	# p	# Iterations		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	1	1.8	4
$\mathcal{Q}_{1a}$	5	1	1.9	4
$\mathcal{Q}_{1b}$	10	1	2.2	6
$\mathcal{Q}_{1a}$	10	1	2.1	5
$\mathcal{Q}_{1b}$	20	2	4.8	12
$\mathcal{Q}_{1a}$	20	2	4.7	14
$\mathcal{Q}_{1b}$	30	3	7.4	12
$\mathcal{Q}_{1a}$	30	3	6.4	10
$\mathcal{Q}_{1b}$	40	4	9.4	19
$\mathcal{Q}_{1a}$	40	4	8.1	15
$\mathcal{Q}_{1b}$	50	3	14.5	17
$\mathcal{Q}_{1a}$	50	2	11.9	18
$\mathcal{Q}_{1b}$	60	1	8.8	27
$\mathcal{Q}_{1a}$	60	1	7.6	25
$\mathcal{Q}_{1b}$	70	3	10.7	21
$\mathcal{Q}_{1a}$	70	3	9.6	21
$\mathcal{Q}_{1b}$	80	5	8.8	25
$\mathcal{Q}_{1a}$	80	4	7.8	24

TABLE 4.5: Numerical results on iterations for solving problem (P_0) with Q_1 and Q_0 with various p and $n = 3$. The rows $\mathcal{Q}_{b1}$ for the results obtained from model Q_1 using bisection branching rules, while rows $\mathcal{Q}_{a1}$ for the results obtained from model Q_1 using advanced branching rules.

branch and bound method using advanced branch rules is less than it solved by bisection branching rules when $p \geq 30$. Furthermore, the reduction of CPU time will be substantial with the p growing.

We plan to conduct larger scale experiments to look into the detailed behavior of the new algorithm as a further work. It is fundamentally important to make a theoretical analysis on the number of iterations in which the new algorithm finds a minimizer. An error bound between a minimizer and an ε-minimizer is an important feature. We leave them as our further work.

Model	# p	# Branches		
		MIN.	AVERG.	MAX.
$\mathcal{Q}_{1b}$	5	1	2.1	5
$\mathcal{Q}_{1a}$	5	1	3.2	10
$\mathcal{Q}_{1b}$	10	1	3.9	10
$\mathcal{Q}_{1a}$	10	1	4.0	8
$\mathcal{Q}_{1b}$	20	3	8.3	24
$\mathcal{Q}_{1a}$	20	3	7.9	23
$\mathcal{Q}_{1b}$	30	5	12.6	23
$\mathcal{Q}_{1a}$	30	5	12.1	19
$\mathcal{Q}_{1b}$	40	7	16.5	34
$\mathcal{Q}_{1a}$	40	5	15.3	38
$\mathcal{Q}_{1b}$	50	5	25.9	32
$\mathcal{Q}_{1a}$	50	4	24.1	34
$\mathcal{Q}_{1b}$	60	2	15.8	46
$\mathcal{Q}_{1a}$	60	1	15.2	48
$\mathcal{Q}_{1b}$	70	5	16.9	27
$\mathcal{Q}_{1a}$	70	4	14.6	24
$\mathcal{Q}_{1b}$	80	7	16.3	48
$\mathcal{Q}_{1a}$	80	6	14.9	43

TABLE 4.6: Numerical results on branches for solving problem (P_0) with Q_1 and Q_0 with various p and $n = 3$. The rows $\mathcal{Q}_{b1}$ for the results obtained from model Q_1 using bisection branching rules, while rows $\mathcal{Q}_{a1}$ for the results obtained from model Q_1 using advanced branching rules.

Chapter 5

Convex Relaxations for the SOLR Problem

In this chapter, we study several relaxations for minimizing the SOLR problem constrained a set of linear constrains. For convince, we rewrite the SOLR problem which is mentioned in Chapter 1 as follows.

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \\ \text{subject to} \quad & A\boldsymbol{x} \leq \boldsymbol{c}, \\ & \boldsymbol{x} \geq \boldsymbol{0}, \end{aligned} \tag{5.1}$$

where $p \geq 2$, $\boldsymbol{n}_i, \boldsymbol{d}_i$ are both vector in $\mathbb{R}^n$ for $i = 1, \ldots, p$, a_i, b_i are real numbers for $i = 1, \ldots, p$, A is an $m \times n$ matrix, $\boldsymbol{c}$ is a vector in $\mathbb{R}^m$. We assume each denominator is positive on its feasible region without loss of generality by proposition 4.1.

Our motivation for studying problem (5.1) using convex relaxation are as follows. It is well know that problem (5.1) has multiple local optimizers which are not general globally optimal. We are interested in globally solving problem (5.1). And if problem (5.1) is too difficult to be solved globally (e.g. because of ill-condition and /or a large number of variables or ratios in $f(\boldsymbol{x})$), we would like to get at least a valid lower bound of the global minimum. The upper bound can be obtained with some efficient methods (algorithms) such as Newton's method or its variants, the DIRECT algorithm mentioned in Chapter 3. Furthermore, the SOLR problem

can be transformed into a quadratic programming by introducing some auxiliary variables. Then semidefinite relaxation, a powerful and computationally efficient approximation technique for several difficult nonconvex quadratical problem, can be applied to solve that kind of problem.

5.1 Reformulation

In this section, we will describe how to reformulate problem (5.1) into an equivalent problem which is able to be relaxed as an SDP.

Let $r_i = \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i}, i = 1, \ldots, p$. Define

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{p} r_i \\ \text{subject to} \quad & \frac{\boldsymbol{n}_i^\top \boldsymbol{x} + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} + b_i} \leq r_i, i = 1, \ldots, p, \\ & A\boldsymbol{x} \leq \boldsymbol{c}, \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned} \tag{5.2}$$

The following theorem asserts that problem (5.1) is equivalent to problem (5.2).

Theorem 5.1. *Problem* (5.1) *is equivalent to problem* (5.2).

Proof. Let $\boldsymbol{x}^*$ be an optimal solution to problem (5.1). Define $r_i^* := \frac{\boldsymbol{n}_i^\top \boldsymbol{x}^* + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}^* + b_i}$. Then it is easy to see that $(\boldsymbol{x}^*, r_1^*, r_2^*, \ldots, r_p^*)$ is feasible for problem (5.2). Suppose there is another feasible solution $(\boldsymbol{x}', r_1', r_2', \ldots, r_p')$ to problem (5.2) such that

$$\sum_{i=1}^{p} r_i' < \sum_{i=1}^{p} r_i^*. \tag{5.3}$$

Since

$$\sum_{i=1}^{p} r_i' = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x}' + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}' + b_i}, \sum_{i=1}^{p} r_i^* = \sum_{i=1}^{p} \frac{\boldsymbol{n}_i^\top \boldsymbol{x} * + a_i}{\boldsymbol{d}_i^\top \boldsymbol{x} * + b_i}, \tag{5.4}$$

and $\boldsymbol{x}'$ is also a feasible solution to problem (5.1). Therefore, the inequality (5.3) contradicts the optimality of $\boldsymbol{x}^*$ for problem (5.1). And similar to vice versa. □

Since $\boldsymbol{d}_i^\top \boldsymbol{x} + b_i > 0$, $\frac{\boldsymbol{n}_i^\top \boldsymbol{x}+a_i}{\boldsymbol{d}_i^\top \boldsymbol{x}+b_i} \leq r_i, \forall i = 1, \ldots, p$ is equivalent to $\boldsymbol{n}_i^\top \boldsymbol{x} + a_i - r_i(\boldsymbol{d}_i^\top \boldsymbol{x} + b_i) \leq 0, \forall i = 1, \ldots, p$. Therefore, problem (5.2) can be rewritten as follows

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{p} r_i \\ \text{subject to} \quad & \boldsymbol{n}_i^\top \boldsymbol{x} + a_i - r_i(\boldsymbol{d}_i^\top \boldsymbol{x} + b_i) \leq 0, i = 1, \ldots, p, \\ & A\boldsymbol{x} \leq \boldsymbol{c}, \\ & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned} \tag{5.5}$$

Let $\boldsymbol{y} = (x_1, \ldots, x_n, r_1, \ldots, r_p)^\top$ and

$$P_i = \begin{pmatrix} 0 & \cdots & 0 & 0 & \cdots & -\boldsymbol{d_i}/2 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & \vdots & \cdots & 0 \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ -\boldsymbol{d_i}^\top/2 & \cdots & \cdots & 0 & \cdots & \vdots & \cdots & 0 \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \end{pmatrix},$$

then problem (5.5) can be formulated as the following form

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{q_0}^\top \boldsymbol{y} \\ \text{subject to} \quad & \boldsymbol{y}^\top P_i \boldsymbol{y} + \boldsymbol{q_i}^\top \boldsymbol{y} \leq -a_i,\ i = 1, \ldots, p, \\ & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j,\ j = 1, 2, \ldots, m, \\ & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \end{aligned} \tag{5.6}$$

where $\boldsymbol{q_0} = (\boldsymbol{0}^{1\times n}, \boldsymbol{1}^{1\times p})^\top$; $\boldsymbol{y} = (\boldsymbol{x}; \boldsymbol{r})^\top$; $\boldsymbol{q}_i = (\boldsymbol{n}_i; -b_i\boldsymbol{e}_i)^\top, i = 1, \ldots, p$; $\boldsymbol{\mu}_j = (A_{j\bullet}; \boldsymbol{0}^{p\times 1})^\top, j = 1, \ldots, m$, $A_{j\bullet}$ is the vector of j-th row. $\boldsymbol{e}_i$ is the i-th standard unit vector.

Because the feasible region for $\boldsymbol{x}$ is convex and nonempty, we can construct a rectangle $B_x = [\boldsymbol{l}_x, \boldsymbol{u}_x]$ by solving problem (4.1) and problem (4.2) in Chapter 3. And for a single linear ratio, it can be equivalently solved by LP when using the Charnes-Cooper transformation [19]. To construct a rectangle $B_r = [\boldsymbol{l}_r, \boldsymbol{u}_r]$ for $\boldsymbol{r}$,

we can solve the following problem (5.7) and problem (5.8) easily.

$$\begin{aligned} l_{ri} = \quad & \text{minimize} \quad & & \left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i\right) \\ & \text{subject to} \quad & & \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ & & & A\boldsymbol{y}^i - \boldsymbol{c} z_i \leq \boldsymbol{0}, \\ & & & \frac{1}{\beta_i} \leq z_i \leq \frac{1}{\alpha_i}, \\ & & & z_i \boldsymbol{l}_x \leq \boldsymbol{y^i} \leq z_i \boldsymbol{u}_x. \end{aligned} \tag{5.7}$$

$$\begin{aligned} u_{ri} = \quad & \text{maximize} \quad & & \left(\boldsymbol{n}_i^\top \boldsymbol{y}^i + a_i z_i\right) \\ & \text{subject to} \quad & & \boldsymbol{d}_i^\top \boldsymbol{y}^i + b_i z_i = 1, \\ & & & A\boldsymbol{y}^i - \boldsymbol{c} z_i \leq \boldsymbol{0}, \\ & & & \frac{1}{\beta_i} \leq z_i \leq \frac{1}{\alpha_i}, \\ & & & z_i \boldsymbol{l}_x \leq \boldsymbol{y^i} \leq z_i \boldsymbol{u}_x, \end{aligned} \tag{5.8}$$

where l_{ri}, u_{ri} are the i-th member of $\boldsymbol{l}_r, \boldsymbol{u}_r$, respectively. Therefore, the rectangle $[\boldsymbol{l}_B, \boldsymbol{u}_B]$ can be constructed by the following formula

$$[\boldsymbol{l}_B, \boldsymbol{u}_B] = \left[\begin{pmatrix} \boldsymbol{l}_x \\ \boldsymbol{l}_r \end{pmatrix}, \begin{pmatrix} \boldsymbol{u}_x \\ \boldsymbol{u}_r \end{pmatrix} \right]. \tag{5.9}$$

Clearly, problem (5.6), an $\mathcal{NP}$-hard [84] problem, is a quadratic programming with linear objective and quadratic and linear constraints, and the development of suitable relaxation is required for problem (5.1). SDP techniques have received a great deal of attention in optimization literature [86], and several SDP relaxations have been proposed for solving that kind of prolem [3, 4, 11, 12, 14, 35, 77].

5.2 Lagrangian Relaxation

Lagrangian relaxation, an important technique for constrained optimization problems, has been applied for semidefinite relaxations for quadratic programming problems [77, 85].

The Lagrangian function of problem (5.6) is

$$\begin{aligned} L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) &= \boldsymbol{q_0}^\top \boldsymbol{y} + \sum_{i=1}^{p} \lambda_i (\boldsymbol{y}^\top P_i \boldsymbol{y} + \boldsymbol{q_i}^\top \boldsymbol{y} + a_i) + \sum_{j=1}^{m} \upsilon_j (\boldsymbol{\mu}_j^\top \boldsymbol{y} - c_j) \\ &\quad -\boldsymbol{\omega}^\top (\boldsymbol{y} - \boldsymbol{l}) + \boldsymbol{\nu}^\top (\boldsymbol{y} - \boldsymbol{u}) \\ &= \sum_{i=1}^{p} \lambda_i \boldsymbol{y}^\top P_i \boldsymbol{y} + \left(\boldsymbol{q_0} + \sum_{i=1}^{p} \lambda_i \boldsymbol{q}_i + \sum_{j=1}^{m} \upsilon_j \boldsymbol{\mu}_j - \boldsymbol{\omega} + \boldsymbol{\nu} \right)^\top \boldsymbol{y} \\ &\quad + \left(\sum_{i=1}^{p} \lambda_i a_i - \sum_{j=1}^{m} \upsilon_j c_j + \boldsymbol{\omega}^\top \boldsymbol{l} - \boldsymbol{\nu}^\top \boldsymbol{u} \right), \end{aligned} \tag{5.10}$$

where $\boldsymbol{\lambda} \in \mathbb{R}_+^p$, $\boldsymbol{\upsilon} \in \mathbb{R}_+^m$, and $\boldsymbol{\omega}, \boldsymbol{\nu} \in \mathbb{R}_+^n$ are multipliers. The Lagrange dual problem of problem (5.6) is

$$\begin{aligned} f^L = \quad & \underset{\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}}{\text{maximize}} \quad \underset{\boldsymbol{y}}{\text{minimize}}\ L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) \\ & \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \boldsymbol{\upsilon} \geq \boldsymbol{0}, \boldsymbol{\omega} \geq \boldsymbol{0}, \boldsymbol{\nu} \geq \boldsymbol{0}. \end{aligned}$$

By weak duality, let f^* is the optimal value of problem (5.1) and we have

Proposition 5.2. $f^l \leq f^*$.

Furthermore, any feasible solution $(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\upsilon}}, \bar{\boldsymbol{\omega}}, \bar{\boldsymbol{\nu}})$ yields a lower bound for f^*. In addition, since $\sum_{i=1}^{p} \lambda_i P_i \not\succeq 0$ because of $\text{diag}(P_i) = \boldsymbol{0}, i = 1, \ldots, p$, which will make $\min_{\boldsymbol{y}} L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu})$ go to negative infinity. Since the lagrange dual problem essentially ignores linear constraints, which will lead to weak relaxations if ally lagrangian duality to primal problems that contain explicit linear equality constraints and bound constraints. An alternative approach it only to dualize the nonlinear constraints presented in [82]. The resulting Lagrange dual problem for problem (5.6) is as follows

$$\begin{aligned} f_0^L = \quad & \underset{\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}}{\text{maximize}} \quad \underset{\boldsymbol{y}}{\text{minimize}}\ L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{0},\boldsymbol{0},\boldsymbol{0}) \\ & \text{subject to} \quad \boldsymbol{\lambda} \geq \boldsymbol{0}, \\ & \qquad\qquad \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & \qquad\qquad \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}. \end{aligned} \tag{5.11}$$

Clearly, we have

Proposition 5.3.

$$f^L \leq f_0^L \leq f^*.$$

5.3 Shor Relaxation

A reformulation of problem (5.11) is

$$\begin{aligned} f^L = \underset{\varepsilon,\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}}{\text{maximize}} \quad & \varepsilon \\ \text{subject to} \quad & L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) - \varepsilon \geq 0, \forall \boldsymbol{y} \in \mathbb{R}^{n+p}, \\ & \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\upsilon} \geq \mathbf{0}, \boldsymbol{\omega} \geq \mathbf{0}, \boldsymbol{\nu} \geq \mathbf{0}. \end{aligned} \tag{5.12}$$

Let $A(\varepsilon,\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) \in \mathcal{S}^{n+1}$ be

$$\left(\begin{array}{c|c} \sum_{i=1}^{p}\lambda_i a_i - \sum_{j=1}^{m}\upsilon_j c_j + \boldsymbol{\omega}^\top \boldsymbol{l} - \boldsymbol{\nu}^\top \boldsymbol{u} - \varepsilon & \frac{1}{2}\left(\boldsymbol{q_0} + \sum_{i=1}^{p}\boldsymbol{\lambda}_i \boldsymbol{q}_i + \sum_{j=1}^{m}\boldsymbol{\mu}_j - \boldsymbol{\omega} + \boldsymbol{\nu}\right)^\top \\ \hline \cdots & \sum_{i=1}^{p}\lambda_i P_i \end{array}\right),$$

and we have

$$L(\boldsymbol{y},\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) - \varepsilon = \begin{pmatrix} 1 \\ \boldsymbol{y} \end{pmatrix}^\top A(\varepsilon,\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) \begin{pmatrix} 1 \\ \boldsymbol{y} \end{pmatrix}.$$

The problem (5.12) can be rewritten as the following SDP problem

$$\begin{aligned} f^{Dshor} = \underset{\varepsilon,\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}}{\text{maximize}} \quad & \varepsilon \\ \text{subject to} \quad & A(\varepsilon,\boldsymbol{\lambda},\boldsymbol{\upsilon},\boldsymbol{\omega},\boldsymbol{\nu}) \succeq 0, \\ & \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\upsilon} \geq \mathbf{0}, \boldsymbol{\omega} \geq \mathbf{0}, \boldsymbol{\nu} \geq \mathbf{0}, \end{aligned} \tag{5.13}$$

whose dual is equivalent to the following problem

$$\begin{array}{ll} \text{minimize} & \boldsymbol{q_0}^\top \boldsymbol{y} \\ \text{subject to} & P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} \leq -a_i, i = 1, \ldots, p, \\ & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succeq 0, \\ & Y = \boldsymbol{y}\boldsymbol{y}^\top. \end{array} \tag{5.14}$$

By dropping the constraint $Y = \boldsymbol{y}\boldsymbol{y}^\top$, we obtain the following form due to Shor relaxation [78]:

$$\begin{array}{lll} f^{Shor} = & \text{minimize} & \boldsymbol{q_0}^\top \boldsymbol{y} \\ & \text{subject to} & P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} \leq -a_i, i = 1, \ldots, p, \\ & & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succeq 0. \end{array} \tag{5.15}$$

Since problem (5.13) is a reformulation of SDP form of problem (5.12), and problem (5.15) is a dual relaxation of problem (5.13), we have the following proposition based on weak duality

Proposition 5.4.

$$f^{Dshor} = f^L \leq f^{Shor} \leq f^*.$$

If the following condition are satisfied

Condition: *The problem* (5.13) *and problem* (5.15) *are feasible, and the set*

$$\mathcal{Y} = \left\{ (\boldsymbol{y}, Y) \;\middle|\; \begin{array}{l} P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} < -a_i, i = 1, \ldots, p, \\ \boldsymbol{\mu}_j^\top \boldsymbol{y} < c_j, j = 1, 2, \ldots, m, \\ \boldsymbol{l} < \boldsymbol{y} < \boldsymbol{u}, \\ \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succ 0. \end{array} \right\}$$

is no empty.

According to the conic duality in [8], we have

Proposition 5.5.

$$f^L = f^{Shor} \leq f^*.$$

5.4 SDP Relaxation for the SOLR Problem

In this section, we first give some definitions of general multilinear and bilinear function which can be viewed as a subclass of quadratic function. And we apply an SDP relaxation for a bilinear programming based on a convex envelop of bilinear functions.

First we give some definitions: let M be a compact polytope $M \subset R^n$, and vert$M$ denote all vertices of polytope M. And if φ is a real-valued function $\varphi : \boldsymbol{x} \to R^n$, epi$(\varphi)$ denotes its epigraph : epi$(\varphi) = \{(z, \boldsymbol{x})|z \geq \varphi(\boldsymbol{x}), \boldsymbol{x} \in \text{dom}(\varphi)\}$, where dom$(\varphi) = \{\boldsymbol{x}|\varphi(\boldsymbol{x}) < \infty\}$.

Definition 5.6. Function $g(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k)$ is said to be a general multilinear function if for each $i = 1, 2, \ldots, k$, function $g({\boldsymbol{x}_1}^0, {\boldsymbol{x}_2}^0, \ldots, \boldsymbol{x}_i, \ldots, {\boldsymbol{x}_k}^0)$ linearly depends on vector $\boldsymbol{x}_i$ in the case that all other $k-1$ vector arguments are fixed.

A function $g(\boldsymbol{x}_1, \boldsymbol{x}_2)$ is called bilinear if it reduces to a linear one by fixing the vector $\boldsymbol{x}_1$ or $\boldsymbol{x}_2$ to a particular value. It is easy to see that bilinear functions compose a subclass of multilinear functions. Of course, it is also a special case of a quadratic function. We refer to optimization problems with bilinear objective and /or constraints as bilinear problems, and they can be viewed as a subclass of quadratic programming in the case of the diag$(P_i) = 0, i = 1, \ldots, p$ in problem (5.6).

Definition 5.7. Let $g(\boldsymbol{x})$ be a real valued lower semicontinuous function, defined a convex set M, dom$(g) = M$. Set $X(g)$ is said to be a generating set of this function, if

$$X(g) = \{\boldsymbol{x}|(\boldsymbol{x}, f(\boldsymbol{x})) \in \text{vert}(\text{epi}(\text{conv}_M g(\boldsymbol{x}))\}.$$

Thus, the generating set of a function $g(\boldsymbol{x})$ is the set of all $\boldsymbol{x}$-coordinates of all vertices of the epigraph of the convex envelop of this function.

Let $g(\boldsymbol{x})$ be a multilinear function on n-dimensional convex polytope $M \subset \mathbb{R}^n$, and $\mathrm{conv}_M g(\boldsymbol{x})$ be a polyhedral function. And there exist $n+1$ linear independent vertices of $M : \varepsilon_i, i = 1, \ldots, n+1$.

Remark 5.8. Because of $\mathrm{conv}_M g(\boldsymbol{x}) = g(\boldsymbol{x})$, $\forall x \in \mathrm{vert}M$. Let us define function $g^\infty(\boldsymbol{x})$ such that $g^\infty(\boldsymbol{x}) = g(\boldsymbol{x})$ if $\boldsymbol{x} \in \mathrm{vert}M$ and $g^\infty(\boldsymbol{x}) = \infty$ otherwise. Thus the necessary and sufficient condition of the polyhedrality of the function $\mathrm{conv}_M g(\boldsymbol{x})$ can be rewritten in the following form:

$$\mathrm{conv}_M g(\boldsymbol{x}) = \mathrm{conv} f^\infty(\boldsymbol{x}) \quad \forall \boldsymbol{x} \in M. \tag{5.16}$$

Let $M = \mathrm{conv}\,\{\varepsilon_i | i = 1, \ldots, n+1\}$. Let $\boldsymbol{x}^0 \in M$, according to Caratheodory's theorem we have:

$$\mathrm{conv}_M g(\boldsymbol{x}^0) = \min \left\{ \sum_{i=1}^{n+1} \alpha_i g(\boldsymbol{x}^i) \;\middle|\; \begin{array}{l} \boldsymbol{x}^0 = \sum_{i=1}^{n+1} \alpha_i \boldsymbol{x}^i, \\ \sum_{i=1}^{n+1} \alpha_i = 1, \alpha_i \geq 0, \forall i = 1, \ldots, n+1, \\ \boldsymbol{x}^i \in P. \end{array} \right\} \tag{5.17}$$

Considering the following SDP programming for bilinear programming problems:

$$\begin{array}{rll} f^{bl} = & \text{minimize} & \boldsymbol{q_0}^\top \boldsymbol{y} \\ & \text{subject to} & P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} \leq -a_i, i = 1, \ldots, p, \\ & & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succeq 0, \\ & & \mathrm{diag}\, Y = \boldsymbol{y}. \end{array} \tag{5.18}$$

We assume that the relative interior of the following set, $(\boldsymbol{y}, Y)$, is nonempty:

$$\mathcal{Y} = \left\{ (\boldsymbol{y}, Y) \;\middle|\; \begin{array}{l} P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} < -a_i, i = 1, \ldots, p, \\ \boldsymbol{\mu}_j^\top \boldsymbol{y} < c_j, j = 1, 2, \ldots, m, \\ \boldsymbol{l} < \boldsymbol{y} < \boldsymbol{u}, \\ \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succ 0, \\ \operatorname{diag} Y = \boldsymbol{y}. \end{array} \right\}$$

We will show that problem (5.18) is an valid SDP relaxation for problem (5.6).

Theorem 5.9. *Considering problem* (5.6) *with* $diag(P_i) = \mathbf{0}$, $i = 1, \ldots, p$. *The problem* (5.18) *provides a lower bound for problem* (5.6). *Thus*

$$f^{bl} \leq f^*.$$

To prove theorem 5.9, we fist prove the following lemma.

Lemma 5.10. *Considering the following linearly constraints bilinear programming problem:*

$$\begin{array}{rll} f_l^* = & \text{minimize} & P \bullet Y + \boldsymbol{q}_0^\top \boldsymbol{y} \\ & \text{subject to} & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \end{array} \tag{5.19}$$

where $diag(P)$=$\mathbf{0}$. *Then the following provides a lower bound for problem* (5.19).

$$\begin{array}{rll} f_l^{bl} = & \text{minimize} & P \bullet Y' + \boldsymbol{q}_0^\top \boldsymbol{y} \\ & \text{subject to} & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y' \end{pmatrix} \succeq 0, \\ & & \operatorname{diag} Y' = \boldsymbol{y}. \end{array} \tag{5.20}$$

Thus,

$$f_l^{bl} \leq f_l^*.$$

Proof. Assume $\boldsymbol{y}$ is a feasible solution of problem (5.19). Let $M = [l, u]^n$, $\text{vert}(M) = \{\pi^i\} \in \{l, u\}^n$. Since $\boldsymbol{y} \in M$, there exist multipliers α_i such that

$$\boldsymbol{x} = \sum_{i=1}^{n+1} \alpha_i \boldsymbol{\pi}^i, \ \sum_{i=1}^{n+1} \alpha_i = 1, \ \alpha_i \geq 0. \tag{5.21}$$

Since $\text{diag}(P_i) = 0$, the quadratic expression $P \bullet Y$ is a bilinear and, hence, multilinear function. Using formula (5.17), there exist α_i satisfying (5.21) and such that

$$P \bullet Y \geq \sum_{i=1}^{n+1} \alpha_i P \bullet Y_{\pi^i},$$

where $Y_{\pi^i} = (\boldsymbol{\pi})^{\boldsymbol{t}}(\boldsymbol{\pi}^i)^\top$. Let $Y' = \sum_{i=1}^{n+1} (\boldsymbol{\pi})^{\boldsymbol{t}}(\boldsymbol{\pi}^i)^\top$, we have

$$P \bullet Y \geq P \bullet Y'. \tag{5.22}$$

Considering the matrix $\begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y' \end{pmatrix}$. By formula (5.21), we have

$$\begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y' \end{pmatrix} = \sum_{i=1}^{n+1} \begin{pmatrix} 1 & (\boldsymbol{\pi}^i)^\top \\ \boldsymbol{\pi} & (\boldsymbol{\pi})(\boldsymbol{\pi}^i)^\top \end{pmatrix} = \sum_{i=1}^{n+1} \begin{pmatrix} 1 \\ \boldsymbol{\pi}^i \end{pmatrix} \begin{pmatrix} 1 \\ \boldsymbol{\pi}^i \end{pmatrix}^\top \succeq 0. \tag{5.23}$$

In addition, $\text{diag} Y' = \sum_{i=1}^{n+1} \alpha_i \pi^i = \boldsymbol{y}$. Then, we have

$$\begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y' \end{pmatrix} \succeq 0,$$
$$\text{diag } Y' = \boldsymbol{y}.$$

It follows that $(\boldsymbol{y}, Y')$ is feasible to the SDP programming (5.20). Furthermore, because of inequality (5.22), we have

$$P \bullet Y + \boldsymbol{q}_0^\top \boldsymbol{y} \geq P \bullet Y' + \boldsymbol{q}_0^\top \boldsymbol{y}.$$

Therefore $f_l^{bl} \leq f_l^*$. □

Now we show theorem (5.9) is valid.

Proof. According to the Largrangian dual problem of problem (5.11) in Section 5.2. Since $\mathrm{diag}(P_i) = 0$ for $i = 1, \ldots, p$, the dual subproblem $f_0^{sl} = \min_{\boldsymbol{y}} L(\boldsymbol{y}, \boldsymbol{\lambda}, \mathbf{0})$ is a linearly bilinear programming problem. According lemma 5.10, the SDP relaxation for the dual subproblem is

$$\begin{array}{rlll} f_l^* = & \text{minimize} & P \bullet Y + \boldsymbol{q}_0^\top \boldsymbol{y} \\ & \text{subject to} & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \end{array} \tag{5.24}$$

where diag(P)=$\mathbf{0}$. Then the following provides a lower bound for problem (5.19).

$$\begin{array}{rlll} f_{l0}^{bl} = & \text{minimize} & \left(\sum_{i=1}^{p} \lambda_i P_i\right) \bullet Y' + \left(\boldsymbol{q}_0 + \sum_{i=1}^{p} \lambda_i q_i\right)^\top \boldsymbol{y} + \sum_{i=1}^{p} \lambda_i a_i \\ & \text{subject to} & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y' \end{pmatrix} \succeq 0, \\ & & \mathrm{diag}\, Y' = \boldsymbol{y}. \end{array} \tag{5.25}$$

Let a symtric matrix $A(\boldsymbol{\lambda}, \boldsymbol{\upsilon}, \boldsymbol{\omega}, \boldsymbol{\nu}, \boldsymbol{\beta}) \in \mathcal{S}^{n+1}$ be

$$\left(\begin{array}{c|c} \sum_{i=1}^{p} \lambda_i a_i & \frac{1}{2}\left(\boldsymbol{q_0} + \sum_{i=1}^{p} \lambda_i \boldsymbol{q}_i + \sum_{j=1}^{m} \upsilon_j \boldsymbol{\mu}_j - \boldsymbol{\omega} + \boldsymbol{\nu} + \boldsymbol{\beta}\right)^\top \\ \hline \cdots & \sum_{j=1}^{p} \lambda_j P_k - \mathrm{diag}(\boldsymbol{\beta}) \end{array}\right).$$

Under this notation, the dual of problem (5.25) is

$$\begin{array}{rlll} f_{l0}^{dbl} = & \underset{\boldsymbol{\upsilon}, \boldsymbol{\omega}, \boldsymbol{\nu}, \boldsymbol{\beta}}{\text{maximize}} & -\sum_{j=1}^{p} \upsilon_j c_j - \boldsymbol{\nu}^\top \boldsymbol{u} + \boldsymbol{\omega}^\top \boldsymbol{l} \\ & \text{subject to} & A(\boldsymbol{\lambda}, \boldsymbol{\upsilon}, \boldsymbol{\omega}, \boldsymbol{\nu}, \boldsymbol{\beta}) \succeq 0, \\ & & \boldsymbol{\upsilon} \geq \mathbf{0}, \boldsymbol{\omega} \geq \mathbf{0}, \boldsymbol{\nu} \geq \mathbf{0}. \end{array} \tag{5.26}$$

By weak duality, we have

$$f_{l0}^{dbl} \leq f_{l0}^{bl} \leq f^{sl}, \forall \boldsymbol{\lambda}.$$

Thus, the optimal value for the dual of problem (5.18), f^{dbl}, is a low bound of f^{sl}, where

$$\begin{array}{rlll} f_{sl}^{dbl} = & \underset{\boldsymbol{v},\boldsymbol{\omega},\boldsymbol{\nu},\boldsymbol{\beta}}{\text{maximize}} & -\sum_{j=1}^{p} v_j c_j - \boldsymbol{\nu}^\top \boldsymbol{u} + \boldsymbol{\omega}^\top \boldsymbol{l} \\ & \text{subject to} & A(\boldsymbol{\lambda}, \boldsymbol{v}, \boldsymbol{\omega}, \boldsymbol{\nu}, \boldsymbol{\beta}) \succeq 0, \\ & & \boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{v} \geq \mathbf{0}, \boldsymbol{\omega} \geq \mathbf{0}, \boldsymbol{\nu} \geq \mathbf{0}. \end{array} \tag{5.27}$$

The dual of problem (5.27) is :

$$\begin{array}{rlll} f^{bl} = & \text{minimize} & \boldsymbol{q_0}^\top \boldsymbol{y} \\ & \text{subject to} & P_i \bullet Y + \boldsymbol{q_i}^\top \boldsymbol{y} \leq -a_i, i = 1, \ldots, p, \\ & & \boldsymbol{\mu}_j^\top \boldsymbol{y} \leq c_j, j = 1, 2, \ldots, m, \\ & & \boldsymbol{l} \leq \boldsymbol{y} \leq \boldsymbol{u}, \\ & & \begin{pmatrix} 1 & \boldsymbol{y}^\top \\ \boldsymbol{y} & Y \end{pmatrix} \succeq 0, \\ & & \operatorname{diag} Y = \boldsymbol{y}. \end{array} \tag{5.28}$$

With the Slater Condition, we have $f^{bl} = f^{dbl}$. Therefore,

$$f^{bl} \leq f_0^{lg} \leq f^*.$$

□

5.5 Conclusions

In this chapter, we reformulated the SOLR problem into a quadratic programming with linear objective and quadratic and linear constraints. The reformulated problem are characterized by the zero diagonal elements of the constraint matrix. Then we reviewed several SDP relaxations and presented a number of new SDP relaxations which can be applied for solving the reformulated problem. Furthermore, the reformulated programming is a bilinear programming that is a special

case of general quadratic programming. Then an SDP relaxation for this particular problem based on the convex envelop of bilinear function can be applied for optimization. Of course, it provides an lower bound for the reformulated problem. In this chapter, some relationship of programmings with different relaxation methods are theoretically proved. Comparisons of these relaxations based on branch and bound algorithm should be addressed by further research.

Chapter 6

Conclusions and Further Works

6.1 Conclusions

We give contributions/conclusions of this dissertation in this section. The main contributions/conclusions of this dissertation falls in the following aspects:

1. Studied the performance of the DIRECT algorithm for solving the SOLR problem with lower dimension. We conducted numerical experiments to show the validness and to measure the probability of the DIRECT algorithm for obtaining a "good" solution for solving the SOLR problem within a given tolerance.

2. Proposed a branch and bound algorithm based on bisection branching rules to globally solve the SOLR problem with lower dimension. We transformed the SOLR problem into an equivalent problem which is a kind of quadratic problem with linear objective and quadratic and linear constraints. We made a linear relaxation for all the quadratic constraints instead of dropping them out. To invest the performance of proposed algorithm, we conducted numerical experiments using randomly generated data sets. And numerical results show that the proposed branch and bound algorithm based on bisection rules outperforms the existing algorithm proposed in [16]. More precisely, the proposed algorithm achieves superiority over the algorithm in [16]: i) reduced at least 93% in CPU time on average; ii) reduced at least 98% in the number of branches on average; iii) reduced at least 98.3% in the number of iterations

on average; iv) greatly reduced the CPU time, number of branches, number of iterations on max and min values.

3. Proposed a branch and bound algorithm based on advanced branching rules to globally solve the SOLR problem. The relaxed model enforces a strong approximation for the quadratic constraints of the original problem, which push us to develop an advanced branching rules. If selected rectangle which contains the current best solution are divided into two sub-rectangles, the advanced branching rules guarantes that the current best solution is belonged to a common edge of the two sub-rectangles. In addition, we proved the advanced branching rule is convergent. And the numerical results indicate that the average CPU time solved using advanced branch rules is less than it solved by bisection branching rules when $p \geq 30$. And the CPU time will be decreased much more with p growing.

4. Reformulated the SOLR into an SDP programming, reviewed several existing SDP relaxations for the reformulated programming and made comparisons for these relaxations.

6.2 Further Works

It is worthwhile to further work on the following directions based on the work we have conducted in this thesis.

1. To find a stopping criterion for the DIRECT algorithm for solving the SOLR problem within a given tolerance.

2. To make an analysis on the number of iteration in which the proposed algorithm finds a minimizer, and find an error bound between the minimizer and ε-minimizer.

3. To conduct larger scale experiments to look into the detail behavior of the proposed algorithms, e.g., to reduce constraints of the relaxed model.

4. To develop a branch and bound algorithm for globally solving the SOLR problem with SDP programming. Comparing the relaxations mentioned in

Chapter 5 by conducting numerical experiments is also worthwhile to be addressed.

Bibliography

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] Yoram Almogy and Oded Levin. Parametric analysis of a multi-stage stochastic shipping problem. *Operational Research*, 69:359–370, 1970.

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

[4] Kurt M Anstreicher and Samuel Burer. Dc versus copositive bounds for standard qp. *Journal of Global Optimization*, 33(2):299–312, 2005.

[5] Esther M. Arkin, Y-J Chiang, Martin Held, Joseph S. B. Mitchell, Vera Sacristan, SS Skiena, and T-C Yang. On minimum-area hulls. *Algorithmica*, 21(1):119–136, 1998.

[6] Xiaowei Bao, Nikolaos V Sahinidis, and Mohit Tawarmalani. Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. *Mathematical programming*, 129(1):129–157, 2011.

[7] Philippe Baptiste, Jacques Carlier, and Antoine Jouglet. A branch-and-bound procedure to minimize total tardiness on one machine with arbitrary release dates. *European Journal of Operational Research*, 158(3):595–608, 2004.

[8] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*, volume 2. Siam, 2001.

[9] Harold P Benson. Solving sum of ratios fractional programs via concave minimization. *Jouranl Optimization Theory Application*, 135(1):1–17, 2007.

[10] HP Benson. On the global optimization of sums of linear fractional functions over a convex set. *Journal of Optimization Theory and Application*, 121(1):19–39, April 2004.

[11] Immanuel M Bomze, Florian Frommlet, and Martin Rubey. Improved sdp bounds for minimizing quadratic functions over the \ ellˆ{1}-ball. *Optimization Letters*, 1(1):49–59, 2007.

[12] Immanuel M Bomze, Marco Locatelli, and Fabio Tardella. New and old bounds for standard quadratic optimization: dominance, equivalence and incomparability. *Mathematical Programming*, 115(1):31–64, 2008.

[13] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[14] Samuel Burer and Dieter Vandenbussche. A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations. *Mathematical Programming*, 113(2):259–282, 2008.

[15] Alberto Cambini, Laura Martein, and Siegfried Schaible. On maximizing a sum of ratios. *Journal of Information and Optimization Sciences*, 10(1):65–79, 1989.

[16] John Gunnar Carlsson and Jianming Shi. A linear relaxation algorithm for solving the sum-of-linear-ratios problem with lower dimension. *Operations Research Letters*, 41(4):381–389, 2013.

[17] Giorgio Carpaneto, Mauro Dell'Amico, and Paolo Toth. Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software (TOMS)*, 21(4):394–409, 1995.

[18] R Chandrasekaran. Minimal ratio spanning trees. *Networks*, 7(4):335–342, 1977.

[19] Abraham Charnes and William W Cooper. Programming with linear fractional functionals. *Naval Research logistics quarterly*, 9(3-4):181–186, 1962.

[20] Danny Z Chen, Ovidiu Daescu, Yang Dai, Naoki Katoh, Xiadong Wu, and Jinhui Xu. Optimizing the sum of linear fractional functions and applications.

In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 707–716. Society for Industrial and Applied Mathematics, 2000.

[21] Danny Z Chen, Ovidiu Daescu, Yang Dai, Naoki Katoh, Xiadong Wu, and Jinhui Xu. Efficient algorithms and implementations for optimizing the sum of linear fractional functions, with applications. *Journal of Combinatorial Optimization*, 9(1):69–90, 2005.

[22] Danny Z Chen, Ovidiu Daescu, Xiaobo Sharon Hu, Xiaodong Wu, and Jinhui Xu. Determining an optimal penetration among weighted regions in two and three dimensions. *Journal of Combinatorial Optimization*, 5(1):59–79, 2001.

[23] Jae Chul Choi and Dennis L Bricker. Effectiveness of a geometric programming algorithm for optimization of machining economics models. *Computers & operations research*, 23(10):957–961, 1996.

[24] Jens Clausen. Branch and bound algorithms-principles and examples. *Department of Computer Science, University of Copenhagen*, pages 1–30, 1999.

[25] ILOG CPLEX. High-performance software for mathematical programming and optimization, 2005.

[26] Yang Dai, Jianming Shi, and Shouyang Wang. Conical partition algorithm for maximizing the sum of dc ratios. *Journal of Global Optimization*, 31(2):253–270, 2005.

[27] Karen Daniels. The restrict/evaluate/subdivide paradigm for translational containment. In *Fifth MSI Stony Brook Workshop on Computational Geometry*, 1995.

[28] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1965.

[29] Anton Dekkers and Emile Aarts. Global optimization and simulated annealing. *Mathematical programming*, 50(1-3):367–393, 1991.

[30] Daniele Depetrini and Marco Locatelli. Approximation of linear fractional-multiplicative problems. *Mathmatical Programming*, 128(1-2):437–443, 2011.

[31] Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[32] MLDM Eigen and Lo DeMaeyer. Relaxation methods. *Techniques of organic chemistry*, 8(Part II), 1963.

[33] James E Falk and Susan W Palocsay. Image space analysis of generalized fractional programs. *Journal of Global Optimization*, 4(1):63–88, 1994.

[34] Roland W Freund and Florian Jarre. Solving the sum-of-ratios problem by an interior-point method. *Journal of Global Optimization*, 19(1):83–102, 2001.

[35] Tetsuya Fujie and Masakazu Kojima. Semidefinite programming relaxation for nonconvex quadratic programs. *Journal of Global Optimization*, 10(4):367–380, 1997.

[36] Jorg M Gablonsk. *Modification of the DIRECT algorithm*. PhD thesis, North Carolina State University, Raleigh,North Carolina, 2001.

[37] Lianbo Gao, Shashi K Mishra, and Jianming Shi. An extension of branch-and-bound algorithm for solving sum-of-nonlinear-ratios problem. *Optimization Letters*, 6(2):221–230, 2012.

[38] Michael R Garey and David S Johnson. Computers and intractability: a guide to np-completeness, 1979.

[39] M Grant and S Boyd. Cvx: Matlab software for disciplined convex programming, version 1.21 (2011). *Available: cvxr. com/cvx*, 2010.

[40] Patrick JF Groenen and Willem J Heiser. The tunneling method for global optimization in multidimensional scaling. *Psychometrika*, 61(3):529–550, 1996.

[41] Christoph Helmberg. *Semidefinite programming for combinatorial optimization*. Konrad-Zuse-Zentrum für Informationstechnik Berlin, 2000.

[42] Reiner Horst and Hoang Tuy. *Global optimization: Deterministic approaches*. Springer, 1996.

[43] Yongwen Hu and Jianming Shi. Algorithms for the sum of linear ratios problem with lower dimension and related problems. Oral presentation at The

9th International Conference on Optimization: Techniques and Applications. Taipei, Taiwan, December 2013.

[44] Yongwen Hu, Jianming Shi, and Shinya Watanabe. A new liner relaxation algorithm for sum of linear ratios problem with lower dimension. Oral presentation at Satellite Conference: The Fourth Conference on Nonlinear Analysis and Optimization. Taipei, Taiwan, August 2014.

[45] Yongwen Hu, Jianming Shi, and Shinya Watanabe. A revised algorithm for solving the sum of linear ratios problem with lower dimension using linear relaxation. *International Journal of Operations Research*, 11(1):28–39, 2014.

[46] Hongwei Jiao, Qigao Feng, Peiping Shen, and Yunrui Guo. Global optimization for sum of linear ratios problem using new pruning technique. *Mathematical Problems in Engineering*, 2008:1–13, 2008.

[47] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Application*, 79(1):157–181, 1993.

[48] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311. ACM, 1984.

[49] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[50] Leonid G Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72, 1980.

[51] Jong-Sung Kim and Ki-Sang Hong. A recursive camera resectioning technique for off-line video-based augmented reality. *Pattern recognition letters*, 28(7):842–853, 2007.

[52] Hiroshi Konno and Natsuroh Abe. Minimization of the sum of three linear fractional functions. *Journal of Global Optimization*, 15(4):419–432, 1999.

[53] Hiroshi Konno and Kenji Fukaishi. A branch and bound algorithm for solving low rank linear multiplicative and fractional programming problems. *Journal of Global Optimization*, 18:283–299, 2000.

[54] Hiroshi Konno and Michimori Inori. Bond portfolio optimization by bilinear fractional programming. *Journal of the Operations Research Society of Japan*, 32(2):143–158, 1989.

[55] Hiroshi Konno, Takahito Kuno, and Yasutoshi Yajima. Global minimization of a generalized convex multiplicative function. *Journal of Global Optimization*, 4:47–62, 1994.

[56] Hiroshi Konno and Hidetoshi Watanabe. Bond portfolio optimization problems and their applications to index tracking: a partial optimization approach. *Journal of the Operations Research Society of Japan-Keiei Kagaku*, 39(3):295–306, 1996.

[57] Hiroshi Konno, Yasutoshi Yajima, and Tomomi Matsui. Parametric simplex algorithms for solving a special class of nonconvex minimization problems. *Journal of Global Optimization*, 1(1):65–81, 1991.

[58] J Kostrowicki and L Piela. Diffusion equation method of global minimization: Performance for standard test functions. *Journal of Optimization Theory and Applications*, 69(2):269–284, 1991.

[59] Takahito Kuno. A branch-and-bound algorithm for maximizing the sum of several linear ratios. *Journal of Global Optimization*, 22(1-4):155–174, 2002.

[60] Takahito Kuno and Toshiyuki Masaki. A practical but rigorous approach to sum-of-ratios optimization in geometric applications. *Computational Optimization and Applications*, 54(1):93–109, 2013.

[61] Jayanth Majhi, Ravi Janardan, Jörg Schwerdt, Michiel Smid, and Prosenjit Gupta. Minimizing support structures and trapped area in two-dimensional layered manufacturing. *Computational Geometry*, 12(3):241–267, 1999.

[62] Richard Kipp Martin. *Large Scale Linear and Integer Optimization: A Unified Approach: A Unified Approach*. Springer, 1999.

[63] Tomomi Matsui. Np-hardness of linear multiplicative programming and related problems. *Journal of Global Optimization*, 9(2):113–119, 1996.

[64] R Garey Michael and David S Johnson. Computers and intractability: A guide to the theory of np-completeness. *WH Freeman & Co., San Francisco*, 1979.

[65] Donald L Miller and Joseph F Pekny. Exact solution of large asymmetric traveling salesman problems. *Science*, 251(4995):754–761, 1991.

[66] Gurobi Optimization. Gurobi optimizer reference manual. *URL: http://www. gurobi. com*, 2012.

[67] Panos M Pardalos and Stephen A Vavasis. Quadratic programming with one negative eigenvalue is np-hard. *Journal of Global Optimization*, 1(1):15–22, 1991.

[68] J Pinter. Global optimization in action, volume 6 of nonconvex optimization and its applications, 1995.

[69] Imre Polik. Sedumi. *Download from http://sedumi. ie. lehigh. edu*, 2010.

[70] Siegfried Schaible. A note on the sum of a linear and linear-fractional function. *Naval Research Logistics Quarterly*, 24(4):691–693, 1977.

[71] Siegfried Schaible. Fractional programming: applications and algorithms. *European Journal of Operational Research*, 7(2):111–120, 1981.

[72] Siegfried Schaible and Toshidide Ibaraki. Fractional programming. *European Journal of Operational Research*, 12(4):325–338, 1983.

[73] Siegfried Schaible and Jianming Shi. Fractional programming: the sum-of-ratios case. *Optimization Methods and Software*, 2(18):219–229, 2003.

[74] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[75] AE Sepulveda and L Epstein. The repulsion algorithm, a new multistart method for global optimization. *Structural optimization*, 11(3-4):145–152, 1996.

[76] Hanif D Sherali. Rlt: A unified approach for discrete and continuous nonconvex optimization. *Annals of Operations Research*, 149(1):185–193, 2007.

[77] Naum Z Shor. Quadratic optimization problems. *Soviet Journal of Computer and Systems Sciences*, 25(6):1–11, 1987.

[78] NZ Shor. Dual quadratic estimates in polynomial and boolean programming. *Annals of Operations Research*, 25(1):163–168, 1990.

[79] Christopher C Skiscim and Susan W Palocsay. Minimum spanning trees with sums of ratios. *Journal of Global Optimization*, 19:103–120, 2001.

[80] IM Stancu-Minasian. *Fractional programming*. Springer, 1997.

[81] Said F Tantawy. A new method for solving linear fractional programming problems. *Australian Journal of Basic and Applied Sciences*, 1(2):105–108, 2007.

[82] Tim Van Voorhis. A global optimization algorithm using lagrangian underestimates and the interval newton method. *Journal of Global Optimization*, 24(3):349–370, 2002.

[83] RJ Vanderbei. Linear programming: foundations and extensions, 1998.

[84] Stephen A Vavasis. Quadratic programming is in np. *Information Processing Letters*, 36(2):73–77, 1990.

[85] Henry Wolkowicz. Semidefinite and lagrangian relaxations for hard combinatorial problems. In *System Modelling and Optimization*, pages 269–309. Springer, 2000.

[86] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer, 2000.

[87] R Yamamoto and H Konno. An efficient algorithm for solving convex–convex quadratic fractional programs. *Journal of Optimization Theory and Applications*, 133(2):241–255, 2007.

[88] Wang Yanjun, Shen Peiping, and Liang Zhian. A branch-and-bound algorithm to globally solve the sum of several linear ratios. *Applied Mathematics and Computation*, 168(1):89–101, 2005.

[89] Y Zhang. Lipsol: a matlab toolkit for linear programming. *Department of Mathematics and Statistics, University of Maryland, Baltimore County, Maryland*, 1995.