A Geometric Approach to Integer Optimization and Its Application for Reachability Analysis in Petri Nets

GU, Shenshen

A Thesis Submitted in Partial Fulfilment

of the Requirements for the Degree of

Doctor of Philosophy

in

Automation and Computer-Aided Engineering

July 2009

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 - 1346

Thesis/Assessment Committee

Professor Chung, Chi-kit Ronald (Chair)

Professor Wang, Jun (Thesis Supervisor)

Professor Li, Duan (Thesis Supervisor)

Professor Lam, Kai-Pui (Committee Member)

Professor Li, Han-Lin (External Examiner)

Abstract

Integer programming plays an important role in operations research and has a wide range of applications in various fields. There are a lot of research directions in the area of integer programming. In this thesis, two main topics will be investigated in details. One is to find the optimal binary solution to a quadratic object function, and the other is to find integer solutions to linear equations.

Finding the optimal binary solution to a quadratic object function is known as the Binary Quadratic Programming problem (BQP), which has been studied extensively in the last three decades. In this thesis, by investigating geometric features of the ellipse contour of a concave quadratic function, we derive new upper and lower bounding methods for BQP. Integrating these new bounding schemes into a proposed solution algorithm of a branch-and-bound type, we propose an exact solution method in solving general BQP with promising preliminary computational results. Meanwhile, by investigating some special structures of the second order matrix and linear term in BQP, several polynomial time algorithms are discussed to solve some special cases of BQP.

In the realm of integer programming, finding integer solutions to linear equations is another important research direction. The problem is proved to be NP-Complete, and several algorithms have been proposed such as the algorithm based on linear Diophantine equations as well as the method based on Groebner bases. Unlike the traditional algorithms, the new efficient method we propose in this thesis is based on our results on zero duality gap and the cell enumeration of an arrangement of hyperplanes in discrete geometry.

Finding integer solutions to linear equations has various real world applications. In the thesis, we investigate its application to the reachability analysis of Petri nets. Introduced by Petri in 1962, Petri net has been a powerful mathematical formalism for modeling, analyzing and designing discrete event systems. In the research community of Petri nets, finding a feasible path from the initial state to the target state in Petri net, known as reachability analysis, is probably one of the most important and challenging subjects. The reachability algebraic analysis is equivalent to finding the nonnegative integer solutions to a fundamental equation constructed from the Petri net. We apply our algorithm in this thesis to reachability analysis of Petri net by

finding the nonnegative integer solutions to the fundamental equation.

摘要

整數規劃不僅在運籌學研究中扮演著十分重要的角色同時對各種領域也有著十分廣泛的應用．在整數規劃研究領域中具有許多研究方向．本篇論文具體討論其中兩個主要課題．其一是尋找二次目標函數的最優 0-1 解，其二是求解線性方程組的整數解．

尋找二次目標函數的最優 0-1 解也稱為 0-1 二次規劃問題．在過去的三十多年時間裡該問題得到了廣泛的研究．在本篇論文中，通過探索二次凹函數橢圓型等高線的幾何特性，我們對 0-1 二次規劃問題提出了新的計算上界和下界的方法．將這些方法與一個分支定界類型的求解演算法相結合，我們得到了一個求解 0-1 二次規劃問題的確切解演算法．同時，通過對 0-1 二次規劃問題的二次矩陣和線性項特殊結構的研究，我們討論了 0-1 二次規劃問題中一些特例的多項式時間複雜度演算法．

在整數規劃的研究領域中，求解線性方程組的整數解也是一個重要的研究方向．該問題被證明是 NP-Complete 問題．對於該問題，一些演算法相繼提出，諸如基於線性 Diophantine 方程組的演算法以及基於 Groebner 基的方法等等．同這些傳統演算法不同，我們在論文中提出的新演算法是基於我們對零對偶間隙的研究結果以及在離散幾何學中對一組超平面所劃分區域進行枚舉的方法．

求解線性方程組整數解的問題具有許多現實的應用．在本論文中，我們研究了其在 Petri 網可達性分析中的應用．1962 年由 Petri 提出的 Petri 網是一個對離散事件系統進行建模、分析以及設計十分有效的數學形式．在 Petri 網的研究中，尋找從初始狀態到終結狀態的有效途徑（即可達性分析）無疑是最重要且最具有挑戰性的課題之一．可達性的代數分析方法等同於對一個由 Petri 網構造的基本方程的非負整數解進行求解．通過尋找基本方程的非負整數解，我們將論文中所提出的算法應用於 Petri 網可達性分析．

Acknowledgement

I would like to express my heartful gratitude to my supervisors, Professor WANG Jun and Professor LI Duan, for their supervision throughout these four years. Without their continuous encouragement and help, this thesis is unlikely to be accomplished in its shape as it is. What is more, I am deeply grateful to Professor LI Duan for his patience and kindness. He always puts high priority on my research and is willing to discuss new ideas with me at anytime he is available. I have learnt a lot from him these years.

I am also grateful to my Ph.D. thesis committee members, Professor LI Han-Lin and Professor LAM Kai-Pui, for giving valuable comments on my research works.

I treasure this chance to thank in particular Professor SUN Xiaolin and Professor YAO Yirong who give me great assistance and valuable comments on my research.

In addition, I would like to convey my appreciation to my labmates and friends GAO Jianjun, YI Lan, LIU Chunli, CHIU Mei-Choi, CUI Xiangyu, WANG Chongyu, HU Xiaolin, Liu Qingshan, Xiang Tao, PAN Yunpeng, HU Jin and former lab visitors Professor ZENG Zhigang and Professor SHEN Yi.

This work is dedicated to my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Integer programming plays an important role in operations research and has a wide range of applications in various fields. There are several research directions in the area of integer programming. In this thesis, we investigate two main topics in details. One topic is to find the optimal binary solution to a quadratic object function. The other is to find integer solutions to linear equations.

1.1 Binary Quadratic Programming Problem

Binary quadratic programming problem is a well known problem in the field of operations research, which is described as follows:

$$(P) \qquad \min_{x\in\{0,1\}^n} f(x) = \frac{1}{2}x^T Qx + c^T x.$$

Binary quadratic programming problem has wide spectra of applications in a variety of fields, for example, financial analysis, molecular conformation problem and cellular radio channel assignment. Many combinatorial optimization problems, such as Max-Cut problem, are special cases of (P). As Max-Cut problem has been proved to be NP-hard, (P) is NP-hard in general.

To solve the binary quadratic programming problem, many algorithms have been proposed. These algorithms can be divided into two main categories. One is the class of heuristic algorithms [17, 18, 19], and the other is the class of exact solution algorithms. Heuristic algorithms include, for example, neural network algorithm, genetic algorithm and simulated annealing al-

gorithm. Exact solution methods for solving (P) can be roughly classified into five categories: algebraic method [33, 1], linearization method [2, 3, 4], cutting plane method [5, 25, 43], reformulation to a quadratic concave minimization problem [7, 8, 9, 10, 11, 12] and implicit enumeration method or branch-and-bound method [29, 13, 14, 32, 15, 16].

For the heuristic algorithms, the main advantage is their cheap computation cost and fast computational speed. However, they have a fatal disadvantage: There is no guarantee to find the global optimum.

For the exact solution algorithms, there is a guarantee to find the global optimum. However, the expense is the computational cost which might be exponential to the problem dimension.

Our idea in this thesis is to take the advantage of both heuristic algorithms and exact solution algorithms. Our algorithm is an exact solution method to find the global optimum. Meanwhile, by investigating geometric properties of the problem, we adopt some heuristic rules in the branch and bound algorithm to speed up the convergence speed.

For the exact solution scheme, we focus in this paper on development of an exact solution method of a branch-and-bound type for solving (P). It is well-known that the efficiency of a branch-and-bound method for solving a general integer minimization problem largely depends on the quality of the upper and lower bounds and the corresponding computational efforts to obtaion them. Existing approaches in the literature for computing the bound of (P) and its subproblems include simple lower bound estimation and improvements via reformulations [16, 48], convex quadratic programming relaxation [29, 14], roof duality [13, 40], decomposition method [32] and semi-definite programming relaxation [43]. Another significant contributing factor to the efficiency of the solution algorithms for binary quadratic programming, although received much less attentions in the literatures, is the capability of variable fixation to fix certain variables at their optimal values based on some optimality conditions. The bounds of the gradient of the objective function is used in [16, 48] to fix variables. The outer box(rectangular) containing the ellipse contour of the objective function is employed in [14] to fix variables.

As a special but important case of integer programming, binary quadratic programming problem processes rich geometric properties. Although there exist rich geometric properties in binary quadratic programming, only a few papers, e.g. [18], have devoted to explore such prominent features hidden behind until recently. An exact solution method has been recently proposed by exploring these rich geometric properties [45]. In that paper, the object function is

assumed to be convex. Unlike that method, in this thesis the objective function is assumed to be concave. Under this assumption, we study the geometric properties behind the original problem to propose some new bounding algorithms and variable fixation methods. The reason to study the objective function in concave form is that, for the concave cases, it is more convenient to find an integer point outside the current contour so that a better solution can be found. We exploit the geometric properties of the ellipse contour of a concave quadratic function. One new upper bound for (P) is derived by finding some special points on the contour. Two new valid lower bounds for (P) are derived by constructing a family of minimum ellipse contours. Based on the properties of the maximum inscribed sphere inside the ellipse contour, a new variable fixation condition is also derived. These findings further lead to some new optimality conditions for binary quadratic programming. Integrating these prominent features of binary quadratic programming into an exact solution scheme, we have developed a solution algorithm of a branch-and-bound type. Preliminary numerical results for test problems in the literature show that our proposed solution algorithm is promising.

1.2 Finding Integer Solutions to Linear Equations

Consider the linear Diophantine equations: $Ax = b, x \in \mathbb{Z}^n$ and the Diophantine equations on a bounded integer set: $Ax = b, x \in X = \{x \in \mathbb{Z}^n | l \leq x \leq u\}$, where $\mathbb{Z}^n$ denotes the set all integer vectors in $\mathbb{R}^n$, A is an $m \times n$ integral matrix with $m < n$ and rank$(A) = m$, and $b \in \mathbb{R}^m$ is integral. It is well known that linear diophantine equations are polynomially solvable [66, 47], while linear diophantine equations on a bounded integer set are NP-complete, as the special case of linear Diophantine Equations with $m = 1$, $x \in \{-1, 1\}^n$ and $b = 0$ is NP-complete [37].

The most classical method in solving linear Diophantine equations is the Smith normal form [20]. And the most popular method in solving linear Diophantine equations is the so called Hermite normal form [47].

Based on the Hermite normal form, many algorithms have been designed for finding integer solutions to linear Diophantine equations. As the computation of the Smith normal form and the Hermite normal form of integer matrix A plays a central role in finding integer solutions to linear Diophantine equations, some methods, such as the Euclidean algorithm in [47], have

been devised to improve the algorithmic efficiency. However, a notorious phenomenon of coefficient explosion gives rise a major obstacle in such computations. Various strategies have been proposed to alleviate this computational difficulty.

Under the linear transformation $x = y + l$, solving linear Diophantine equations over bounded integer set $[l, u]$ is equivalent to solving the following problem: $Ay = d, 0 \leq y \leq \beta$, $y \in \mathbb{Z}^n$, where $d = b - Al$ and $\beta = u - l$. Based on lattice basis reduction, an algorithm was developed to identify if there exists a $y \in \mathbb{Z}^n$ satisfying bound constraints. In fact, the algorithm is to firstly derive the Hermite normal form, based on the Lovassz basis reduction algorithm which possesses good capability in avoiding coefficient explosion, to obtain a short solution x_d and a short basis x_λ to system $Ay = d$. Secondly, branching on integer linear combinations of x_λ is adopted to obtain a solution that satisfies bound constraints, or to prove an infeasibility.

Utilizing the results of [21] to express explicitly integer solutions to a linear equation of two variables, [22] developed an algorithm based on the Euclid's algorithm for computing the set of integer solution of $Ax = b$ on bounded set X. In its first phase, the algorithm in [22] reduces the problem dimension recursively by aggregating two variables into an artificial variable with calculated lower and upper bounds, finally into a linear system with only two variables whose integer solutions can be specified. In the second phase of the algorithm, by repeating using the results for a linear equation of two variables, the solution set is expanded by determining progressively integer values for remaining undecided elements of x.

We propose in the thesis a novel method for solving the linear Diophantine equations on a bounded integer set. Our method is based on a recognition that whether or not there exists a solution to the fundamental solution is equivalent to whether or not the distance from X to the affine solution set of $Ax = b$ in R^n attains zero. Furthermore, from recent results in [67] for solving binary quadratic programming problem, finding the distance from an integer set to an affine set can be efficiently achieved by the cell enumeration method for an arrangement of hyperplanes in discrete geometry. This cell enumeration approach provides a promising platform for designing an efficient method to find integer solutions to linear equations on a bounded integer set.

1.3 Petri Nets and Reachability Analysis

A Petri net is a particular kind of bipartite directed graph consisting of three types of elements: places, transitions, and directed arcs connecting places and transitions. In a graphical representation of a Petri net, places are depicted by circles and transitions as bars. Furthermore, each place may hold a non-negative number of tokens, depicted by a corresponding number of solid dots. The distribution of tokens on places, called Petri net marking, defines the state of the modeled system. A marking for a Petri net with m places is represented by an $(m \times 1)$ vector M, where $M_j, j = 1, 2, \ldots, m$, are nonnegative integers representing the number of tokens in the corresponding places. Fig.3.3.2 illustrate a simple Petri net with four places, four transitions and eight arcs.

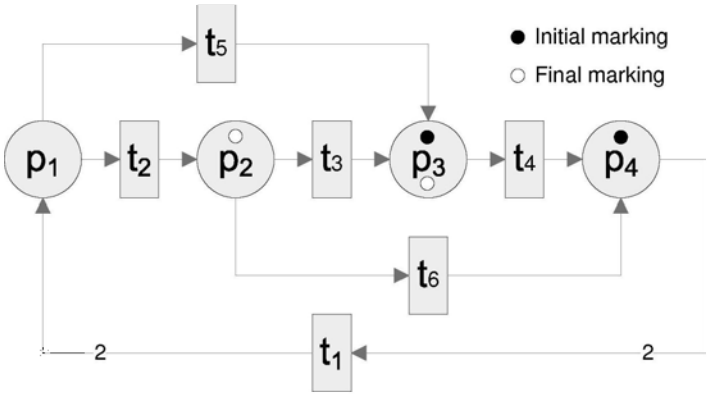


Figure 1.1: A simple with four places and four transitions

A general Petri net is characterized by a five-tuple (P, T, I, O, M_0) [69], where $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of m places, $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of n transitions, D_I: $(P \times T) \mapsto Z_+^{m \times n}$ is an input function that defines weights associated with directed arcs from places to transitions, where $Z_+^{m \times n}$ (Z_+^m) is the set of $(m \times n)$ dimensional matrices (m dimensional vector) with all entries being in Z_+, D_O: $(P \times T) \mapsto Z_+^{m \times n}$ is an output function that defines weights associated with directed arcs from transitions to places, and M_0: $P \mapsto Z_+^m$ is the initial marking.

The change of the distribution of the tokens represents the dynamics of the modeled system, while the distribution of tokens on places may change according to the following enabling rule

and firing rule [69]:

Enabling Rule: A transition t is said to be enabled if each input place p of t contains at least the number of tokens equal to the weight of the arc connecting p to t.

Firing Rule: (a) An enabled transition t may or may not fire depending on the additional interpretation, and (b) A firing of an enabled transition t removes form each input place p the number of tokens equal to the weight of the directed arc connecting p to t, and deposits in each output place p the number of tokens equal to the weight of the arc connecting t to p.

Reachability analysis is no doubt one of the most important behavioral properties of Petri nets. Given both the initial state M_0 and a target state M, a natural question to answer is whether or not we have a sequence of firing rules such that the system can reach the specific target state within finite steps. There are two primary approaches in investigating reachability: Reachability graph analysis and reachability algebraic analysis. The first method is based on the creation and investigation of a reachability graph or a reduced reachability graph. However using the reachability graph will be encountered with the state explosion problem while transferring the reachability graph to a reduced counterpart is proved to be NP-hard. The second approach is based on methods of linear algebra. It is well known that a necessary condition for reachability of marking M from some other marking M_0 of a Petri net is the existence of a nonnegative integer vector solution of the following system of linear equations,

$$M = M_0 + DF \tag{1}$$

where $F = [f_1, f_2, \ldots, f_m]'$ is a firing count vector with f_i indicating the number of firing for transition i in the whole progress and the incidence matrix is given by

$$D = [d(t_i, p_j)] = [d_{ij}],\ i = 1, 2, \ldots, m = |T|,\ j = 1, 2, \ldots, n = |P| \tag{2}$$

where $d(t_i, p_j) = O(p_j, t_i) - I(p_j, t_i)$. This equation is often called the fundamental equation of Petri net. When the Petri net is acyclic, i.e., has no directed circuits, the above condition becomes both necessary and sufficient[61].

The first step in the reachability algebraic analysis is to find firing count vectors by solving the fundamental equation directly or by other methods. Then the second step is to translate a firing count vector into a firing sequence, if there is any.

1.4 Thesis Organization

In this thesis, both the binary quadratic programming problem and the problem of finding the integer solutions to linear equation are discussed. Algorithms to the above two problems are proposed and studied in details in this work. This thesis is divided into seven chapters and it is organized as follows:

In Chapter 2, some polynomial-time solvable algorithms to several special cases of the binary quadratic programming problem are summarized.

In Chapter 3, a new branch and bound type algorithm for the binary quadratic programming problem is proposed. By investigating the geometric properties of the binary quadratic programming problem, some novel algorithms to calculate the upper bounds and lower bounds are designed to improve the algorithm efficiency.

In Chapter 4, the properties of three special diagonal cases of binary quadratic programming problem are studied. And by combining the basic algorithm and the dynamic programming algorithm, three polynomial-time solvable algorithms to these special problems are proposed.

Chapter 5 gives a new algorithm, based on our results on zero duality gap and the cell enumeration of an arrangement of hyperplanes in discrete geometry, to find the integer solutions to linear equations.

In Chapter 6, a brief introduction to Petri Nets is given. In the realm of Petri Nets, the study of reachability analysis is of a highest concern. Since reachability analysis is equivalent to finding integer solutions to certain linear equations, the algorithm proposed in Chapter 5 is applied to reachability analysis.

Chapter 7 gives a conclusion to this thesis. In addition, several future research directions in these areas are also presented.

□ **End of chapter.**

Chapter 2

Polynomially Solvable Cases of Binary Quadratic Programs

We summarize in this chapter polynomially solvable subclasses of binary quadratic programming problems studied in the literature and report some new polynomially solvable subclasses revealed in our recent research. It is well known that the general binary quadratic programming program is NP-hard. Identifying polynomially solvable subclasses of binary quadratic programming problems not only offers theoretical insight into the complicated nature of the problem, but also provides platforms to design relaxation schemes for exact solution methods. We discuss and analyze in this chapter five polynomially solvable subclasses of binary quadratic programs, including problems with special structures in the matrix Q of the quadratic objective function, problems defined by a special graph or a logic circuit and problems characterized by zero duality gap of the SDP relaxation. Examples and geometric illustrations are presented to provide algorithmic and intuitive insights into the problems.

2.1 Introduction

We consider in this chapter the following unconstrained 0-1 quadratic programming or binary quadratic programming problem:

$$(0\text{-}1QP) \qquad \min_{x\in\{0,1\}^n} x^TQx + c^Tx,$$

where $Q = (q_{ij})_{n\times n}$ is symmetric and $c \in \mathbb{R}^n$. Termed also as the *pseudo-Boolean programming*, problem $(0\text{-}1QP)$ is a classical combinatorial optimization problem and is well known

to be NP-hard (see [37]).

There exist many real-world applications of 0-1 quadratic programming, including financial analysis [46], molecular conformation problem [49] and cellular radio channel assignment [32]. Many combinatorial optimization problems, such as the Max-Cut problem (see e.g., [34, 38]), are special cases of the 0-1 quadratic programming problems. Various exact solution methods of a branch-and-bound framework for solving $(0\text{-}1QP)$ and its variants have been proposed in the literature (see, e.g., [26][29][32][43][44][45][48][51] and the references therein).

We focus in this chapter on the polynomially solvable cases of the quadratic binary programming problems. Identifying polynomially solvable subclasses of binary quadratic programming problems not only offers theoretical insight into the complicated nature of the problem, but also provides useful information and powerful relaxations for designing efficient algorithms for finding optimal solution to $(0\text{-}1QP)$. More specifically, the properties of the polynomially solvable subclasses of $(0\text{-}1QP)$ provide hints and facilitate the derivation of efficient relaxations for the general form of $(0\text{-}1QP)$. Polynomially solvable binary quadratic programs even play an important role in devising exact methods for linearly constrained quadratic 0-1 programming. For example, the Lagrangian relaxation of the quadratic 0-1 knapsack problem, which is a special case of $(0\text{-}1QP)$, turns out to be polynomially solvable and thus makes it possible to efficiently compute the Lagrangian bounds in a branch-and-bound method for the quadratic 0-1 knapsack problem.

It is sometimes more convenient to consider some equivalent forms of $(0\text{-}1QP)$. Since $x_i^2 = x_i$ for $x_i \in \{0, 1\}$, $(0\text{-}1QP)$ can be reduced to the following homogenous form $(0\text{-}1QP_h)$ without the linear term, using the substitution $Q := Q + \mathrm{diag}(c)$, where $\mathrm{diag}(c)$ is the diagonal matrix formed by vector c,

$$(0\text{-}1QP_h) \qquad \min_{x\in\{0,1\}^n} x^T Q x.$$

In many binary quadratic programming models arising from combinatorial optimization, the decision variables take values -1 or 1. The resulting binary quadratic programs take the following form:

$$(BQP) \qquad \min_{x\in\{-1,1\}^n} x^T Q x + c^T x.$$

It can be seen that $(0\text{-}1QP)$ with 0-1 variables (in x space) can be reduced to a form of (BQP) with $(-1, 1)$ variables (in y space) using transformation $x_i = \frac{1}{2}(y_i + 1)$.

As $x_i^2 = 1$, for both $x_i = 1$ and -1, we can assume, without loss of generality, that all diagonal elements of Q in (BQP) are zero. Thus, we can write the objective function in (BQP) as

$$\sum_{1\le i<j\le n} 2q_{ij}x_ix_j + \sum_{i=0}^{n} c_ix_i.$$

By introducing an artificial variable $x_0 = 1$, we further have

$$f(x) = \sum_{0\le i<j\le n} 2q_{ij}x_ix_j,$$

where $q_{0i} = \frac{1}{2}c_i$, $i = 1, \ldots, n$. Since for any $x \in \{-1, 1\}^{n+1}$, $f(x) = f(-x)$, we can relax the domain of x_0 to $\{-1, 1\}$ and (BQP) now takes the following equivalent homogenous form:

$$(BQP_h) \qquad \min_{x\in\{-1,1\}^{n+1}} x^TQx,$$

where $Q := \begin{pmatrix} 0 & \frac{1}{2}c^T \\ \frac{1}{2}c & Q \end{pmatrix}$.

The well-known max-cut problem, which has being attracting remarkable attentions in recent years in combinatorial optimization, can be expressed in the form of (BQP_h). Consider a graph $G = (E, V)$ with vertex set $V = \{1, \ldots, n\}$ and edge set $E = \{ij \mid 1 \le i < j \le n\}$. For every edge $ij \in E$, there is an associated weight w_{ij}. For a given set $S \subseteq V$, a *cut* $\delta(S)$ is the set of all edges with one endpoint in S and the other in $V \setminus S$, and the weight of cut $\delta(S)$ is then given by $\sum_{ij\in\delta(S)} w_{ij}$. The max-cut problem is to find a cut $\delta(S)$ with the maximum weight. Note that each $x \in \{-1, 1\}^n$ corresponds to a partition that divides V into $S = \{i \in V \mid x_i = 1\}$ and $V \setminus S = \{i \in V \mid x_i = -1\}$. We can now express the max-cut problem as the following binary quadratic problem,

$$(Max\text{-}Cut) \qquad \max \frac{1}{2} \sum_{1\le i<j\le n} w_{ij}(1 - x_ix_j)$$
$$\text{s.t. } x \in \{-1, 1\}^n.$$

While all the weights in the conventional definition for the max-cut problem considered in graph theory are assumed to be nonnegative, we consider here a more general setting of the max-cut problem without confining the weights to be nonnegative.

This chapter aims to give a systematic survey of the polynomially solvable subclasses of (0-1QP) and its variants studied in the literature and to report some recent progress in this

subject. Our goal is to present a self-contained writing and to provide step-by-step examples and geometric illustrations in an effort to capture the essence of the polynomial solvability of binary quadratic programming problems. In Section 2.2, we discuss the problem $(0\text{-}1QP)$ with all off-diagonal elements of Q being non-positive. This subclass of problems has been known for long time to be polynomially solvable due to the total unimodularity of the constraint matrix in its linear integer programming reformulation. Its relation to the maximum flow problem is also discussed. In Section 2.3, we analyze the polynomial solvability of problem $(0\text{-}1QP_h)$ with a fixed rank Q using the properties of zonotope in discrete geometry. The relationship between zonotope and hyperplane arrangement is exploited to derive an efficient procedure to enumerate all extreme points of a zonotope. Sections 2.4 and 2.5 devote to problems defined by a special graph or a logic circuit. Relations between the polynomial solvability and the special properties of the series-parallel graph and logic circuit are studied. We investigate in Section 2.6, a possible zero duality gap between problem (BQP) and its SDP relaxation. A sufficient condition for the polynomial solvability of (BQP) via the SDP relaxation is presented. We conclude this chapter in Section 8 with a brief summary.

2.2 Problem $(0\text{-}1QP)$ with all off-diagonal elements of Q being non-positive

Consider a subclass of problem $(0\text{-}1QP)$ where all off-diagonal elements of Q are non-positive. It is easy to see that $x_ix_j = \min(x_i, x_j)$ when $x_i,\ x_j \in \{0, 1\}$. Since $x_i^2 = x_i$, we can assume, without of loss of generality, $q_{ii} = 0$, $i = 1, \ldots, n$. Let $z_{ij} = x_ix_j$. If $q_{ij} \leq 0$ for $1 \leq i < j \leq n$, then $(0\text{-}1QP)$ is equivalent to the following linear integer programming problem:

$$\min \sum_{i=1}^{n} c_ix_i + 2 \sum_{1\leq i<j\leq n} q_{ij}z_{ij} \tag{1}$$

$$\text{s.t. } z_{ij} \leq x_i,\ 1 \leq i < j \leq n, \tag{2}$$

$$z_{ij} \leq x_j,\ 1 \leq i < j \leq n, \tag{3}$$

$$x_i,\ x_j,\ z_{ij} \in \{0, 1\},\ 1 \leq i < j \leq n. \tag{4}$$

Consider the linear programming relaxation of the above problem by replacing constraint (4) with

$$x_i,\ x_j,\ z_{ij} \in [0, 1],\ 1 \leq i < j \leq n. \tag{5}$$

Recall that a matrix $A = (a_{ij})$ is called *totally unimodular* (TU) if every square sub-matrix of A has determinant $+1$, -1 or 0. It is well known that a linear programming problem with a totally unimodular constraint matrix and an integral right-hand side has an integral optimal solution. Recall also that a matrix A is TU if (i) $a_{ij} \in \{+1, -1, 0\}$ for all i, j; (ii) Each column contains at most two nonzero coefficients ($\sum_{i=1}^{m} |a_{ij}| \leq 2$); and (iii) There exists a partition (M_1, M_2) of the set M consisting of the rows of A such that each column j contains two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$.

Note that the constraint matrix in the linear programming relaxation problem (1)-(3) and (5) is of the form $\begin{pmatrix} C \\ I \end{pmatrix}$ where C comes from these inequalities of $z_{ij} \leq x_i$ and $z_{ij} \leq x_j$. It suffices to show C is TU as a matrix A is TU iff $(A^T, I)^T$ is TU. Recall that a matrix A is TU iff A^T is TU. Note that there is one 1 and one -1 in each row of C and the third sufficient condition mentioned above can be satisfied by selecting $M_1 = C$ and $M_2 = \emptyset$.

In conclusion, $(0\text{-}1QP)$ with all off-diagonal elements of Q being non-positive can be reduced to a linear programming problem and thus can be solved in polynomial time [40][52].

The polynomial solvability of this subclass of $(0\text{-}1QP)$ can be also shown by associating the problem with a graph and reducing the problem to a maximum flow problem. Consider a directed graph $G = (V, E)$ with $V = (s, 1, 2, \ldots, n, t)$, where s denotes the source and t the sink, and with $E = E_s \cup E_Q \cup E_t$, where

$$E_s = \{sj \mid j = 1, \ldots, n\},$$
$$E_Q = \{ij \mid q_{ij} < 0,\ 1 \leq i < j \leq n\},$$
$$E_t = \{jt \mid j = 1, \ldots, n\}.$$

The capacities of the arcs in E are defined as follows:

$$e_{sj} = \max\{0, -2 \sum_{i=j+1}^{n} q_{ji} - c_j\}, \quad sj \in E_s, \tag{6}$$
$$e_{ij} = -2q_{ij}, \quad ij \in E_Q, \tag{7}$$
$$e_{jt} = \max(0, 2 \sum_{i=j+1}^{n} q_{ji} + c_j), \quad jt \in E_t. \tag{8}$$

Let $(U, \overline{U})$ be a partition of G with $s \in U$ and $t \in \overline{U}$. The set of arcs $\delta^+(U) = \{ij \mid i \in U,\ j \in \overline{U}\}$ is called an $s - t$ cut. The capacity of $\delta^+(U)$ is $\sum_{ij \in \delta^+(U)} e_{ij}$. The *minimum-cut*

problem is to find a cut with the minimum capacity. Let Ψ be the capacity of the minimum-cut of G. Then $\Psi = \min_U \sum_{ij\in\delta^+(U)} e_{ij}$. Associate each cut $\delta^+(U)$ of G with a 0-1 vector $(1, x_1, \ldots, x_n, 0)$ satisfying $x_i = 1$ if $i \in U$ and $x_i = 0$ otherwise. Similar to the proof for Property 6 in [30], we prove the following result which is also stated in [50].

Theorem 2.2.1 *Problem* (0-1QP) *with all off-diagonal elements of* Q *being non-positive can be reduced to the minimum-cut problem of the graph* $G = (V, E)$ *via the following relation:*

$$\min_{x\in\{0,1\}^n}\{x^TQx + c^Tx\} = \Psi - \sum_{j=1}^{n} e_{sj}.$$

Proof. By (6)-(8), we have

$$\begin{aligned}
\Psi &= \min_{x\in\{0,1\}^n}\{\sum_{j=1}^{n} e_{sj}(1-x_j) + \sum_{1\le i<j\le n} e_{ij}x_i(1-x_j) + \sum_{j=1}^{n} e_{jt}x_j\} \\
&= \sum_{j=1}^{n} e_{sj} + \min_{x\in\{0,1\}^n}\{\sum_{j=1}^{n}\min(0, 2\sum_{i=j+1}^{n} q_{ji} + c_j)x_j - 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}x_i + 2\sum_{1\le i<j\le n} q_{ij}x_ix_j \\
&\quad + \sum_{j=1}^{n}\max(0, 2\sum_{i=j+1}^{n} q_{ji} + c_j)x_j\} \\
&= \sum_{j=1}^{n} e_{sj} + \min_{x\in\{0,1\}^n}\{\sum_{j=1}^{n}(2\sum_{i=j+1}^{n} q_{ji} + c_j)x_j - 2\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} q_{ij}x_i + 2\sum_{1\le i<j\le n} q_{ij}x_ix_j\} \\
&= \sum_{j=1}^{n} e_{sj} + \min_{x\in\{0,1\}^n}\{\sum_{j=1}^{n} c_jx_j + 2\sum_{1\le i<j\le n} q_{ij}x_ix_j\} \\
&= \sum_{j=1}^{n} e_{sj} + \min_{x\in\{0,1\}^n}\{x^TQx + c^Tx\}.
\end{aligned}$$

This proves the theorem. □

It is well-known that the minimum-cut problem is equivalent to the maximum-flow problem that can be solved in polynomial time (see [47]). Therefore, problem (0-1QP) with all off-diagonal elements of Q being non-positive can be solved by computing the maximum-flow of a graph with $n + 2$ vertices and $2n + n(n - 1)/2$ arcs. Algorithms with different complexity bounds have been proposed for finding a maximum-flow in G (see e.g., [36][39][47]), for example, an $O(n^3)$ maximum-flow algorithm proposed in [36] or [39].

2.3 Problem $(0\text{-}1QP_h)$ with fixed rank Q

We consider now a subclass of problem $(0\text{-}1QP_h)$ where Q is negative semidefinite and $rank(Q)$ = d. Let $G = -Q$. In this situation, there exists a row full rank $d \times n$ matrix, V, such that G = V^TV, where the rows of V are suitably scaled eigenvectors of G. Problem $(0\text{-}1QP_h)$ can be thus expressed as

$$(BQP_{fr}) \qquad \max_{x\in\{0,1\}^n} x^TGx = x^TV^TVx = \sum_{i=1}^{d}(v_ix)^2,$$

where v_i is the i-th row of matrix V.

If d is equal to 1, i.e., the matrix G is of rank one with $G = v_1^Tv_1$, the solution to (BQP_{fr}) can be easily found by inspection. More specifically, we only need to select x such that the absolute value of v_1x is maximized on $\{0,1\}^n$.

In general cases with $rank(G) = d > 1$, we consider a linear map Φ: $x \in \mathbb{R}^n \to z = Vx \in \mathbb{R}^d$, in which Φ maps the hypercube $[0,1]^n$ into a convex polytope $Z(V) = \Phi([0,1]^n) = \{z \in \mathbb{R}^d \mid z = Vx,\ x \in [0,1]^n\}$, known as a *zonotope*. Note that

$$\max_{x\in\{0,1\}^n} x^TGx = \max_{x\in\{0,1\}^n} \sum_{i=1}^{d}(v_ix)^2 = \max_{z\in Z(V)} \sum_{i=1}^{d} z_i^2 = \max_{z\in Z(V)} \|z\|^2,$$

where the second equality is due to that the maximization of a convex function over a convex set is always achieved at the vertices. Based on the same argument, the convex function $\|z\|^2$ achieves its maximum over the convex set $Z(V)$ at some extreme point $\tilde{z}$. Thus, (BQP_{fr}) reduces to a problem of finding the maximum norm in a zonotope.

Theorem 2.3.1 *For any extreme point $\tilde{z}$ of the zonotope $Z(V)$, there is a point $\tilde{x} \in \{0,1\}^n$ such that $\tilde{z}$ = $V\tilde{x}$.*

Proof. Since V is row full rank, we can assume that $V = (\hat{V}, V_1)$, where $\hat{V}$ is a $d \times d$ nonsingular matrix. Let $x = \begin{pmatrix} \hat{x} \\ \bar{x} \end{pmatrix}$, where $\hat{x}$ is a d-dimensional vector corresponding to the columns of $\hat{V}$. Letting $\bar{x} = 0$ in the equation $\tilde{z} = Vx$, we obtain $\tilde{z} = \hat{V}\hat{x}$. Then $\tilde{x} = \begin{pmatrix} \hat{V}^{-1}\tilde{z} \\ 0 \end{pmatrix}$ satisfies $\tilde{z}$ = $V\tilde{x}$ and is an extreme point of $[0,1]^n$. Indeed, suppose that there exist $\tilde{x}_1$ and $\tilde{x}_2$ with $\tilde{x}_1 \neq \tilde{x}_2$ such that $\tilde{x} = \lambda\tilde{x}_1 + (1-\lambda)\tilde{x}_2$ for some $\lambda \in (0,1)$. Then

$\tilde{x}_1 = \begin{pmatrix} \hat{x}_1 \\ 0 \end{pmatrix}$ and $\tilde{x}_2 = \begin{pmatrix} \hat{x}_2 \\ 0 \end{pmatrix}$ for some $\hat{x}_1,\ \hat{x}_2 \in [0,1]^d$ with $\hat{x}_1 \neq \hat{x}_2$. Thus, $\tilde{z} = \lambda \hat{V}\hat{x}_1 + (1-\lambda)\hat{V}\hat{x}_2$. Since $\hat{V}$ is nonsingular and $\hat{x}_1 \neq \hat{x}_2$, we deduce that $\hat{V}\hat{x}_1,\ \hat{V}\hat{x}_2 \in Z(V)$ and $\hat{V}\hat{x}_1 \neq \hat{V}\hat{x}_2$, which in turns implies that $\tilde{z}$ is not an extreme point of $Z(V)$, a contradiction. □

The following is a classical result in discrete geometry (see, e.g., [56]) which gives a polynomial upper bound of the number of extreme points of $Z(V)$ for fixed d.

Theorem 2.3.2 *Let $N_{ep}(Z)$ denote the number of extreme points of the zonotope $Z(V)$. Then $N_{ep}(Z) = O(n^{d-1})$.*

An immediate implication of Theorems 2.3.1 and 2.3.2 is that problem (0-1QP_h) with fixed rank Q is polynomially solvable.

We now discuss how to enumerate all the extreme points of the zonotope $Z(V)$. Let v^j denote the jth column vector of V. Assume that the regularity condition is satisfied for the zonotope $Z(V)$, i.e., each column of V is nonzero and $v^i \neq kv^j$ for any $i \neq j$ and $k \neq 0$. Associated with $Z(V)$, we define a set of hyperplanes in $\mathbb{R}^d$ with v^j $(j = 1, \ldots, n)$ being normal vectors:

$$\mathcal{A}(V) = \{h_j \mid j = 1, \ldots, n\},$$

where $h_j = \{y \in \mathbb{R}^d \mid (v^j)^T y = 0\}$ for $j = 1, \ldots, n$. The set $\mathcal{A}(V)$ is called *central arrangement* of V. Denote $h_j^+ = \{y \in \mathbb{R}^d \mid (v^j)^T y > 0\}$ and $h_j^- = \{y \in \mathbb{R}^d \mid (v^j)^T y < 0\}$. For any $c \in \mathbb{R}^d$, define the location vector $\gamma(c) \in \{+, 0, -\}^n$ by

$$\gamma(c)_j = \begin{cases} +, & \text{if } c \in h_j^+, \\ 0, & \text{if } c \in h_j, \\ -, & \text{if } c \in h_j^-. \end{cases}$$

Let $c \in \mathbb{R}^d$ be such that $\gamma(c)_j \neq 0$ for $j = 1, \ldots, n$. A *cell* of the arrangement $\mathcal{A}(V)$ is defined as the following d-dimensional subset:

$$C_c = \{y \in \mathbb{R}^d \mid \gamma(y) = \gamma(c)\}. \tag{9}$$

Obviously, C_c is invariant for any $y \in C_c$. Thus, a cell can be represented by its sign vector. Denote by $C(V)$ the set of all cells of the arrangement $\mathcal{A}(V)$, i.e.,

$$C(V) = \{C_c \mid c \in \mathbb{R}^d\}.$$

For any cell $C_c \in C(V)$, denote $\gamma^+(c) = \{j \mid \gamma(c)_j = +\}$ and $\gamma^-(c) = \{j \mid \gamma(c)_j = -\}$.

Theorem 2.3.3 *There is a one-to-one correspondence between the extreme points of $Z(V)$ and the cells of $\mathcal{A}(V)$.*

Proof. For each cell $C_c \in C(V)$, define x_c by

$$(x_c)_j = \begin{cases} 1, & \text{if } j \in \gamma^+(c) \\ 0, & \text{if } j \in \gamma^-(c). \end{cases} \tag{10}$$

Let $z_c = Vx_c$. Then $c^T z_c = \sum_{j\in\gamma^+(c)} c^T v^j$. Since $c^T v^j > 0$ for $j \in \gamma^+(c)$ and $c^T v^j < 0$ for $j \in \gamma^-(c)$, z_c is the unique optimal solution to the linear program $\max_{z\in Z(V)} c^T z$. Thus z_c is an extreme point of the polytope $Z(V)$. Conversely, for any extreme point $\tilde{z}$ of $Z(V)$, there is a $c \in \mathbb{R}^d$ such that $\tilde{z}$ is the unique optimal solution to the linear program $\max_{z\in Z(V)} c^T z$. Notice that

$$\max_{z\in Z(V)} c^T z = \max_{x\in[0,1]^n} \sum_{j=1}^{n} x_j (c^T v^j).$$

So $\tilde{z}$ must be of the form Vx_c with x_c being defined in (10). There must be no j such that $c^T v^j = 0$, i.e., $\gamma(c)_j \neq 0$ for any j, since otherwise the optimal solution to the linear program $\max_{z\in Z(V)} c^T z$ is not unique. The cell C_c defined in (6.2) is then the cell in $C(V)$ corresponding to $\tilde{z}$. The one-to-one property of the above correspondence can be easily established by noting that V is row full rank. □

Theorem 2.3.3 implies that enumeration of all the extreme points of the zonotope $Z(V)$ is equivalent to the enumeration of all the cells of the arrangement $\mathcal{A}(V)$ for which various procedures have been proposed (see [23][24][35][54]).

Note that the central arrangement $\mathcal{A}(V)$ satisfies $\cap_{j=1}^n h_j = \{0\}$ and the cells of $\mathcal{A}(V)$ are symmetric to the origin. We thus only need to generate half of the cells or the corresponding sign vectors. Consider a shift of the last hyperplane $h = \{x \in \mathbb{R}^d \mid (v^n)^T y = b\}$, where $b \neq 0$. The intersection of $\mathcal{A}(V)$ and h is a general arrangement of $n-1$ hyperplanes in $\mathbb{R}^{d-1}$. It can be seen that the sign vectors (cells) of $\mathcal{A}'(V) = \mathcal{A}(V) \cap h$ corresponds to the half of the sign vectors of $\mathcal{A}(V)$ with the last element being + or $-$.

Now, consider a general arrangement $\mathcal{A} = \{h_j \mid j = 1, \ldots, m\}$, where $h_j = \{y \in \mathbb{R}^d \mid a_j^T y = b_j,\ j = 1, \ldots, m\}$. The sign vector of a cell in a general arrangement can be defined

similarly as for the central arrangement. A *root* cell is the cell with all + elements in the sign vector. A root cell can be found by selecting any cell and reversing the orientation of some of the hyperplanes if necessary. Two cells are called neighbors if only one of the hyperplanes separates them, i.e., the sign vectors differ only in exactly one element. A *parent* cell of c is a unique neighbor of c which contains one more + in its sign vector. Any cell with c being its parent is called a *child* of c. If a unique parent of each cell (except for the root cell) is assigned, then a directed tree structure can be obtained for the cells and the reverse search algorithm can be used to traverse this tree backward, enumerating all the cells exactly once. A procedure to search all the adjacent cells of a cell c is needed in the reverse search algorithm. The procedure of cell enumeration can be described as follows.

Procedure 1 (Cell Enumeration)

Input: a cell c represented by its sign vector, and the hyperplanes represented by (A, b)

Output: a set $C(A)$ containing all the cells of the arrangement (rooted at c)

begin

(i) output c to $C(A)$.

(ii) call a subroutine to list all adjacent cells of c

(iii) **for** each cell e of c **do**

if c is the unique parent of e **then**

recurse the procedure with e as the input cell

endif

endfor

end

The above recursive procedure starts from the root cell and terminates when all the cells are enumerated. The details of the procedures for finding all neighbors of a cell and searching for the unique parent of a cell can be found in [24][54]. To illustrate the cell enumeration procedure, let's consider an instance of (0-1QP_h) where $Q = -V^T V$ and

$$V = \begin{pmatrix} -1 & -1 & 0 & 1 & 0 \\ -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

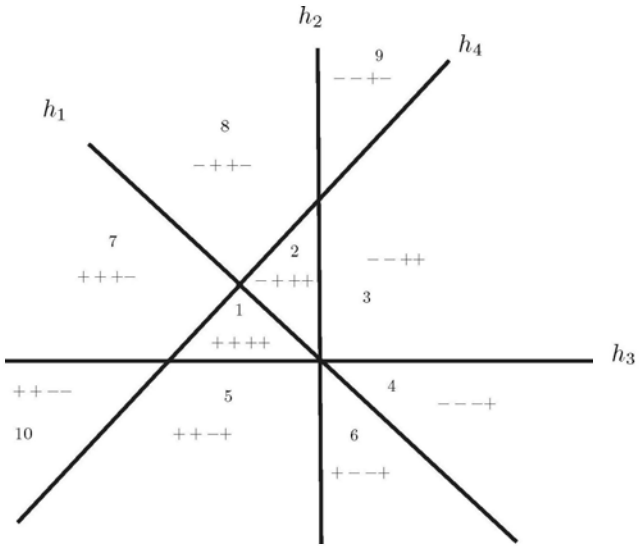


Figure 2.1: Illustration for cell enumeration process

Using the parallel translation of the last hyperplane of the arrangement $y_3 = 1$, the reduced general arrangement contains 4 hyperplanes in $\mathbb{R}^2$ and is represented by

$$A = \begin{pmatrix} -1 & -1 & 0 & 1 \\ -1 & 0 & 1 & -1 \end{pmatrix}, \quad b = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Figure 2.1 illustrates the cell enumerating process of the arrangement (A, b), where each cell is represented by its sign vector and the number indicates the order of the cell enumeration process in Procedure 1.

As there are 10 cells in the reduced general arrangement, there are 20 cells in the original central arrangement. Thus, the zonotope $Z(V)$ has 20 extreme points among which $z_c = Vx_c$, where $x_c = (1, 1, 0, 1, 0)^T$, is the optimal solution to $\max_{z \in Z(V)} \|z\|^2$. Therefore, $x_c = (1, 1, 0, 1, 0)^T$ is the optimal solution to the original problem (0-1QP_h) with optimal value 6.

2.4 Problem (BQP) defined by a series-parallel graph

We consider graph $G = (E, V)$. Given a subset of vertex $T \subset V$, we use $G[T]$ to denote an induced subgraph of G, where it consists of T and all edges whose endpoints are contained in

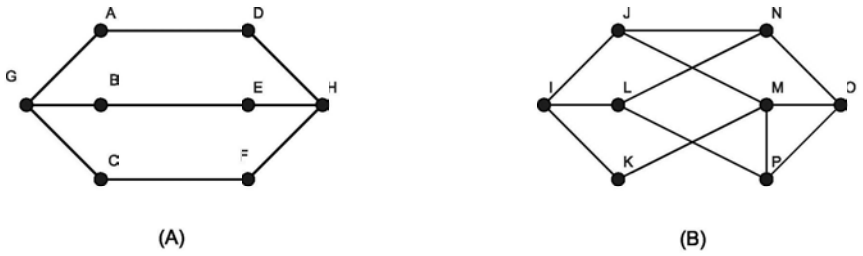


Figure 2.2: Examples of series-parallel and non-series-parallel graphs

T. For any node $v \in V$, the degree of v is the cardinality of cut $\delta(\{v\})$, denoted as $deg(v)$.

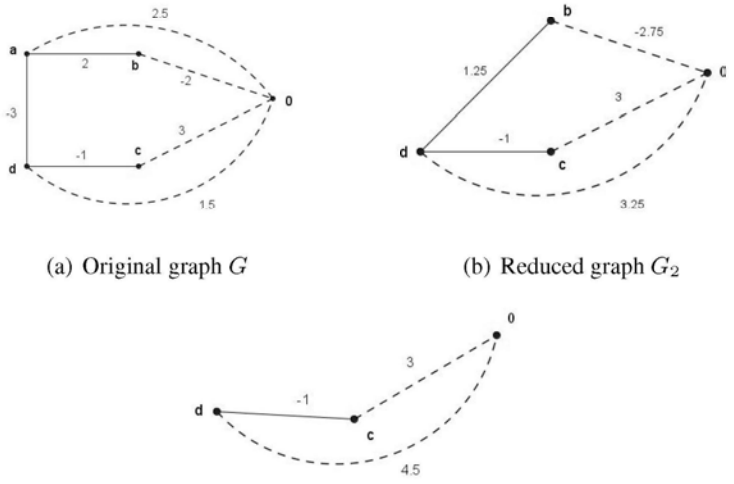
Given two edge sets $E_1 \subset E$ and $E_2 \subset E$ in graph G such that $E_1 \cap E_2 = \emptyset$, we use $\beta(E_1, E_2, G)$ to denote the weight of a cut $\delta(U)$ such that $E_1 \subset \delta(U)$ and $E_2 \cap \delta(U) = \emptyset$ and the weight of such a cut, $w(\delta(U))$, is maximized in G. Therefore, $\beta(E_1, E_2, G)$ can be interpreted as a *constrained* max-cut that must include all edges in E_1 and does not include any edge in E_2. Furthermore, $\beta(\emptyset, \emptyset, G)$, for short $\beta(G)$, actually is the weight of the max-cut of graph G. Note $w(\delta(\emptyset))$ = 0.

We use K_n to denote the complete graph with n vertices, where all n vertices are pairwise adjacent. A graph G is a contractible to G', if G' can be obtained from G by a sequence of elementary contractions, in which edge ij is replaced by a single vertex whose incident edges are the edges other than ij that were incident to i or j. The multiple edges arising from the contraction are merged into a single edge in such a procedure. A graph is called *series-parallel* if it is not contractible to K_4. Graph (A) in Figure 2.2 is series-parallel and (B) is not.

We consider problem (BQP) and reduce it first to a max-cut problem. Define a graph $G(Q) := \{V, E\}$ for problem (BQP), which is associated to $Q = \{q_{ij}\}_{n\times n}$, as follows:

$$V = \{1, 2, \cdots, n\},$$
$$ij \in E \Leftrightarrow q_{ij} \neq 0,$$
$$w_{ij} = 2q_{ij},$$

where w_{ij} is the weight assigned to edge ij. We then construct a new graph $G(Q, c)$ by adding a universal vertex $\{0\}$ which is connected to each vertex of $G(Q)$ and assign weight $w_{0j} = c_j$



(a) Original graph G

(b) Reduced graph G_2

(c) Reduced graph G_3

Figure 2.3: The original graph and the reduced graphs of the example instance

to edge $0j$, for j = 1, ..., n. Clearly, $G(Q) = G(Q, c) \setminus \{0\}$. Then, solving (BQP) is equivalent to finding the max-cut of graph $G(Q, c)$:

$$\begin{aligned} \max \quad & \sum_{i=0}^{n-1} \sum_{j=i+1}^{n} \{w_{ij} | y_i = -y_j\} \\ \text{s.t.} \quad & y_i^2 = 1, \ \text{ for } i = 0, \cdots, n. \end{aligned}$$

Consider an instance of (BQP) with

$$Q = \begin{pmatrix} 0 & 1 & 0 & -1.5 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 \\ -1.5 & 0 & -0.5 & 0 \end{pmatrix}, \quad c = \begin{pmatrix} 2.5 \\ -2 \\ 3 \\ 1.5 \end{pmatrix}$$

The correspondent graph of this example problem is given in Figure 2.3(a). It is easy to check that graph $G(Q)$ in this example is series-parallel.

If graph $G(Q)$ is series-parallel, then graph $G(Q, c)$ is not contractible to K_5. Recall the facts [25] that any sub-graph of a series-parallel graph is still series-parallel and there always exists a vertex in a series-parallel graph that has degree not greater than 2. The main result in [25] is that if graph $G(Q)$ is series-parallel, the corresponding max-cut problem of graph $G(Q, c)$ can be solved by a linear-time algorithm which we are presenting below.

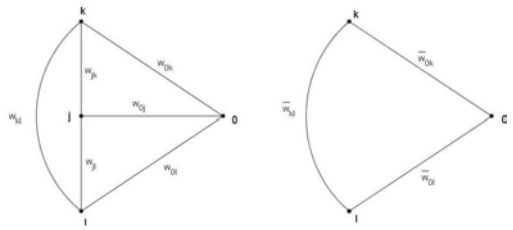


Figure 2.4: The graph of $\{j, k, l, 0\}$

If graph $G(Q, q)$ is of 3 vertices or less, the max-cut problem can be solved by enumeration. Otherwise, for every vertex i in $G(Q)$, we compute its degree d_i and place all vertices with degree not greater than 2 into a list L, which can be achieved in linear time $O(n)$. In each iteration, we choose a vertex j from L and perform a reduction. We need to consider the following three different situations.

Case 1. If $deg(j) = 2$, let k and l be the vertices adjacent to j in $G(Q)$. We assume that $G(Q, q)$ contains all three edges $0k$, $0l$ and kl. Otherwise, we can add the missing edge with weight 0. Let W be the subgraph of $G(Q, q)$ induced by $\{0, j, k, l\}$ with edge weights the same as in $G(G, q)$. See the left subgraph of Figure 2.4 for graphical presentation of subgraph W. Note that any cut of W either contains 2 edges of $0k$, $0l$ and kl, or none of them.

The max-cut problem is solved by recursively generating $G' := G(Q, q) \setminus \{j\}$. All the edge weights in $G(Q, q) \setminus \{j\}$ are the same as in $G(Q, q)$, except for these edges in subgraph W, $0k$, kl, and $0l$, which need to be modified. For the reduced graph $W' = W \setminus \{j\}$ depicted in the right subgraph of Figure 2.4, there are only three possible cuts, $\{0k, lk\}$, $\{0l, lk\}$ and $\{0k, 0l\}$. All of such cuts have to satisfy the following balance equations,

$$\begin{aligned}
\bar{w}_{0l} + \bar{w}_{0k} &= \beta(\{0l, 0k\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W), \\
\bar{w}_{0k} + \bar{w}_{lk} &= \beta(\{0k, kl\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W), \\
\bar{w}_{0l} + \bar{w}_{lk} &= \beta(\{0l, kl\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W).
\end{aligned}$$

The meaning of the above equations is clear. For example, the weight of the cut $\{0k, kl\}$ in the reduced graph W' should be equal to that of the max cut involving $\{0k, kl\}$ in

the original graph W, while taking away the contribution of the edges leading to node j, $\beta(\emptyset, \{0k, kl, 0l\}, W)$. The solution to the above system of linear equations is

$$\begin{aligned}
\bar{w}_{0l} &:= 0.5[\beta(\{0l, kl\}, \emptyset, W) + \beta(\{0l, 0k\}, \emptyset, W) \\
&\quad -\beta(\{0k, kl\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W)], \\
\bar{w}_{0k} &:= 0.5[\beta(\{0k, kl\}, \emptyset, W) + \beta(\{0l, 0k\}, \emptyset, W) \\
&\quad -\beta(\{0l, kl\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W)], \\
\bar{w}_{lk} &:= 0.5[\beta(\{0l, kl\}, \emptyset, W) + \beta(\{0k, kl\}, \emptyset, W) \\
&\quad -\beta(\{0k, 0l\}, \emptyset, W) - \beta(\emptyset, \{0k, kl, 0l\}, W)].
\end{aligned}$$

It is evident that $\beta(G(Q, q)) = \beta(G') + \beta(\emptyset, \{0k, kl, 0l\}, W)$. The optimal cut in G' is extended to an optimal cut in $G(Q, q)$ by taking the appropriate cut in W. Set then $deg(l) = deg(l) - 1$ and $deg(k) = deg(k) - 1$. If $deg(l) \leq 2$ or $deg(k) \leq 2$, add l or k to L.

Case 2. If $deg(j) = 1$, let k be the vertex adjacent to j in $G(Q)$. Let W be the subgraph of $G(Q, q)$ induced by $\{0, j, k\}$, in which the weights are the same as in $G(Q, q)$. In $G' := G(Q, q) \setminus \{j\}$, we only need to modify the weight of edge $0k$ to

$$\bar{w}_{0k} := \beta(\{0k\}, \emptyset, W) - \beta(\emptyset, \{0k\}, W).$$

It is clear $\beta(G(Q, q) = \beta(G') + \beta(\emptyset, \{0k\}, W)$. Set $deg(k) = deg(k) - 1$. If $deg(k) \leq 2$, we include k in L.

Case 3. If $deg(j) = 0$, the problem can be solved in $G' := G(Q, q) \setminus \{j\}$ and in the subgraph induced by $\{j, 0\}$, separately.

In any of the above three cases, we reduce the nodes of the graph by one in each iteration. If the size of $G(Q, q)$ is n, the computational effort needed by this algorithm is bounded by $O(n)$.

We now illustrate the above solution scheme for the example given in Figure 2.3(a).

Step 1 The initial list is given by $L := \{a, b, c, d\}$. As $deg(a) = 2$, we consider a reduced graph $G_2 = G \setminus \{a\}$ given in Figure 2.3(b). Let subgraph W_1 be induced by vertices $\{a, b, d, 0\}$.

We calculate $\beta(\{0b, 0d\}, \emptyset, W_1)$ based on its definition. Consider two possible cuts in W_1 that include edges $0b$, $0d$ in subgraph H, $\{ab, ad, 0b, 0d\}$ and $\{0a, 0b, 0d\}$. Thus,

$$\beta(\{0b, 0d\}, \emptyset, W_1) = \max\{(2 - 3 - 2 + 1.5), (2.5 - 2 + 1.5)\} = 2.$$

Similarly, we can get $\beta(\{0b, bd\}, \emptyset, W_1) = 0$, $\beta(\{0d, bd\}, \emptyset, W_1) = 6$ and $\beta(\emptyset, \{0b, bd, 0d\}, W_1) = 1.5$. Furthermore, the modified weights for $0b$, bd and $0d$ are given as

$$\begin{aligned}
\bar{w}_{0b} &= 0.5[\beta(\{0b, bd\}, \emptyset, W_1) + \beta(\{0b, 0d\}, \emptyset, W_1) - \beta(\{0d, bd\}, \emptyset, W_1) \\
&\quad -\beta(\emptyset, \{0b, bd, 0d\}, W_1)] = -2.75, \\
\bar{w}_{bd} &= 0.5[\beta(\{0b, bd\}, \emptyset, W_1) + \beta(\{0d, bd\}, \emptyset, W_1) - \beta(\{0b, 0d\}, \emptyset, W_1) \\
&\quad -\beta(\emptyset, \{0b, bd, 0d\}, W_1)] = 1.25, \\
\bar{w}_{0d} &= 0.5[\beta(\{0d, bd\}, \emptyset, W_1) + \beta(\{0d, 0b\}, \emptyset, W_1) - \beta(\{0b, bd\}, \emptyset, W_1) \\
&\quad -\beta(\emptyset, \{0b, bd, 0d\}, W_1)] = 3.25.
\end{aligned}$$

We also have

$$\beta(G) = \beta(G_1) + \beta(\emptyset, \{0b, bd, 0d\}, W_1).$$

After deleting a, the node list is updated to $L := \{b, d, c\}$.

Step 2 As $deg(b) = 1$ in graph G_2, we consider a reduced graph $G_3 = G_2 \setminus \{b\}$ given in Figure 2.3(c). Let subgraph W_2 be induced by vertices $\{b, d, 0\}$. We have

$$\begin{aligned}
&\beta(\{0d\}, \emptyset, W_2) = 4.5, \quad \beta(\emptyset, \{0d\}, W_2) = w(\delta(\emptyset)) = 0, \\
&\bar{w}_{0d} = \beta(\{0d\}, \emptyset, W_2) - \beta(\emptyset, \{0d\}, W_2) = 4.5.
\end{aligned}$$

It is clear that $\beta(G_2) = \beta(G_3) + \beta(\emptyset, \{0d\}, W_2)$.

Step 3 There are only 3 vertices in G_3. Comparing all possible cuts yields $\beta(G_3) = 7.5$ with max cut $\{0c, 0d\}$. Tracing back gives rise,

$$\begin{aligned}
\beta(G_2) &= \beta(G_3) + \beta(\emptyset, \{0d\}, W_2) = 7.5 + 0 = 7.5, \\
\beta(G) &= \beta(G_2) + \beta(\emptyset, \{0b, bd, 0d\}, W_1) = 7.5 + 1.5 = 9.
\end{aligned}$$

The remaining problem is how to identify the optimal solution to the primal problem. As the max cut in G_3 gives rise an optimal division as $(\{c, d\}, \{0\})$. Comparing two possible

"expanding" divisions of nodes in G_2, $(\{d, c\}, \{b, 0\})$ and $\{d, c, b\}, \{0\})$ yields the optimal division in G_2, $(\{d, c\}, \{b, 0\})$. Finally, comparing two possible "expanding" divisions of nodes in G, $(\{a, d, c\}, \{b, 0\})$ and $(\{d, c\}, \{a, b, 0\})$ identifies the optimal division of the entire problem, $(\{a, d, c\}, \{b, 0\})$.

We indicate here that the solution process dictated by the above graphical method can be also produced by the basic algorithm which is also applicable to binary situations with $x \in \{-1, 1\}^n$. Expressing $f(x)$ as

$$\begin{aligned} f_4(x_1, x_2, x_3, x_4) &= 2x_1x_2 - 3x_1x_4 - x_3x_4 + 2.5x_1 - 2x_2 + 3x_3 + 1.5x_4 \\ &= x_1\Delta_4(x_2, x_3, x_4) + \Theta_4(x_2, x_3, x_4), \end{aligned}$$

where $\Delta_4 = 2x_2 - 3x_4 + 2.5$ and $\Theta_4 = -x_3x_4 - 2x_2 + 3x_3 + 1.5x_4$, we have

$$\phi_4(x_2, x_3, x_4) = \frac{1}{2}(1 - x_2)(1 + x_4) - 1,$$

which leads to a reduced form of the objective function

$$\begin{aligned} f_3(x_2, x_3, x_4) &= \phi_4(x_2, x_3, x_4)\Delta_4(x_2, x_3, x_4) + \theta_4(x_2, x_3x_4) \\ &= 1.25x_2x_4 - x_3x_4 - 2.75x_2 + 3x_3 + 3.25x_4 - 3.75. \end{aligned}$$

Note that the graphical representation of the max-cut problem corresponding to $f_3(x_2, x_3, x_4)$ is exactly Figure 2.3(b). We further write f_3 in the following form,

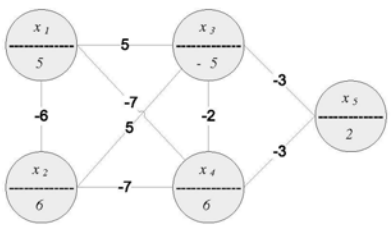
$$f_3(x_2, x_3, x_4) = x_2\Delta_3(x_3, x_4) + \Theta_3(x_3, x_4),$$

with $\Delta_3 = 1.25x_4 - 2.75$ and $\Theta_3 = -x_3x_4 + 3x_3 + 3.25x_4 - 3.75$. We can derive $\phi_3(x_3, x_4) = 1$ which yields

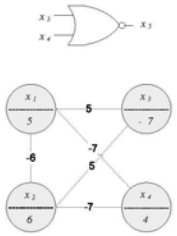
$$f_2(x_3, x_4) = 3x_3 + 4.5x_4 - x_3x_4 - 6.5,$$

whose graphical representation is exactly Figure 2.3(c). Minimizing $f_2(x_3, x_4)$ yields $x_3^* = -1$ and $x_4^* = -1$. We can then determine $x_2^* = \phi_3(x_3, x_4) = 1$ and $x_1^* = \phi_4(x_2, x_3, x_4) = -1$.

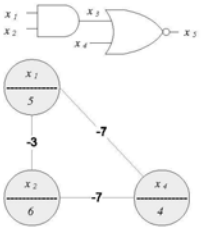
When the corresponding graph of problem (BQP) is serial-parallel, there are at least one row and one column in Q that have no more than two non-zero elements. This pattern remains unchanged during the reduction process. As ϕ_k is at most a quadratic function, f_k remains to be a quadratic function. In essence, if the structure of (BQP) is governed by a serial-parallel graph, the coupling among x_i's is low, and the problem can be solved efficiently by the basic algorithm.



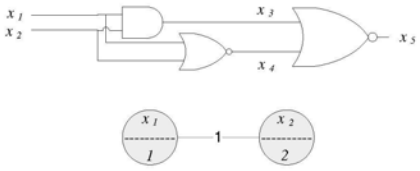
(a) Original neural network



(b) Reduced neural network after 1 iteration



(c) Reduced neural network after 2 iterations



(d) Reduced neural network after 3 iterations

Figure 2.5: The original and reduced neural networks of the example problem

2.5 Problem $(0\text{-}1QP)$ defined by a logic circuit

Let $w_{ij} = -2q_{ij}$, $I_i = -c_i$ for $i, j = 1, 2 \ldots, n$. The objective function $f(x) = x^T Qx + c^T x$ in $(0\text{-}1QP)$ can be expressed as the following form

$$E(x) = -\sum_{1 \le i < j \le n} w_{ij} x_i x_j - \sum_{i=1}^{n} I_i x_i,$$

which can be viewed as the energy function of a neural network where $w_{ij} \in \mathbb{R}$ is the weight associated with the connection between neurons j and i, $x_i \in \{0, 1\}$ is the activation value of neuron i, and $I_i \in \mathbb{R}$ is the threshold of neuron i. For example, the following objective function

$$\begin{aligned} f = \quad & -[-6x_1x_2 + 5x_1x_3 - 7x_1x_4 + 5x_2x_3 - 7x_2x_4 - 2x_3x_4 - 3x_3x_5 - 3x_4x_5] \\ & -[5x_1 + 6x_2 - 5x_3 + 6x_4 + 2x_5], \end{aligned}$$

can be expressed as the energy function of the neural network in Figure 2.5(a).

It can be verified easily for the example in Figure 2.5(a) that for any value of x_3 and x_4, we should assign x_5 at $x_3 \bar{\vee} x_4 = 1 - \max\{x_3, x_4\}$, i.e., optimal x_5 which minimizes the energy function should be the output of a NOR logic gate if we assign x_3 and x_4 to be the inputs of the gate. This conclusion can be also derived from our earlier discussion of the basic algorithm.

Let x_j and x_k be the inputs to a logic gate. Then x_i is the output of an AND logic gate if $x_i = x_j \wedge x_k = \min\{x_j, x_k\}$, the output of an OR logic gate if $x_i = x_j \vee x_k = \max\{x_j, x_k\}$, the output of a NAND logic gate if $x_i = x_j \bar{\wedge} x_k = 1 - \min\{x_j, x_k\}$, and the output of a NOR logic gate if $x_i = x_j \bar{\vee} x_k = 1 - \max\{x_j, x_k\}$. We can now relate the following special form of the 3-variable energy function,

$$E(x_i, x_j, x_k) = -[w(x_i x_j + x_i x_k) + w_{jk} x_j x_k] - [I x_i + I_j x_j + I_k x_k] + K,$$

with these 4 different logic gates. Using the basic algorithm, we can identify eight cases of different combinations of w and I and their corresponding logic gates, which are given in the following table. Figure 2.6 offers details in figuring out these eight cases. For example, both conditions of $w < 0$ and $-w - I < 0 < -2w - I$ give rise case 6.

x_j	x_k	$-w(x_j + x_k) - I$	Cases 1 & 5	Case 2	Case 3	Cases 4 & 8	Case 6	Case 7
0	0	$-I$	< 0	≥ 0	≥ 0	≥ 0	< 0	< 0
0	1	$-w - I$	< 0	< 0	≥ 0	≥ 0	< 0	≥ 0
1	0	$-w - I$	< 0	< 0	≥ 0	≥ 0	< 0	≥ 0
1	1	$-2w - I$	< 0	< 0	< 0	≥ 0	≥ 0	≥ 0
		Logic Gate		OR	AND		NAND	NOR
		$\phi(x_j, x_k)$	1	$x_j + x_k - x_j x_k$	$x_j x_k$	0	$1 - x_j x_k$	$(1 - x_j)(1 - x_k)$

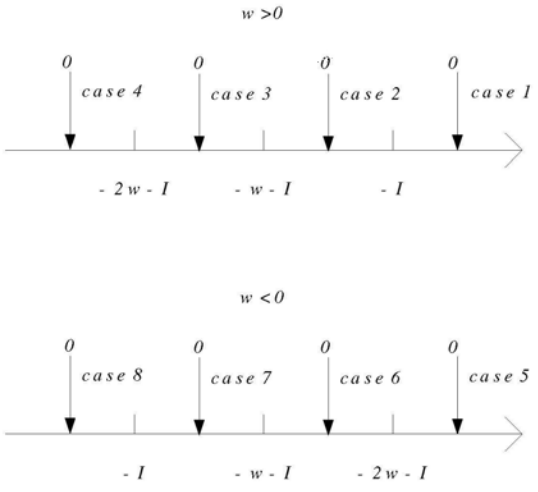


Figure 2.6: Eight cases of different combinations of w and I.

Replacing x_i by $\phi(x_j, x_k)$ in the 4 cases associated with different digital logic gates yields

a reduced form for the energy function,

$$\bar{E}(x_j, x_k) = -\bar{w}_{jk}x_jx_k - [\bar{I}_jx_j + \bar{I}_kx_k] + \bar{K},$$

where the calculation of $\bar{w}_{jk}$, $\bar{I}_j$ and $\bar{I}_k$ is summarized in the following table.

AND	$\bar{I}_j = I_j$, $\bar{I}_k = I_k$ and $\bar{w}_{jk} = w_{jk} + 2w + I$
OR	$\bar{I}_j = I_j + w + I$, $\bar{I}_k = I_k + w + I$ and $\bar{w}_{jk} = w_{jk} - I$
NAND	$\bar{I}_j = I_j + w$, $\bar{I}_k = I_k + w$ and $\bar{w}_{jk} = w_{jk} - 2w - I$
NOR	$\bar{I}_j = I_j - I$, $\bar{I}_k = I_k - I$ and $\bar{w}_{jk} = w_{jk} + I$

Based on the above recognition between problem $(0\text{-}1QP)$ and logic circuits, Chakradhar and Bushnell have designed an iterative method [31] to check whether or not a neural network corresponding to $(0\text{-}1QP)$ can be converted into a logic circuit. If we are able to construct a logic circuit such that all the consistent input/output values together minimize the energy function of the neural network, then the original problem $(0\text{-}1QP)$ can be solved by a linear time algorithm.

The assumptions to ensure that the quadratic function f can be transformed into a combinational logic circuit are (i) The neural network corresponding to the energy function and all the reduced neural networks generated during the iteration have at least one vertex of degree one or two; and (ii) Both edges incident to the vertex with degree two have equal weights.

A satisfaction of the above assumptions will enable us, in each iteration, to identify a vertex with degree one or two by uniquely determining the corresponding logic gate.

Let us now apply this solution scheme to the example problem in Figure 2.5(a). As the terms involving x_5 satisfy the condition of NOR logic gate with x_3 and x_4 being the inputs and x_5 being the output: w_{35} = w_{45} = w = $-3 < 0$, w_{l5} = 0, for $l \neq 3$ and 4, $-I$ = $-2 < 0 < -w - I$ = 1, we express x_5 as $(1 - x_3)(1 - x_4)$, resulting in the reduced neural network given in Figure 2.5(b).

We find in Figure 2.5(b) that x_1, x_2 and x_3 satisfy the condition of AND logic gate with x_1 and x_2 being the inputs and x_3 being the output: w_{13} = w_{23} = w = $-5 > 0$, w_{34} = 0, and $-2w - I$ = $-3 < 0 < -w - I$ = 2. Expressing x_3 as x_1x_2 results in the reduced neural network given in Figure 2.5(c).

From Figure 2.5(d), we can figure out x_1, x_2 and x_4 satisfy the condition of NOR logic gate with x_1 and x_2 being the inputs and x_4 being the output: w_{14} = w_{24} = w = $-7 < 0$, $-I$

= $-4 < 0 < -w - I$ = 3. Expressing x_4 as $(1 - x_1)(1 - x_2)$ results in the reduced neural network given in Figure 2.5(d).

Solving the reduced binary quadratic minimization problem,

$$\min -x_1x_2 - x_1 - 2x_2,$$

yields x_1^* = 1 and x_2^* = 1. Further calculation gives $x_3^* = x_1^*x_2^* = 1$, $x_4^* = (1 - x_1^*)(1 - x_2^*) = 0$ and $x_5^* = (1 - x_3^*)(1 - x_4^*) = 0$.

The condition to define problem (0-1QP) by a logic circuit is very strict, especially the requirement of the same weights of the edges incident to the vertex of degree two which is to be removed. If problem (0-1QP) can be defined by a logic circuit, matrix Q and its reduced forms generated during the process all have, at least, one row and one column that have no more than two non-zero elements, and when there are two, these two elements are the same. These conditions are stronger than the conditions for problems defined by the series-parallel graph.

2.6 SDP representation of Lagrangian dual and polynomial solvability

Based on our recent finding in [55], we discuss in this section how to identify a polynomially solvable subclass of (BQP) using Lagrangian dual. Notice that (P) can be rewritten as

$$\begin{aligned} (BQP_c) \qquad & \min f(x) = x^TQx + c^Tx \\ & \text{s.t. } x_i^2 - 1 = 0,\ i = 1, \ldots, n. \end{aligned}$$

Dualizing each $x_i^2 - 1 = 0$ by a multiplier λ_i, we get the Lagrangian relaxation problem (L_λ):

$$\begin{aligned} d(\lambda) &= \inf_{x \in \mathbb{R}^n} L(x, \lambda) := f(x) + \sum_{i=1}^{n} \lambda_i(x_i^2 - 1) \\ &= \inf_{x \in \mathbb{R}^n} \{x^T(Q + \mathrm{diag}(\lambda))x + c^Tx - e^T\lambda\}, \end{aligned} \tag{11}$$

where $e = (1, \ldots, 1)^T$ and $\mathrm{diag}(\lambda)$ denotes the diagonal matrix with λ_i being its ith diagonal element. Obviously, the weak duality holds

$$d(\lambda) \le f(x), \quad \text{for any } x \in \{-1, 1\}^n.$$

The dual problem of (P_c) (or (BQP)) is

$$(D) \quad \max_{\lambda \in \mathbb{R}^n} d(\lambda).$$

Notice that the dual problem (D) can be rewritten as

$$v(D) = \max_{\lambda \in \mathbb{R}^n} d(\lambda) = \max_{\lambda \in \mathbb{R}^n} \inf_{x \in \mathbb{R}^n} \{x^T[Q + \mathrm{diag}(\lambda)]x + c^T x - e^T\lambda\},$$

which has an equivalent form:

$$\begin{aligned} v(D) \quad = \quad & \max_{(\lambda,\tau) \in \mathbb{R}^{n+1}} \quad -\tau \\ & \text{s.t.} \;\; x^T[Q + \mathrm{diag}(\lambda)]x + c^T x - e^T\lambda \geq -\tau, \quad x \in \mathbb{R}^n. \end{aligned} \tag{12}$$

Let function $g(x)$ be the constraint in problem (12),

$$g(x) = x^T[Q + \mathrm{diag}(\lambda)]x + c^T x - e^T\lambda + \tau.$$

Using homogeneous quadratic form (see [53] and Section 3.4 in [28]), we show below that $g(x) \geq 0, \forall\, x \in \mathbb{R}^n$, the satisfaction of the constraint in problem (12), is equivalent to

$$G(x,t) = (x^T, t) \begin{pmatrix} Q + \mathrm{diag}(\lambda) & \frac{1}{2}c \\ \frac{1}{2}c^T & \tau - e^T\lambda \end{pmatrix} \begin{pmatrix} x \\ t \end{pmatrix} \geq 0, \quad \forall\, (x,t) \in \mathbb{R}^{n+1},$$

which holds true if and only if

$$\begin{pmatrix} Q + \mathrm{diag}(\lambda) & \frac{1}{2}c \\ \frac{1}{2}c^T & \tau - e^T\lambda \end{pmatrix} \succeq 0.$$

Since $g(x) = G(x,1)$, $G(x,t) \geq 0$ for all $(x,t) \in \mathbb{R}^{n+1}$ implies $g(x) \geq 0$ for all $x \in \mathbb{R}^n$. Now, suppose that $g(x) \geq 0$ for all $x \in \mathbb{R}^n$. Then, $g(t^{-1}x) \geq 0$ for all $x \in \mathbb{R}^n$ and $t \neq 0$, which implies

$$t^{-2}x^T[Q + \mathrm{diag}(\lambda)]x + t^{-1}c^T x - e^T\lambda + \tau \geq 0, \; \forall x \in \mathbb{R}^n, \; t \neq 0,$$

or equivalently,

$$G(x,t) = x^T[Q + \mathrm{diag}(\lambda)]x + c^T xt + (\tau - e^T\lambda)t^2 \geq 0, \;\; \forall x \in \mathbb{R}^n, \; t \neq 0.$$

By continuity, we have

$$G(x,t) = x^T[Q + \mathrm{diag}(\lambda)]x + c^T xt + (\tau - e^T\lambda)t^2 \geq 0, \;\; \forall\, (x,t) \in \mathbb{R}^{n+1}.$$

Thus, the dual problem (D) can be expressed by the following equivalent semidefinite programming formulation,

$$(D_{SDP}) \qquad \max_{(\lambda,\tau)\in\mathbb{R}^{n+1}} \ -\tau \qquad (13)$$
$$\text{s.t.} \ \begin{pmatrix} Q+\text{diag}(\lambda) & \frac{1}{2}c \\ \frac{1}{2}c^T & \tau - e^T\lambda \end{pmatrix} \succeq 0.$$

Since (D_{SDP}) is a semidefinite programming problem, it is polynomially solvable. The above discussion implies that if there is no duality gap between (BQP) and (D_{SDP}), i.e., $v(BQP) = v(D) = v(D_{SDP})$, then $v(BQP)$ is polynomially computable.

The following theorem further gives a sufficient condition for the polynomial solvability of (BQP).

Theorem 2.6.1 *Assume that the optimal solution λ^* to (D_{SDP}) satisfies $Q^* = Q+\text{diag}(\lambda^*) \succ 0$. Then $x^* = -\frac{1}{2}(Q^*)^{-1}c$ is the unique optimal solution to (BQP) and $v(BQP) = v(D) = v(D_{SDP})$. Moreover, (BQP) is polynomially solvable.*

Proof. From [27], we know that, for any $\lambda \in \mathbb{R}^n$, $d(\lambda) > -\infty$ with x solving (L_λ) if and only if

(i) $Q + \text{diag}(\lambda) \succeq 0$;

(ii) $(Q + \text{diag}(\lambda))x = -\frac{1}{2}c$.

Since the optimal solution λ^* to (D_{SDP}) satisfies $Q^* \succ 0$, we can verify that (D) or (D_{SDP}) is equivalent to the following problem,

$$(D_1) \qquad \sup \ \Phi(\lambda) = -\frac{1}{4}c^T(Q+\text{diag}(\lambda))^{-1}c - e^T\lambda \qquad (14)$$
$$\text{s.t.} \ \ Q + \text{diag}(\lambda) \succ 0.$$

Thus, λ^*, an interior point of the feasible region of (D_1), also solves (D_1). By KKT theorem, we must have $\nabla\Phi(\lambda^*) = 0$, where Φ is defined in (14). Calculating the gradient of Φ at λ^* and setting it at zero yield the following,

$$\frac{1}{4}c^T(Q^*)^{-1}\text{diag}(e_i)(Q^*)^{-1}c = 1, \quad i = 1,\ldots,n. \qquad (15)$$

This is to say $(x_i^*)^2 = 1$, for all $i = 1,\ldots,n$. Thus $x^* \in \{-1,1\}^n$. As $Q^* \succ 0$, x^* is the unique optimal solution to (BQP) and $v(BQP) = v(D) = v(D_{SDP}) = v(D_1)$. Moreover,

since λ^* is polynomially computable and $x^* = -\frac{1}{2}(Q + \mathrm{diag}(\lambda^*))^{-1}c$, we deduce that (BQP) is polynomially solvable. □

2.7 Summary

We have summarized the state-of-the-art of polynomially solvable cases for binary quadratic programming problems. Separating certain easy subclasses from a general NP-hard class facilitates identification schemes to peel off hard covers of some seemingly intractable, but actually manageable, binary quadratic programming problems. Furthermore, investigation of this subject not only helps us better understand inherent nature of the problem, but also stimulates innovative thinking for development of solution schemes for general binary quadratic programming problems.

□ End of chapter.

Chapter 3

Geometric Solution Approach to Binary Quadratic Programming Problem

3.1 Introduction

We consider in this chapter an exact solution method to the following unconstrained quadratic 0-1 programming problem:

$$(P) \qquad \min_{x\in\{0,1\}^n} f(x) = \frac{1}{2}x^T Qx + c^T x,$$

where Q is an $n \times n$ symmetric matrix and $c \in \mathbb{R}^n$. Without loss of generality, we assume in the sequel that $f(x)$ is strongly concave, i.e., Q is negative definite. Otherwise, we can always rewrite $f(x)$ in an equivalent strongly concave form using the property $x_i^2 = x_i$, for $x_i \in \{0, 1\}$. We assume that $n > 2$ and the eigenvalues of matrix Q are ranked in an ascending order: $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n < 0$.

We focus in this chapter on development of an exact solution method of a branch-and-bound type for solving (P). It is well-known that the efficiency of a branch-and-bound method for solving a general integer minimization problem largely depends on the quality of the lower bound and upper bound estimation and the corresponding computational efforts to obtain them. Existing approaches in the literature for computing a lower bound of (P) include simple lower bound estimation and improvements via reformulations, convex quadratic programming re-

laxation, roof duality, decomposition method and semi-definite programming relaxation. Our approach here is based on the geometric features of the ellipse contour of a concave quadratic function. As a special, albeit important, case of integer programming, binary quadratic programming problem processes rich geometric properties. Although there exist rich geometric properties in binary quadratic programming, only a few papers, e.g. [18], have devoted to explore such prominent features hidden behind until recently. An exact solution method has been recently proposed in [45] by exploring these rich geometric properties .

Another significant contributing factor to the efficiency of the solution algorithms for zero-one optimization, although received much less attentions in the literature, is the capability of variable fixation to fix certain variables at their optimal values based on some optimality conditions. The bounds of the gradient of the objective function can be used to fix variables. The inscribed sphere inside the ellipse contour of the objective function can be also employed to fix variables.

3.2 Perturbed Quadratic Function and Contour

In this section, we investigate basic properties of the contour of the perturbed quadratic objective function.

Define the perturbed quadratic objective function as follows,

$$f_\mu(x) = \frac{1}{2}x^T(Q - \mu I)x + (c + \frac{\mu e}{2})^T x, \tag{1}$$

where μ is a positive parameter, I is an $n \times n$ identity matrix, and e is an n-dimensional vector with all elements equal to 1.

Since $x_i^2 = x_i$ for any $x_i \in \{0, 1\}$, it can be easily seen that $f(x) = f_\mu(x)$ on $\{0, 1\}^n$. Thus, the following perturbed problem is equivalent to (P):

$$(P_\mu) \qquad \min_{x \in \{0,1\}^n} f_\mu(x).$$

When setting μ large enough, e.g., $(\lambda_n - \mu) < 0$ or

$$\mu - q_{ii} \geq \sum_{j=1, j \neq i}^{n} |q_{ij}|, \;\; i = 1, \ldots, n,$$

matrix $(Q - \mu I)$ will be negative definite. Therefore in the following sections, we always assume $f(x)$ is concave.

Since Q is a real symmetric matrix, an orthogonal matrix P which consists of eigenvectors of Q can be found such that $P^T P = I$ and $P^T Q P = \Lambda = diag\{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ with all $\lambda_i < 0,\ i = 1, \ldots, n$. Note that $f(x)$ can be written as

$$f(x) = \frac{1}{2}(x - x^0)^T Q (x - x^0) - \frac{1}{2} c^T Q^{-1} c, \tag{2}$$

where $x^0 = -Q^{-1}c$. Also note that $\{x \in R^n | f(x) \geq v\}$ forms an ellipsoid for any $v < -\frac{1}{2}c^T Q^{-1} c$ and the center of the ellipse $f(x) = v$ is x^0. It is clear that any point in $\{0,1\}^n$ outside the ellipse $f(x) = v$ possesses an objective value less than v.

3.3 Upper Bound

Here we introduce our new method to find an upper bound for (P). Suppose that the incumbent, the best solution found so far, is $\tilde{x}$ and the current best upper bound is $\tilde{v} = f(\tilde{x})$. The next question is how to search for a solution better than $\tilde{x}$. If the incumbent is not the optimal solution to the original problem, there must exist an intersection between the ellipse contour $f(x) = \tilde{v}$ and the box $[0,1]^n$. Note that any outer normal vector on the ellipse $f(x) = \tilde{v}$ represents a descent direction. Therefore, if we can find a point $\bar{x}$ located on the ellipse contour $f(x) = \tilde{v}$ and within the box $[0,1]^n$, the normal vector of this point directs to a better integer point.

If any point on the objective contour$f(x) = \tilde{v}$ is inside the box $[0,1]^n$, this point provides a descent direction. We thus propose to solve the following problem to identify the point on $f(x) = \tilde{v}$ that is closest to the center of the box $[0,1]^n$.

$$\begin{aligned} \min \quad & \textstyle\sum (x_i - 0.5)^2 \\ s.t. \quad & f(x) = \tilde{v}, \\ & x_i \in [0,1]. \end{aligned} \tag{3}$$

See Fig. 3.1 for an illustration.

3.3.1 The Nearest Point to Center Point $\frac{e}{2}$

Dropping the constraint, $x \in [0,1]^n$, the above problem can be solved almost analytically. Consider $f(x) = \frac{1}{2}x^TQx + c^Tx,\ x \in \{0,1\}^n$. Let $x - x_0 = Py$, we have $f(x) = g(y) = \frac{1}{2}\sum_{i=1}^n \lambda_i y_i^2 - \frac{1}{2}c^TQ^{-1}c$.

Let $\alpha = P^T(\frac{e}{2} - x_0)$. To find the point on the contour of $f(x) = \tilde{v}$ that is closest to $(\frac{1}{2}, \frac{1}{2}, \ldots, \frac{1}{2})^T$ is equivalent to solving the following optimal problem.

$$\begin{array}{ll} \min & \sum(y_i - \alpha_i)^2 \\ s.t. & \sum_{i=1}^n \lambda_i y_i^2 = 2\tilde{v} + c^TQ^{-1}c. \end{array} \tag{4}$$

The Lagrangian function of (4) is

$$L(y,\mu) = \sum(y_i - \alpha_i)^2 + \nu(\sum_{i=1}^n \lambda_i y_i^2 - 2\tilde{v} - c^TQ^{-1}c), \tag{5}$$

where ν is the Khun-Tucker multiplier. We have the following KKT Conditions,

$$\begin{aligned} &y_i + \nu\lambda_i y_i = \alpha_i,\ i = 1, \ldots, n, \\ &\textstyle\sum_{i=1}^n \lambda_i y_i^2 = 2\tilde{v} + c^TQ^{-1}c. \end{aligned}$$

The above necessary conditions yield $y_i = \frac{\alpha_i}{1+\nu\lambda_i}, i = 1, \ldots, n$, and the value of ν can be found by solving the following nonlinear equation by some numerical method,

$$\sum_{i=1}^n \frac{\lambda_i \alpha_i^2}{(1+\nu\lambda_i)^2} = 2\tilde{v} + c^TQ^{-1}c. \tag{6}$$

From the second-order necessary condition, we must have

$$\nu \leq \frac{-1}{\lambda_n}$$

It is not difficult to find out that $\sum_{i=1}^n \frac{\lambda_i \alpha_i^n}{(1+\nu\lambda_i)^2}$ is a strictly decreasing function of ν in $(-\infty, \frac{-1}{\lambda_n}]$. Thus, there is a unique solution of (6) in $(-\infty, \frac{-1}{\lambda_n}]$.

The first equation in the KKT Conditions actually states that the direction $\alpha - y$ is proportional to the norm vector on the surface of $g(y) = \tilde{v}$ at y,

$$\alpha - y = \gamma\Lambda y,$$

where γ is a constant.

There is an obvious problem of the above approach: the point on the contour that is closest to the center point $\frac{e}{2}$ could be outside the box $[0,1]^n$ in certain situations. For example, in Fig. 3.1, the point on $f(x) = f(0,0)$ that is closest to the center, $[-0.0455, 0.5640]$, is outside of the box $[0,1]^2$. In such situations, we can not take advantage of the information of the point to find a better binary point. To overcome this problem and to find $\bar{x}$ under this situation, we developed the following iterative method.

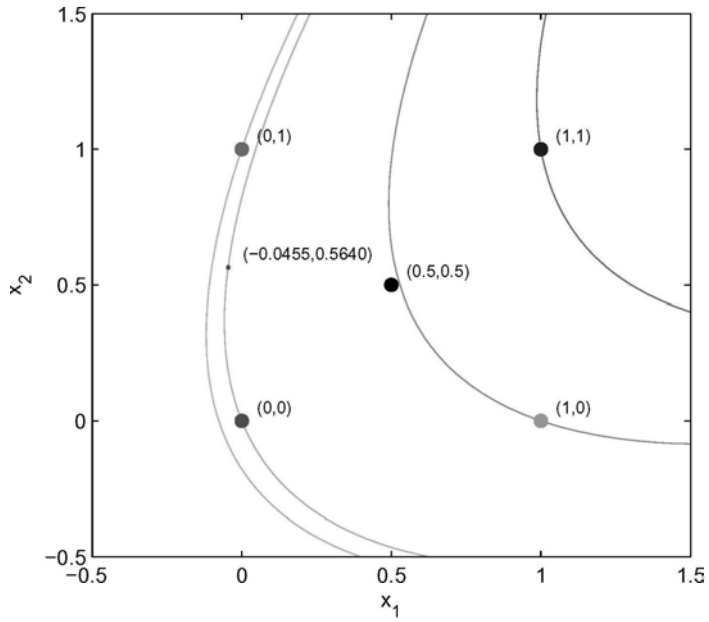


Figure 3.1: The closest point on the contour with respect to the center point is not with the box $[0,1]^2$

3.3.2 An iterative method to find point $\bar{x}$

Suppose that an initial point x_0 is located on the current ellipse contour and x_0 is not inside the box $[0,1]^n$. To find a point that is located on the contour and inside the box, we design the following iterative method. First, we compute the tangent plane of the contour at the initial point x_0. Then we can calculate the project point x_P at the tangent plane with respect to the center point $\frac{e}{2}$. Connecting the center point and the project point x_P with a straight line, there exists an intersection point x_I on the contour. Check the property of these two points x_P and x_I. If one of them inside the box, we find $\bar{x}$. Otherwise, let $x_0 = x_I$ and repeat the above

procedure.

Example 3.3.1 *We consider a problem with* $f(x) = \frac{1}{2}x^TQx + c^Tx$, *where*

$$Q = \begin{pmatrix} -16 & 43 \\ 43 & -123 \end{pmatrix},\ c = (46, 35)^T.$$

Suppose that the incumbent is $(0,0)^T$ *and we locate an initial point on the ellipse contour, i.e.* $x_0 = (-0.0622, 0.1142)^T$. *Since* x_0 *is not within the box* $[0,1]^n$, *we apply the above iterative method to find point* $\bar{x}$. *As shown in Fig. 3.2(a), with* $x_0 = (-0.0622, 0.1142)^T$ *we can figure out both* $x_p = (-0.1210, 0.2814)^T$ *and* $x_I = (-0.0753, 0.2975)^T$. *Since both* x_P *and* x_I *are located outside of the box* $[0,1]^n$, *we set* $x_0 = x_P = (-0.1210, 0.2814)^T$ *and repeat this procedure. In Fig. 3.2(b), it is shown that,* $x_P = (-0.0554, 0.5447)^T$ *and* $x_I = (-0.0064, 0.5407)^T$ *in the second iteration. Due to the same reason that neither* x_P *nor* x_I *is located within the box* $[0,1]^n$, *we set* $x_0 = x_P = (-0.0554, 0.5447)^T$ *and perform the third iteration. In the third iteration, we figure out* $x_P = (0.0661, 0.6989)^T$ *and* $x_I = (0.0773, 0.6937)^T$ *as shown in Fig. 3.2(c). Since both* x_P *and* x_I *are within the box* $[0,1]^n$, *we stop the procedure and set* $\bar{x} = x_I = (0.0773, 0.6937)^T$. *Fig. 3.2 demonstrates the whole process to find the point* $\bar{x}$.

The general procedure to calculate x_P and x_I is given now as follows,

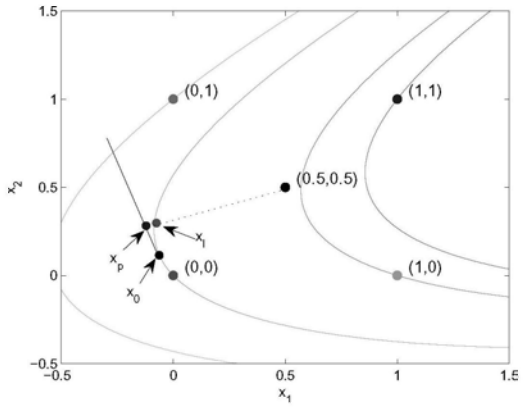
Suppose that $a^Tx = b$ is the expression of the tangent plane at x_0. Finding the project point x_P at the tangent with respect to the center point $\frac{e}{2}$ is equivalent to solving the following optimal problem:

$$\begin{aligned} \min \quad & \tfrac{1}{2} \parallel x_P - \tfrac{e}{2} \parallel^2 \\ s.t. \quad & a^Tx_P = b \end{aligned} \tag{7}$$

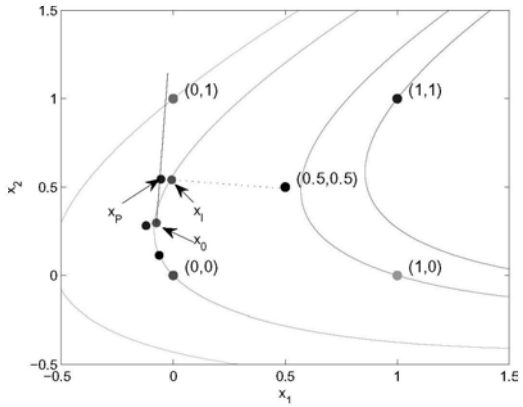
Solving this optimization problem, we get $x_P = \frac{e}{2} - \lambda a$, where $\lambda = \frac{\frac{1}{2}a^Te - b}{\|a\|^2}$. The intersection point x_I can be expressed as $x_I = \frac{1}{2}\beta e + (1-\beta)x_P,\ (0 \le \beta \le 1)$. Since x_I is located on the contour $f(x) = \tilde{v}$, we have $f(x_I) = \tilde{v}$. Substitute $x_I = \frac{1}{2}\beta e + (1-\beta)x_P$ into the above equation, we get a quadratic equation according to β. Solve this equation, we can get the value of β and further the value of x_I.

More specifically, suppose that the function of the current contour is given as follows

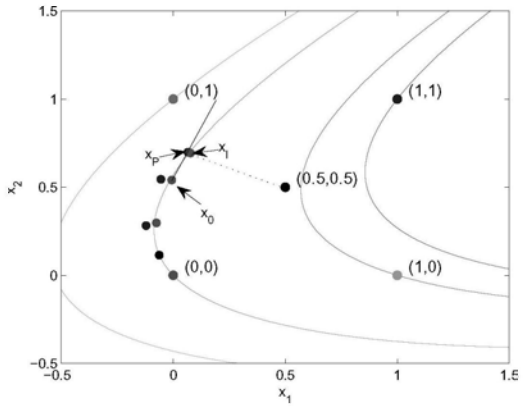
$$\frac{1}{2}x_I^TQx_I + c^Tx_I = \tilde{v} \tag{8}$$



(a) first iteration



(b) second iteration



(c) third iteration

Figure 3.2: Finding the point $\bar{x}$ with iterative method

Substitute $x_I = \beta x + (1-\beta)x_P$ into the above equation gives rise

$$\frac{1}{2}(\beta x + (1-\beta)x_P)^T Q(\beta x + (1-\beta)x_P) + c^T(\beta x + (1-\beta)x_P) = \tilde{v}$$

Writing out this equation we get the following expression:

$$\begin{aligned} &\tfrac{1}{2}(x^TQx - x^TQx_P - x_P^TQx + x_P^TQx_P)\beta^2 \\ &+(\tfrac{1}{2}x^TQx_P + \tfrac{1}{2}x_P^TQx - x_P^TQx_P + c^Tx + c^Tx_P)\beta \\ &+\tfrac{1}{2}x_P^TQx_P + c^Tx_P = \tilde{v} \end{aligned} \tag{9}$$

Let $A = \frac{1}{2}(x^TQx - x^TQx_P - x_P^TQx + x_P^TQx_P)$, $B = (\frac{1}{2}x^TQx_P + \frac{1}{2}x_P^TQx - x_P^TQx_P + c^Tx + c^Tx_P)$ and $C = \frac{1}{2}x_P^TQx_P + c^Tx_P - \tilde{v}$, Eq.(9) can be simplified as $A\beta^2 + B\beta + C = 0$. This is a quadratic equation in one variable. Solving this equation, we can get two roots of β. Choose the root within the range $[0, 1]$ and substitute it into $x_I = \beta x + (1-\beta)x_P$, then we get the result of x_I.

3.3.3 The Upper Bound Achieved from the Point $\bar{x}$

After finding the point $\bar{x}$, which is on the ellipse $f(x) = \tilde{v}$ and within the box $[0, 1]^n$, the gradient of $f(x) = \tilde{v}$ at $\bar{x}$ can be calculated as $Q\bar{x} + c$. The next incumbent, $\hat{x}$, can be identified a follows,

$$\hat{x}_j = \begin{cases} = 1, & \text{if } (Q\bar{x} + c)_j < 0, \\ = 0 \text{ or } 1, & \text{if } (Q\bar{x} + c)_j = 0, \quad j = 1, \ldots, n. \\ = 0, & \text{if } (Q\bar{x} + c)_j > 0, \end{cases} \tag{10}$$

Example 3.3.2 *Consider the problem stated in Example 3.3.1. It is clear from Fig.3.3 that the optimal solution is* $(0, 1)^T$. *Suppose that the incumbent is* $(0, 0)^T$. *The normal vector at the point* $\bar{x} = (0.0773, 0.6937)^T$, $(74.5923, -47.0012)^T$, *defines the shadow region. Note that any integer point inside the shadow region, in our case* $(0, 1)^T$, *is a better solution.*

The following optimality condition is obvious.

Theorem 3.1 *A solution* $\tilde{x} \in \{0, 1\}^n$ *is an optimal solution to if and only if there is no point in* $\{0, 1\}^n$ *outside the ellipsoid* $E = \{x \in R^n | f(x) \geq \tilde{v}\}$, *where* $\tilde{v} = f(\tilde{x})$.

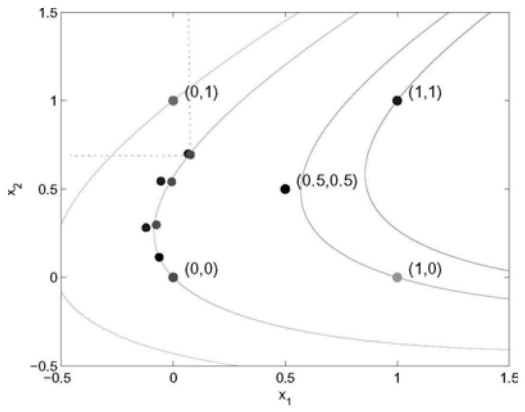


Figure 3.3: Normal vector on the ellipse and better solution

3.4 Lower Bounds

In this section, we discuss how to compute a good lower bound of the optimal value of (P). We first introduce a lower bound derived from the maximum distance sphere. Two new lower bounds are then derived by exploiting the geometric properties of the ellipse contour of $f(x)$.

3.4.1 Lower bound derived from the maximum distance sphere

The center of the ellipse contour

$$E(\tilde{v}) = \{x \in R^n | f(x) = \tilde{v}\}$$

is

$$x^0 = -Q^{-1}c.$$

Ranking the integer points in $\{0,1\}^n$ according to a descending order with respect to their distance to x^0 yields

$$x^1, x^2, x^3, \ldots$$

with corresponding distance

$$r^1 \geq r^2 \geq r^3 \geq \ldots$$

where

$$r^i = \left|\left| x^i - x^0 \right|\right|.$$

Let $\underline{f}^i$ be the contour level such that $E(\underline{f}^i) = \{x \in R^n | f(x) = \underline{f}^i\}$ is the minimum circumscribed contour containing the ball $S_i = \{x \in R^n | \left|\left| x - x^0 \right|\right|^2 = r_i^2\}$, we have the following theorem:

Theorem 3.2 *$\underline{f}^i = \frac{\lambda_1}{2} r_i^2 - \frac{1}{2} c^T Q^{-1} c$, for $i = 1, 2, \ldots$. In addition, $\underline{f}^1$ is a lower bound of (P).*

Proof. Consider $f(x) = \frac{1}{2} x^T Q x + c^T x,\ x \in \{0,1\}^n$. Let $x - x_0 = Py$, we have $f(x) = g(y) = \frac{1}{2} \sum_{i=1}^n \lambda_i y_i^2 - \frac{1}{2} c^T Q^{-1} c$. It can be easily seen that the intersection points between the ellipse and the sphere in y coordinates are $y = (r_i, 0, \ldots, 0)$ and $y = (-r_i, 0, \ldots, 0)$. Substitute y into $g(y) = \frac{1}{2} \sum_{i=1}^n \lambda_i y_i^2 - \frac{1}{2} c^T Q^{-1} c$, we can easily get $\underline{f}^i = \frac{\lambda_1}{2} r_i^2 - \frac{1}{2} c^T Q^{-1} c$. Since r^1 is the distance between the center of the ellipse and the farthest 0-1 point. All the 0-1 points are located inside this sphere and thus located inside the ellipse. As a result, $\underline{f}^1$ is a lower bound of (P). □

3.4.2 Lower Bound Based on the kth Farthest Point

The following algorithm provides a lower bound to Problem (P) based on the kth farthest point to center point of the ellipse ($k \geq 2$), which is better than $\underline{f}^1$.

Algorithm 3.4.1

Step 1 Calculate x^1, r^1 and $\underline{f}^1$. Let $x^* = x^1$ and $\bar{f} = f(x^*)$. Let $k = 2$.

Step 2 Calculate x^k, r^k and $\underline{f}^k$. If $\bar{f} \leq \underline{f}^k$, stop and x^* is the optimal solution.

step 3 Let

$$x^* = \begin{cases} x^*, \text{ if } f(x^*) \leq f(x^k) \\ x^k, \text{ if } f(x^k) < f(x^*) \end{cases}$$

$$\bar{f} = f(x^*)$$

Step 4 If $k = 2^n$, stop and x^* is the optimal solution. Let $k = k + 1$ and go back to Step 2.

Theorem 3.4.1 *Algorithm 3.4.1 identifies optimal solution of (P) in finite steps.*

Proof. From Step 3, we know that x^* is the incumbent solution. If the algorithm stops at Step 2, as all points, $x^k, x^{k+1}, \ldots, x^{2^n}$, process objective values larger than or equal to $\underline{f}^k$, and $\bar{f}$ corresponds the best solution among $x^1, x^2, \ldots, x^{k-1}$, condition $\bar{f} \leq f^k$ implies that incumbent x^* is optimal. If the algorithm stops at Step 4, the algorithm does an exhaustive search before reaching the conclusion. □

Example 3.4.1 *We consider a problem with $f(x) = \frac{1}{2}x^T Qx + c^T x$, where*

$$Q = \begin{pmatrix} -10 & 3 \\ 3 & -10 \end{pmatrix}, \; c = (-5, 2.5)^T.$$

By applying Step 1 in Algorithm 3.4.1, we get $x^1 = (1, 1)^T$, $r^1 = 1.7159$ and $\underline{f}^1 = -17.8342$ as shown in Fig. 3.4(a). Let $x^* = x^1 = (1, 1)^T$, $\bar{f} = f(x^*) = -9.5$ and $k = 2$.

Go to Step 2 in Algorithm 3.4.1 and figure out that $x^2 = (1, 0)^T$, $r^2 = 1.4711$ and $\underline{f}^2 = -12.7628$ as shown in Fig. 3.4(b). Since $f(x^*) > \underline{f}^2$ and $f(x^*) > f(x^2) = -10$, according to the algorithm, we set $x^* = x^2$ and $\bar{f} = f(x^2) = -10$.

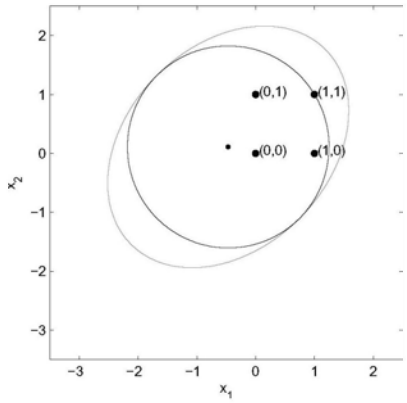
Go back to Step 2 in the algorithm and figure out that $x^3 = (0, 1)^T$, $r^3 = 1.0052$ and $\underline{f}^3 = -5.2628$ as shown in Fig. 3.4(c). Since, at this time, $\bar{f} = -10 \leq \underline{f}^3 = -5.2628$, the algorithm stops and $x^* = (1, 0)$ is the optimal solution.

Fig. 3.4 illustrates the whole progress that the global optimum of this problem can be achieved after three iterations by applying the above algorithm.

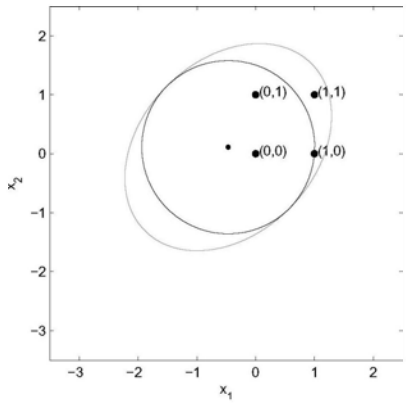
In real implementation, we do not only implement Algorithm 3.4.1 as the exact algorithm but also use this algorithm to provide us with a lower bound. For the latter case, we implement the algorithm for a few steps. When we stop the algorithm at the kth iteration, the optimal value is bounded by $[\underline{f}^k, \bar{f}]$.

3.4.3 Finding the kth Farthest 0-1 Point

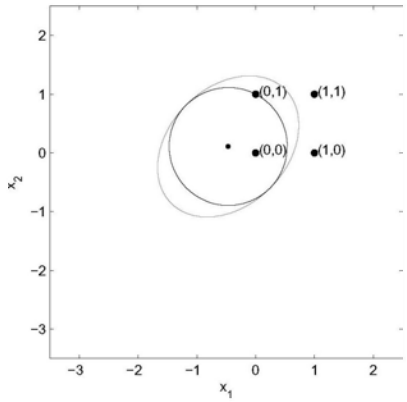
To apply Algorithm 3.4.1 in practice, we need to sort the zero-one point with respect to their distance to the center point in a descending order. Here we introduce an algorithm to find the kth farthest zero-one point. Suppose that the nearest zero-one points to x_0 is $\tilde{x}$, then the distance of any other zero-one points to x_0 can be written as:



(a) first iteration



(b) second iteration



(c) third iteration

Figure 3.4: Optimal solution achieved after three iterations

$$
\begin{aligned}
& d^2(x, x_0) \\
&= \textstyle\sum_{i=1}^{n}(x_i - x_{0i})^2 \\
&= \textstyle\sum_{i=1}^{n}(x_i - \tilde{x}_i + \tilde{x}_i - x_{0i})^2 \\
&= \textstyle\sum_{i\notin I}(x_i - \tilde{x}_i + \tilde{x}_i - x_{0i})^2 + \sum_{j\in I}(x_j - \tilde{x}_j + \tilde{x}_j - x_{0j})^2 \\
&= \textstyle\sum_{i\notin I}(\tilde{x}_i - x_{0i})^2 + \sum_{j\in I}(1 - \tilde{x}_j - x_{0j})^2 \\
&= \textstyle\sum_{i=1}^{n}(\tilde{x}_i - x_{0i})^2 - \sum_{j\in I}(\tilde{x}_j - x_{0j})^2 + \sum_{j\in I}(1 - \tilde{x}_j - x_{0j})^2 \\
&= \textstyle\sum_{i=1}^{n}(\tilde{x}_i - x_{0i})^2 + \sum_{j\in I}(1 - 2x_{0j})(1 - 2\tilde{x}_j)
\end{aligned}
\tag{11}
$$

where $I = \{i | x_i \neq \tilde{x}_i,\ i = 1, 2, \ldots, n\}$ is an index set. If $x_{0j} < 0.5$, then $\tilde{x}_j = 0$ and $(1 - 2x_{0j})(1 - 2\tilde{x}) = (1 - 2x_{0j}) > 0$, otherwise, $\tilde{x}_j = 1$ and $(1 - 2x_{0j})(1 - 2\tilde{x}) = -(1 - 2x_{0j}) > 0$. Then from (11), we get

$$
d^2(x, x_0) = \sum_{i=1}^{n}(\tilde{x}_i - x_{0i})^2 + \sum_{j\in I}|1 - 2x_{0j}|
$$

Let $a_j = |1 - 2x_{0j}|,\ (j = 1, 2, \ldots, n)$ and $d_J = \sum_{j\in J} a_j,\ (J \subseteq \{1, 2, \ldots, n\})$. By sorting d_J on all possible J in descending order, we can get, x^k, the kth farthest integer point to x^0, where $k = 1, 2, \ldots, 2^n$ and its corresponding distance. However, when n is very large, performing this sorting task is very difficult and time consuming. In addition, in most cases we just need to sort a small portion of these zero-one points. Following is a novel algorithm we invent.

Consider dividing all these 2^n zero-one points into $n + 1$ groups, $T_i (i = 0, 1, \ldots, n)$. Group T_i contains x, where $\sum_{j=1}^{n}|x_j - \tilde{x}_j| = i$. It is obvious that group T_i contains C_i^n zero-one points and both T_0 and T_n contain just one 0-1 point. They are $\tilde{x}$ and x^1, the nearest and farthest point to x_0 respectively. And for every group $T_i (i = 1, \ldots, n-1)$, we can easily find out the nearest point $t_{i,mim}$ and the farthest point $t_{i,max}$ in each group by finding out the i largest and smallest elements in set $\{a_1, a_2, \ldots, a_n\}$ respectively.

From Figure. 3.5, we can easily find that $t_{i,max}$ is the farthest point within the points in $T_0, \ldots, T_i$. From this important property, it obvious that if we want to sort the point farther than $t_{l,max},\ (l = 0, \ldots, n-1)$, we just need to sort the points in $T_{l+1}, \ldots, T_n$.

For the method to sort the elements within $T_{l+1}, \ldots, T_n$ in a descending order, we first construct a graph to represent the points in these groups and to apply the k-th farthest algorithm.

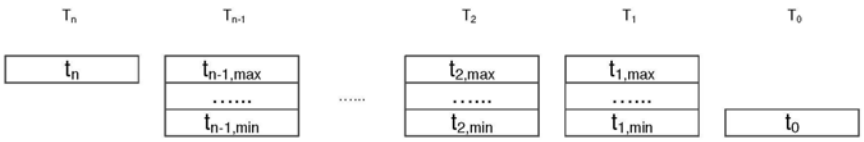


Figure 3.5: Group $T_i,\ i = 0, 1, \ldots, n$

For example, the graph in Figure 3.6 represents all the points in T_2 and T_3 when $n = 4$, i.e. $\{a_1 + a_2, a_1 + a_3, a_1 + a_4, a_2 + a_3, a_2 + a_4, a_3 + a_4, a_1 + a_2 + a_3, a_1 + a_2 + a_4, a_1 + a_3 + a_4, a_2 + a_3 + a_4\}$. And then we can apply the k-th shortest path algorithm proposed in [6] to this graph to help us perform the task of picking up the farthest points in T_2 and T_3 one by one.

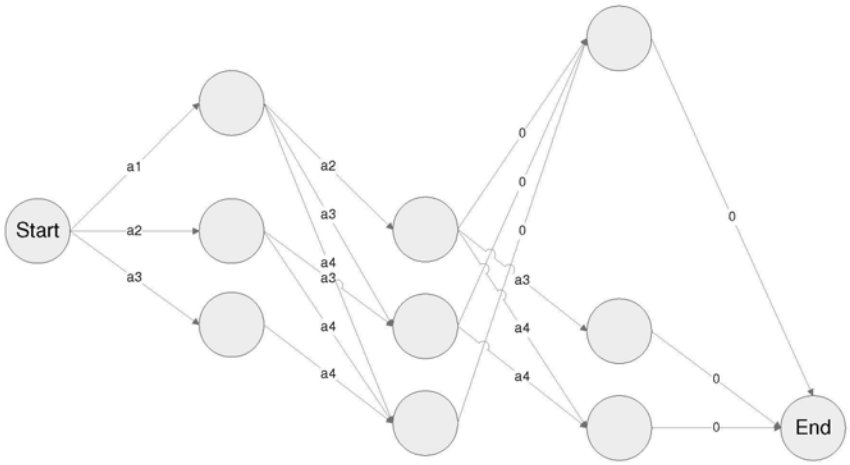


Figure 3.6: A graph represents all the points in T_2 and T_3 when $n = 4$.

3.4.4 A Condition for optimal solution to (P) within the farthest k 0-1 Points

Here we derive a condition for optimal solution to (P) within the farthest k 0-1 points, so that the method described in the previous subsection can be applied.

When the condition number of Q, C_Q, is 1, the locus of the objective contour $f(x) = C$ becomes a sphere and the optimal solution to (P) is clearly x^1, the integer point in $\{0,1\}^n$ that has the maximum 2-norm distance to $x^0 = -Q^{-1}c$.

A natural question to ask is what is the condition for optimal solution to (P) within the farthest k 0-1 points, i.e. $x^1, x^2, \ldots, x^k$.

Theorem 3.3 *Optimal solution to* (P) *must be within* $x^1, x^2, \ldots, x^k$ *if*

$$C_Q = \frac{\lambda_n}{\lambda_1} > \frac{\| x^{k+1} - x^0 \|^2}{\| x^1 - x^0 \|^2} \tag{12}$$

or if

$$\frac{2(f(x^1) - f(x^0))}{\lambda_n} > \| x^{k+1} - x^0 \|^2 . \tag{13}$$

Proof. Please refer to Figure 3.7. Let the maximum objective contour that inside the sphere $(x - x^0)^T(x - x^0) = (x^1 - x^0)^T(x^1 - x^0)$ is $f(x) = \tilde{v}$. Clearly, we have

$$(x^1 - x^0)^T(x^1 - x^0) = \frac{2\tilde{v} + c^T Q^{-1} c}{\lambda_n} \tag{14}$$

The maximum x^0-centered sphere that inside $f(x) = \tilde{v}$ is given by

$$(x - x^0)^T(x - x^0) = \frac{2\tilde{v} + c^T Q^{-1} c}{\lambda_1} \tag{15}$$

If

$$\frac{2\tilde{v} + c^T Q^{-1} c}{\lambda_1} > \| x^{k+1} - x^0 \|^2 \tag{16}$$

then all integer points, except $x^1, x^2, \ldots, x^k$, will be falling inside the ellipse $\{x \in \mathbb{R}^n | f(x) \leq \tilde{v}\}$. Combine Eq. (14) with Eq. (16), we get

$$\frac{\frac{2\tilde{v} + c^T Q^{-1} c}{\lambda_1}}{\frac{2\tilde{v} + c^T Q^{-1} c}{\lambda_n}} > \frac{\| x^{k+1} - x^0 \|^2}{\| x^1 - x^0 \|^2}.$$

We finally have

$$\frac{\lambda_n}{\lambda_1} > \frac{\| x^{k+1} - x^0 \|^2}{\| x^1 - x^0 \|^2}$$

which is Eq. (12).

On the other hand, the shortest axis of the ellipse $\{x \in \mathbb{R}^n | f(x) \geq f(x^1)\}$ is $\frac{2(f(x^1)-f(x^0))}{\lambda_n}$. If Eq. (13) is satisfied, then no other integer point will have an object valve less than $f(x^1)$. □

Fig. 3.7 illustrate the above theorem with an example. In this case, x^2 is within the sphere $(x-x^0)^T(x-x^0) = \frac{2\tilde{v}+c^TQ^{-1}c}{\lambda_n}$. That is to say $\frac{\lambda_n}{\lambda_1} > \frac{\|x^2-x^0\|^2}{\|x^1-x^0\|^2}$ is satisfied in this example. Therefore, $x^1 = (1,1)^T$ is the optimal solution.

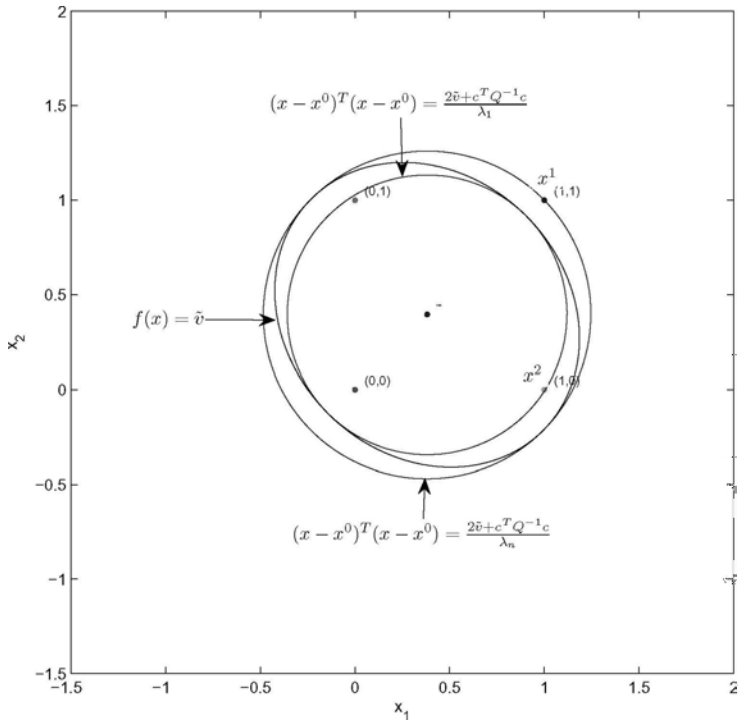


Figure 3.7: Illustrative example to Theorem 3.3

Let $l = \max\{i | \frac{\lambda_n}{\lambda_1} > \frac{\|t_{i,max},x_0\|^2}{\|\bar{x}-x^0\|^2}\}$. Then we just need to sort the points in $T_{l+1}, \ldots, T_n$ in order to find the optimal solution to (P).

3.4.5 Improved lower bound achieved on the switching points

The lower bound in the previous subsection is derived by using the information of the sphere center at x^0. This lower bound can be further improved by considering more spheres centered at the switching points located on the longest axis of a contour ellipse.

Let $\tilde{v} = f(\tilde{x})$, where $\tilde{x} \in \{0,1\}^n$ is the incumbent. Let $E(\tilde{v}) = \{x \in \mathbb{R}^n | f(x) = \tilde{v}\}$. Let $2\rho(\tilde{v})$ be the length of the longest axis of $E(\tilde{v})$, where

$$\rho(\tilde{v}) = \sqrt{\frac{2(\tilde{v} - f(x^0))}{\lambda_n}}$$

The longest axis of $E(\tilde{v})$ can be expressed as

$$I(\tilde{v}) = \{x = x^0 + \alpha P_n | \alpha \in [-\rho(\tilde{v}), \rho(\tilde{v})]\}$$

where $P_n = (P_{1n}, \ldots, P_{nn})^T$ is the last column of the orthogonal matrix P.

We consider now a family of the integer points in $\{0,1\}^n$ that are farthest to points on the longest axis $I(\tilde{v})$ parameterized by parameter α. For $i = 1, \ldots, n$, if $P_{in} \neq 0$, then the equation $\alpha P_{in} + x_i^0 = 0.5$ determines a $switch - point$ $\alpha = (0.5 - x_i^0)/P_{in}$ at which the ith component of $\bar{x}$ of $\{0,1\}^n$ that is farthest to $x^0 + \alpha P_n$ switches from $\bar{x}_i$ to $1 - \bar{x}_i$. It is easy to see that there are at most n different switch-points in the interval $(-\infty, +\infty)$. Consequently, there are at most $n+1$ different farthest points of $\{0,1\}^n$ to the points on the longest axis $I(\tilde{v})$. Suppose these m switch-points are:

$$x_{s_1}, x_{s_2}, \ldots, x_{s_n}$$

For each switch point, rank the integer points in $\{0,1\}^n$ according to a descending order with respect to their distance to x_{s_i},

$$x_{s_i}^1, x_{s_i}^2, x_{s_i}^3, \ldots$$

with corresponding distance

$$r_{s_i}^1 \geq r_{s_i}^2 \geq r_{s_i}^3 \geq \cdots$$

where

$$r_{s_i}^j = \left|\left| x_{s_i}^j - x_{s_i} \right|\right|$$

Let $\underline{f}_i^j$ be the contour level such that $E(\underline{f}_i^j) = \{x \in R^n | f(x) = \underline{f}_i^j\}$ is the minimum circumscribed contour containing the ball $S_i^j = \{x \in R^n | \, ||x - x_{s_i}||^2 = (r_{s_i}^j)^2\}$. We can utilize the following theorem to calculate $\underline{f}_i^j$.

Consider a standard ellipsoid $E = \{y \in R^n | \frac{1}{2}\sum_{i=1}^n \lambda_i y_i^2 \leq \tilde{v}\}$ with $0 < \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$. We have the following theorem for the maximum sphere centered on the longest axis of E.

Theorem 3.4 *For any given $\alpha \in \{-\sqrt{\frac{2\tilde{v}}{\lambda_1}}, \sqrt{\frac{2\tilde{v}}{\lambda_1}}\}$, the radius of the maximum inscribed sphere centered at $(\alpha, 0, \ldots, 0)^T$, $r(\alpha)$, satisfies*

$$r(\alpha) = \begin{cases} \dfrac{2\tilde{v}}{\lambda_n} - \dfrac{\alpha^2\lambda_1}{\lambda_n - \lambda_1}, \text{ if } \alpha^2 \leq \beta^2 \\ (\sqrt{\dfrac{2\tilde{v}}{\lambda_1}} - \alpha)^2, \text{ if } \alpha^2 > \beta^2 \end{cases}$$

where the constant β satisfies

$$\beta = \sqrt{\frac{2\tilde{v}(\lambda_n - \lambda_1)^2}{\lambda_1\lambda_n^2}}$$

Proof. Due to the symmetry, we only need to consider cases with $\alpha > 0$. Given α, finding out the maximum $(\alpha, 0, 0, \ldots, 0)$-centered sphere inside E is to identify the minimum distance from $(\alpha, 0, 0, \ldots, 0)$ to the ellipse by solving the following minimization problem:

$$\begin{array}{ll} \max & r(y) = (y_1 - \alpha)^2 + \sum_{i=2}^n y_i^2 \\ s.t. & \frac{1}{2}\sum_{i=1}^n \lambda_i y_i^2 = \tilde{v} \end{array} \tag{17}$$

Also due to the symmetry, we only need to find a solution with all its components nonnegative. The Lagrangian function of Problem (17) is:

$$L_\alpha(y, \mu) = (y_1 - \alpha)^2 + \sum_{i=2}^n y_i^2 + \mu(\frac{1}{2}\sum_{i=1}^n \lambda_i y_i^2 - \tilde{v}), \tag{18}$$

where $\mu \in \mathbb{R}$ is the Lagrangian multiplier. From the first order necessary condition, we get

$$\begin{cases} \frac{\partial L_\alpha(y,\mu)}{\partial y_1} = 2(y_1 - \alpha) + \mu\lambda_1 y_1 = 0 \\ \frac{\partial L_\alpha(y,\mu)}{\partial y_i} = 2y_i + \mu\lambda_i y_i = 0, \quad i = 2, \ldots, n \\ (y_1 - \alpha)^2 + \sum_{i=2}^n y_i^2 = r^2 \end{cases} \tag{19}$$

which implies

$$\begin{cases} y_1(1 + \frac{1}{2}\mu\lambda_1) = \alpha \\ y_i(1 + \frac{1}{2}\mu\lambda_i) = 0, \quad i = 2, \ldots, n \\ \frac{1}{2}\sum_{i=2}^n y_i^2 = \tilde{v} \end{cases} \tag{20}$$

We assume first that $\lambda_{n-1} < \lambda_n$. If $\alpha^2 < -\frac{2\tilde{v}(\lambda_n-\lambda_1)^2}{\lambda_1\lambda_n^2}$, then

$$\begin{aligned} \mu^* &= -\frac{2}{\lambda_n} \\ y_1^* &= \frac{\alpha}{1+\frac{1}{2}\mu\lambda_1} = \frac{\alpha\lambda_n}{\lambda_n-\lambda_1} \\ y_2^* &= y_3^* = \ldots = y_{n-1}^* = 0 \\ y_n^* &= \sqrt{\frac{2\tilde{v}}{\lambda_n} - \frac{\alpha^2\lambda_1\lambda_2}{(\lambda_n-\lambda_1)^2}} \end{aligned} \tag{21}$$

satisfy the first order necessary condition. Furthermore, we get $1+\frac{1}{2}\mu^*\lambda_i > 0, i = 1, \ldots, n-1$ and $1 + \frac{1}{2}\mu^*\lambda_n = 0$. Thus, the above solution satisfies the second-order sufficient condition, i.e., for any nonzero $y = (y_1, y_2, \ldots, y_n)^T$ satisfying $\lambda_1 y_1^* y_1 + \lambda_n y_n^* y_n = 0$, $y^T \text{diag}\{1 + \frac{1}{2}\mu^*\lambda_i\}y > 0$. Finally, the corresponding optimal radius satisfies $r^2(\alpha) = \frac{2\tilde{v}}{\lambda_n} - \frac{\alpha^2\lambda_1}{\lambda_n-\lambda_1}$.

If $\alpha^2 \geq -\frac{2\tilde{v}(\lambda_n-\lambda_1)^2}{\lambda_1\lambda_n^2}$, then

$$\begin{aligned} \mu^* &= \frac{\sqrt{2}\alpha}{\sqrt{\lambda_1\tilde{v}}} - \frac{2}{\lambda_1} \\ y_1^* &= \sqrt{\frac{2\tilde{v}}{\lambda_1}} \\ y_2^* &= y_3^* = \ldots = y_{n-1}^* = y_n^* = 0 \end{aligned} \tag{22}$$

satisfy the first order necessary condition. Note

$$\mu^* = \frac{\sqrt{2}\alpha}{\sqrt{\lambda_1\tilde{v}}} - \frac{2}{\lambda_1} \geq -\frac{2}{\lambda_n}. \tag{23}$$

Therefore, when $\alpha^2 > -\frac{2\tilde{v}(\lambda_n-\lambda_1)^2}{\lambda_1\lambda_n^2}$, the second order sufficient condition is satisfied due to $1 + \frac{1}{2}\mu^*\lambda_i > 0$, $i = 1, \ldots, n$, and when $\alpha^2 = -\frac{2\tilde{v}(\lambda_n-\lambda_1)^2}{\lambda_1\lambda_n^2}$, the second order necessary condition is satisfied. The corresponding optimal radius $r(\alpha)$ satisfies $r^2(\alpha) = (\sqrt{\frac{2\tilde{v}}{\lambda_1}} - \alpha)^2$.

The above proof can be easily extended to situations where there exist multiple largest eigenvalues.

□

Note that $f(x) = \frac{1}{2}(x-x^0)^T Q(x-x^0) - \frac{1}{2}c^T Q^{-1}c = \tilde{v}$ can be transformed into a standard form $\frac{1}{2}\sum_{i=1}^n \lambda_i y_i^2 = \tilde{v} + \frac{1}{2}c^T Q^{-1}c$ by an orthogonal transformation $x = Py + x^0$ and the distance is invariant under any orthogonal transformation. Combined with the above proof, we can immediately get the result of Theorem 3.4. Figure 3.8 illustrates the minimum contours circumscribed the switch-point-centered points along the longer axis in situation of $n = 2$.

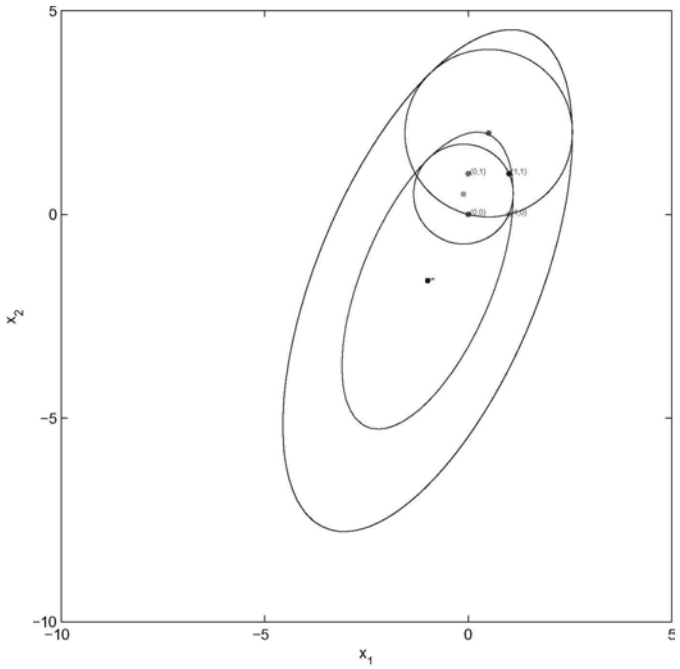


Figure 3.8: Lower bounds calculated with respect to different switching points

The following algorithm provides a lower bound to Problem (P) based on the Kth farthest point to switch-point of the ellipse.

Algorithm 3.4.2

Step 1 *Find out all switching points $s_1, s_2, \ldots, s_m$ on the longest axis of the ellipse contour.*

Step 2 *Calculate the lower bound with respect to each switching point. Select the switching point s_i with the best lower bound.*

Step 3 *Calculate $x^1_{s_i}$, $r^1_{s_i}$ and $\underline{f}^1_{s_i}$. Let $x^*_{s_i} = x^1_{s_i}$ and $\bar{f} = f(x^*_{s_i})$. Let $k = 2$.*

Step 4 *Calculate $x^k_{s_i}$, $r^k_{s_i}$ and $\underline{f}^k_{s_i}$. If $\bar{f} \leq \underline{f}^k_{s_i}$, stop and $x^*_{s_i}$ is the optimal solution.*

step 5 *Let*

$$x^*_{si} = \begin{cases} x^*_{s_i}, \text{ if } f(x^*_{s_i}) \leq f(x^k_{s_i}) \\ x^k_{s_i}, \text{ if } f(x^k_{s_i}) < f(x^*_{s_i}) \end{cases}$$
$$\bar{f} = f(x^*_{s_i})$$

Step 6 If $k = 2^n$, stop and $x^*_{s_i}$ is the optimal solution. Let $k = k + 1$ and go back to Step 4.

The above algorithm either provides a better lower bound or the optimal solution to (P). The proof is similar to the proof of Theorem 3.4.1.

3.5 Variable Fixation

In this section, we discuss how to fix some variables at their optimal values in (P) or in its subproblems using information from the objective contour. Variable fixation based on some optimality condition is one of the most efficient strategies in exact solution methods for solving (P).

3.5.1 One classical sufficient conditions

The following is a well-known sufficient condition for fixing variable by utilizing the bounds of the gradient of $f(x)$.

Theorem 3.5 *([48]) Let x^* denote the optimal solution of . Let*

$$l_i = c_i + \frac{1}{2}q_{ii} + \sum_{j \neq i} \min(0, q_{ij}), \tag{24}$$

$$u_i = c_i + \frac{1}{2}q_{ii} + \sum_{j \neq i} \max(0, q_{ij}), \tag{25}$$

(i) If $l_i \geq 0$, then $x^*_i = 1$;
(ii) If $u_i \leq 0$, then $x^*_i = 0$;

The above result can be used to fix variables in a branch-and-bound method. Let I_{free} denote the indexes of the free variables. I^0_{free} the index of variables fixed at 0 and I^1_{free} the index of variables fixed at 1 at a node of the branch-and-bound tree.

The following procedure uses Theorem 3.5 to fix variables in the branch-and-bound method.

Procedure 2 *Input: I^0_{fix}, I^1_{fix} and I_{free}. Output: updated I^0_{fix}, I^1_{fix} and I_{free}.*

Step 1. For each $i \in I_{free}$. compute the gradient bounds l_i and u_i by Eqs. 24 and 25, respectively.

Step 2. If $l_i > 0$, *set* $I^0_{fix} := I^0_{fix} \cup \{i\}$, $I^0_{free} := I^0_{free} \setminus \{i\}$. *If* $u_i < 0$, *set* $I^1_{fix} := I^1_{fix} \cup \{i\}$, $I^1_{free} := I^1_{free} \setminus \{i\}$.

Step 3. Update the gradient bounds and repeat Steps 1-2 until no variable can be fixed. Return.

3.5.2 A new sufficient optimality condition

Now we consider a new approach to fix variables at 0 or 1. Let $\bar{x}(\mu)$ be the farthest of $\{0,1\}^n$ to $x^0(\mu)$. Let $\bar{t}(\mu) = f_\mu(\bar{x}(\mu))$. By Theorem 3.4, the maximum sphere that inside the ellipse contour $E_\mu(\bar{t}(\mu)) = \{x \in \mathbb{R}^n | f_\mu(x) = \bar{t}(\mu)\}$ is

$$\bar{S}(\mu) = \{x \in \mathbb{R}^n | \parallel x - x^0(\mu) \parallel = \bar{r}(\mu)\}, \tag{26}$$

where

$$\bar{r}(\mu) = \frac{2(\bar{t}(\mu) + \frac{1}{2}c^T Q^{-1} c)}{\lambda_n}. \tag{27}$$

Denote $H^i = \{x \in \mathbb{R}^n | x_i = 1 - \bar{x}_i(\mu)\}$. Let $y^i(\mu)$ be the farthest point from $H^i \cap \{0,1\}^n$ to $x^0(\mu)$. Since $\bar{x}(\mu)$ is the farthest point from $\{0,1\}^n$ to $x^0(\mu)$, for $i = 1, 2, \ldots, n$, $y^i(\mu)$ can be given as follows:

$$y^i(\mu)_j = \bar{x}(\mu)_j,\ j \neq i, y^i(\mu)_i = 1 - \bar{x}(\mu)_i. \tag{28}$$

Define

$$\omega_i(\mu) = \parallel y^i(\mu) - x^0(\mu) \parallel . \tag{29}$$

If $\omega_i(\mu) < \bar{r}(\mu)$, then all points of $\{0,1\}^n$ on the hyperplane H^i are inside of $\bar{E}_\mu(\bar{t}(\mu))$ and thus $\bar{S}(\mu)$. Therefore, for any point $H^i \cap \{0,1\}^n$, it holds $f(x) = f_\mu(x) > f_\mu(\bar{x}_\mu) = f(\bar{x})$, which in turn implies that the ith variable in the optimal solution of (P) must be equal to $\bar{x}_i(\mu)$. In summary, we have the following result.

Theorem 3.6 *Let* x^* *be the optimal solution of* (P). *If for some* $i \in \{1, 2, \ldots\}$, $\omega_i(\mu) < \bar{r}(\mu)$, *then* $x^*_i = \bar{x}_i(\mu)$.

Figure 3.9 demonstrates a two-dimensional example where $\bar{x}(\mu) = (1,1)^T$, $y^1(\mu) = (0,1)$, $y^2(\mu) = (1,0)$. We see that both $\| y^1(\mu) - x^0(\mu) \| < \bar{r}(\mu)$ and $\| y^2(\mu) - x^0(\mu) \| < \bar{r}(\mu)$. Therefore, both x_1^* and x_2^* can be fixed at 1 by the inscribed sphere.

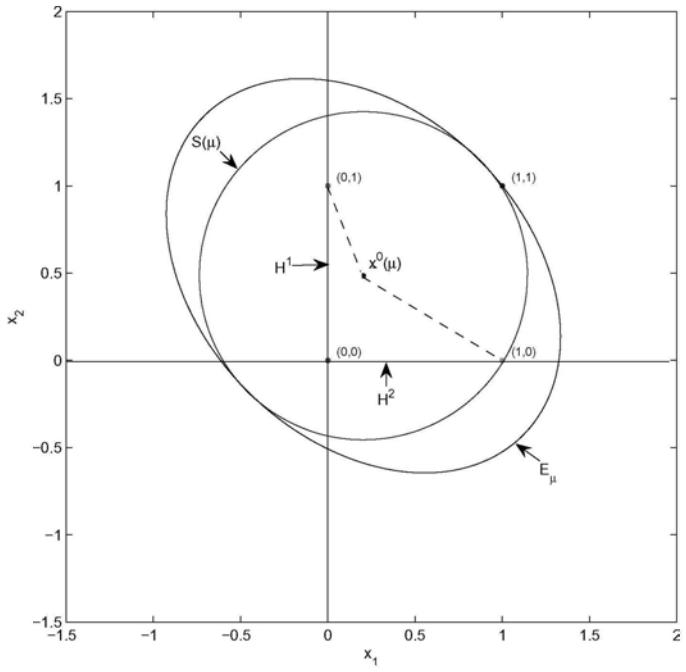


Figure 3.9: Illustration of variable fixation by inscribed sphere

A procedure that employs Theorem 3.6 to fix variables can be described as follows:

Procedure 3 *Input: I_{fix}^0, I_{fix}^1 and I_{free}. Output: updated I_{fix}^0, I_{fix}^1 and I_{free}.*

Step 1. Compute $\bar{r}_\mu$ and $\bar{x}_\mu$ of $\{0,1\}^n$.

Step 2. For each $i \in I_{free}$, compute $y^i(\mu)$. If $\omega_i(\mu) < \bar{r}(\mu)$, then set $I_{free} := I_{free} \setminus \{i\}$, and set $I_{fix}^0 := I_{fix}^0 \cup \{i\}$ if $\bar{x}(\mu) = 0$, or set $I_{fix}^1 := I_{fix}^1 \cup \{i\}$ if $\bar{x}(\mu) = 1$.

3.6 The Algorithm

Integrating the upper bounding method, the lower bounding method variable fixation strategies, techniques of seeking feasible solutions and optimality conditions discussed in the previous sections, we formally present an exact solution algorithm of a branch-and-bound type.

An initial incumbent solution is first computed and is set to be the current best upper bound. Procedure 5.1 is used to fix some variables at their optimal values. The algorithm matains a list L of nodes at each of I^0_{fix} and I^1_{fix}, the indexes of fixed variables, I_{free} the index of free variables, and l and u, the lower and upper bounds of the gradient of the free variables, are stored. At each iteration, a node is selected from the list for which the upper bound $u(\mu)$ and the lower bound $l(\mu)$ are computed for a suitable μ. If $u(\mu)$ is less than the current best upper bound, then the current best upper bound is replaced by this value. A node is fathomed if the lower bound is greater than or equal to the current best upper bound. Otherwise, conditions for variable fixation are checked. Again, a node is fathomed if all free variables are fixed. If there are still free variables for this node, then one of the free variables, say x_i, is selected to prune into two new nodes with $x_i = 0$ and $x_i = 1$, respectively. The gradients bounds are updated and the two new nodes are added to the list. The algorithm terminates with the incumbent being the optimal solution if there is no node left in the list.

we describe the main algorithm as follows.

Algorithm 3.6.1 (Branch-and-bound method for (P))

Step 0. (Initialization).

(i) Choose $x^0 = -Q^{-1}c$ and the farthest point $\tilde{x}$ of $\{0,1\}^n$ to x^0. Let $f_{opt} = f(\tilde{x})$ and $x_{opt} = \tilde{x}$.

(ii) (Variable fixation). Set $I^0_{fix} = I^1_{fix} = \phi$ and $I_{free} = \{1, 2, \ldots, n\}$. Apply Procedure 2 and 3 to fix as many variables as possible at their optimal values.

(iii) If $I_{free} = \phi$, set $k = 0$. Otherwise, set $L = \{(I^0_{fix}, I^1_{fix}, I_{free}, l, u)\}$ and $k = 1$.

Step 1. If $k = 0$, stop and x_{opt} is the optimal solution. Otherwise select a node from L and construct the subproblem at the selected node by setting $\tilde{Q} = (Q)_{I_{free} \times I_{free}}$ and $\tilde{c}_i = c_i + \sum_{j \in I^1_{fix}} q_{ij}$ for $i \in I_{free}$. Let

$$v_{fix} = \frac{1}{2} \sum_{i \in I^1_{fix}} \sum_{j \in I^1_{fix}} q_{ij} + \sum_{i \in I^1_{fix}} c_i$$

Set $k := k - 1$

Step 2. (Upper bounding, lower bounding and fathoming) Choose a suitable $\mu \geq 0$.

(i) Compute the upper bound $u(\mu)$ defined in Eq.(10) for the subproblem at this node. If $u(\mu) + v_{fix} < f_{opt}$, update $f_{opt} = u(\mu) + v_{fix}$.

(ii) Find out all the switch-points for the subproblem and calculate their associate lower bound by Theorem 3.4. If the dimension of the subproblem, n', is great than $n \times 20\%$, set $k = \frac{1}{2}n'$ and choose the swithch-point that has the best lower bound then improve the lower bound $l(\mu)$ according to Algorithm 3.4.2. If $l(\mu) \geq f_{opt} - v_{fix}$, fathom the current node and goto Step 1.

Step 3. (Variable fixation and fathoming) Apply Procedure 2 and 3 to the subproblem at the current node. If all the free variables are fixed. goto Step 1.

Step 4. (Branching). Choose an $i \in I_{free}$ and set $I_{free} := I_{free} \backslash \{i\}$. Set $I^0_{fix} := I^0_{fix} \cup \{i\}$ and $I^1_{fix} := I^1_{fix} \cup \{i\}$, respectively, to generate two new nodes. Add these two nodes to L and set $k := k + 2$. Goto Step 1.

3.7 Numerical Results

In this section, we present numerical results for proposed algorithm. The main purpose of our computational experiments is to test the efficiency of the new upper bounding and lower bounding methods discussed in the previous sections.

The algorithm was coded by Visual C++ 6.0 and run on a Pentium IV PC. Two classes of test problems in the literature are considdered in our computational experiments. The first class of test problems is Carter-type of problems with $q_{ij} \in [-50, 50]$ for $i \neq j$, $q_{ii} = 0$, and $c_i \in [-d_m, d_m]$, where $d_m = 25(n-1)p$ and $p \in (0, 1)$ is the parameter to control the diagonal dominance of Q. The second class of test problems is Williams-type of problems with $q_{ij} \in [-50, 50]$ for $i \neq j, q_{ii} = 0$, and $c_i \in [-25, 25]$. In problems of a Williams-type, q_{ij} takes zero with a probability $1 - d$ where $d \in (0, 1)$ is the adjustable parameter to control the density of Q.

Numerical results are reported in Tables 3.1, 3.2 and 3.3, where N_{node} is the average number of nodes generated in the method for 10 test problems, T_{cpu} is the average CPU time used for 10 test problems. Tables 3.1, 3.2 and 3.3 indicate that our algorithm has considerably good performance in terms of average number of nodes generated and average CPU time. This is largely due to the improvement of the lower bound and the efficiency of fixing variables by the proposed new optimality condition.

Table 3.1: Numerical Results for Carter-type test Problems

n	d	N_{node}	T_{cpu}
40	0.8	787.6	1.262
40	0.9	1706.2	1.529
50	0.8	6540.3	9.181
50	0.9	1814.1	3.164
80	1.0	2091.2	6.302

Table 3.2: Numerical Results for Williams-type test Problems ($n = 40$)

n	40	40	40	40	40
d	0.2	0.3	0.5	0.8	0.9
$N^1_{node}\|T^1_{cpu}$	106|0.282	7|0.266	19|0.296	2151|14.734	2961|25.844
$N^2_{node}\|T^2_{cpu}$	41|0.265	78|0.282	121|0.297	1681|12.438	1531|9.063
$N^3_{node}\|T^3_{cpu}$	66|0.281	13|0.250	37|0.250	2216|11.984	2835|31.953
$N^4_{node}\|T^4_{cpu}$	35|0.250	103|0.281	131|0.313	700|3.172	5184|42.828
$N^5_{node}\|T^5_{cpu}$	139|0.297	37|0.250	66|0.266	1768|19.547	113|3.875
$N^6_{node}\|T^6_{cpu}$	50|0.265	28|0.250	154|0.375	1169|7.922	3249|32.828
$N^7_{node}\|T^7_{cpu}$	36|0.266	103|0.297	120|0.313	755|7.375	3121|23.313
$N^8_{node}\|T^8_{cpu}$	46|0.250	21|0.250	67|0.266	436|2.047	1026|2.984
$N^9_{node}\|T^9_{cpu}$	154|0.344	55|0.266	120|0.313	496|3.953	1623|4.295
$N^{10}_{node}\|T^{10}_{cpu}$	166|0.359	16|0.266	211|0.484	1181|11.437	1409|10.406
$N^{ave}_{node}\|T^{ave}_{cpu}$	83.9|0.286	46.1|0.266	104.6|0.317	1255.3|9.461	2305.2|18.739

□ **End of chapter.**

Table 3.3: Numerical Results for Williams-type test Problems ($n \geq 60$)

n	60	60	60	70	70	80
d	0.1	0.2	0.7	0.1	0.2	0.1
$N^1_{node}\|T^1_{cpu}$	933|4.750	379|2.204	211|1.250	619|3.375	465|3.437	766|10.984
$N^2_{node}\|T^2_{cpu}$	253|1.518	227|1.390	561|6.875	705|5.250	426|3.391	1722|26.015
$N^3_{node}\|T^3_{cpu}$	394|1.718	666|6.890	301|1.625	859|4.813	430|3.640	552|6.109
$N^4_{node}\|T^4_{cpu}$	364|1.641	133|1.031	79|1.000	1098|7.093	166|1.812	852|14.715
$N^5_{node}\|T^5_{cpu}$	598|2.531	346|1.985	379|2.219	1669|29.469	248|2.047	741|9.172
$N^6_{node}\|T^6_{cpu}$	575|1.703	183|1.110	667|7.047	727|8.828	526|4.750	869|10.094
$N^7_{node}\|T^7_{cpu}$	260|0.453	325|1.609	465|3.156	491|3.953	466|3.702	1701|41.187
$N^8_{node}\|T^8_{cpu}$	362|1.266	226|1.187	379|2.203	827|5.547	377|2.437	1423|18.453
$N^9_{node}\|T^9_{cpu}$	519|3.031	231|1.297	79|1.016	761|5.922	426|3.375	998|6.594
$N^{10}_{node}\|T^{10}_{cpu}$	814|3.312	491|3.266	631|6.188	442|2.985	430|3.109	869|12.250
$N^{ave}_{node}\|T^{ave}_{cpu}$	507.2|2.192	320.7|2.197	375.2|3.258	819.8|7.724	396.0|3.170	1049.3|15.557

Chapter 4

Polynomial Algorithms to Binary Quadratic Programming Problems with Q being a Tri-Diagonal or Five-Diagonal Matrix

We focus in this chapter polynomial algorithms to binary quadratic programming problems with Q being a tri-diagonal or five-diagonal matrix by taking advantage of the basic algorithm proposed in [1, 33, 44]. We review the basic algorithm firstly and then modify this algorithm to solve binary quadratic programming problems with Q being a tri-diagonal or five-diagonal matrix. Furthermore, by combining the basic algorithm and dynamic programming method, we propose a new method to solve linear constraint binary quadratic programming problems with Q being a tri-diagonal matrix.

4.1 Basic Algorithm to Binary Quadratic Programming in General Form

We first consider problem $(0\text{-}1QP)$ in its general form in this section. Denote by $\Delta_i(x)$ the ith derivative of $f(x) = x^T Qx + c^T x$ at x,

$$\begin{aligned}\Delta_i(x) &= \frac{\partial f}{\partial x_i} \\ &= f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_n) - f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n).\end{aligned}$$

Denote by $\Theta_i(x)$ the ith residual

$$\begin{aligned}\Theta_i(x) &= f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_n) \\ &= f(x) - x_i \Delta_i(x).\end{aligned}$$

Both $\Delta_i(x)$ and $\Theta_i(x)$ are, in general, *linear* functions of $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. Moreover, f can be expressed as

$$f(x) = x_i \Delta_i(x) + \Theta_i(x). \tag{1}$$

It is clear that a point $x \in \{0,1\}^n$ is a solution to (0-1QP) only if for all $i = 1, \ldots, n$,

$$x_i = \begin{cases} 1, & \text{if } \Delta_i(x) < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

The basic algorithm [41][33] is developed based on the above necessary optimality condition. We first express $f(x)$ in (0-1QP) as

$$f(x) = x_n \Delta_n(x_1, \ldots, x_{n-1}) + \Theta_n(x_1, \ldots, x_{n-1}). \tag{3}$$

From the optimal condition (2), the global minimizer of f satisfies

$$x_n = \begin{cases} 1, & \text{if } \Delta_n(x_1, \ldots, x_{n-1}) < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Therefore, if we can express x_n defined in (4) as a polynomial of $x_1, \ldots, x_{n-1}$, $\phi_n(x_1, \ldots, x_{n-1})$, then we can eliminate x_n from the expression of $f(x)$ in (3),

$$f_{n-1}(x_1, \ldots, x_{n-1}) = \phi_n(x_1, \ldots, x_{n-1}) \Delta_n(x_1, \ldots, x_{n-1}) + \Theta_n(x_1, \ldots, x_{n-1}).$$

Note that, in general cases, $f_{n-1}(x_1, \ldots, x_{n-1})$ may not be a quadratic function, as $\phi_n(x_1, \ldots, x_{n-1})$, in general, is not a linear function. Performing the same elimination process for f_{n-1}, we will get a function f_{n-2} of $x_1, \ldots, x_{n-2}$ and this process continues recursively until we obtain $f_1(x_1)$. Let x^* denote the optimal solution of (0-1QP). Notice that $x_1^* = 1$ if $f_1(1) < f_1(0)$ and $x_1^* = 0$ otherwise. Then $x_2^*, \ldots, x_n^*$ can be obtained by using $x_{i+1}^* = \phi_{i+1}(x_1^*, \ldots, x_i^*)$ recursively for $i = 1, \ldots, n-1$.

The basic algorithm [41][33] can then be described as follows.

Algorithm 4.1.1 (Basic Algorithm for $(0\text{-}1QP)$**)**

Step 0. Set $f_n(x) = f(x)$ and $k = n$.

Step 1. Calculate

$$\Delta_k(x_1, \ldots, x_{k-1}) = \frac{\partial f_k}{\partial x_k},$$
$$\Theta_k(x_1, \ldots, x_{k-1}) = f_k(x_1, \ldots, x_{k-1}, 0).$$

Determine the polynomial expression of ϕ_k defined by

$$\phi_k(x_1, \ldots, x_{k-1}) = \begin{cases} 1, & \text{if } \Delta_k(x_1, \ldots, x_{k-1}) < 0, \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Step 2. Compute

$$f_{k-1}(x_1, \ldots, x_{k-1}) = \phi_k(x_1, \ldots, x_{k-1})\Delta_k(x_1, \ldots, x_{k-1}) + \Theta_k(x_1, \ldots, x_{k-1}).$$

Step 3. If $k > 1$, then set $k := k - 1$ and go to Step 1. Otherwise, set $x_1^* = 1$ if $f_1(1) < f_1(0)$ and $x_1^* = 0$ if $f_1(1) \geq f_1(0)$. Calculate x_k^* by $x_k^* = \phi_k(x_1^*, \ldots, x_{k-1}^*)$ for $k = 2, \ldots, n$.

It is proved in [41] that the basic algorithm produces an optimal solution x^* to $(0\text{-}1QP)$. The following small-size example illustrates the algorithm.

Example 4.1.1

$$\max_{x \in \{0,1\}^3} f(x) = 4x_1x_2 - x_1x_3 + 2x_2x_3.$$

By the algorithm, we have $\Delta_3(x_1, x_2) = -x_1 + 2x_2$ and thus

$$\phi_3(x_1, x_2) = \left\{ \begin{array}{ll} 1, & \text{if } \Delta_3(x_1, x_2) < 0 \\ 0, & \text{otherwise} \end{array} \right\} = x_1(1 - x_2).$$

Hence we get

$$\begin{aligned} f_2(x_1, x_2) &= \phi_3(x_1, x_2)\Delta_3(x_1, x_2) + \Theta_3(x_1, x_2) \\ &= x_1(1 - x_2)(-x_1 + 2x_2) + 4x_1x_2 \\ &= 5x_1x_2 - x_1. \end{aligned}$$

Since $\Delta_2(x_1) = 5x_1$, we get

$$\phi_2(x_1) = \left\{ \begin{array}{ll} 1, & \text{if } g_2(x_1) < 0, \\ 0, & \text{otherwise} \end{array} \right\} = 0.$$

Thus,

$$f_1(x_1) = \phi_2(x_1)\Delta_2(x_1) + \Theta_2(x_1) = -x_1.$$

Therefore, $x_1^* = 1$, $x_2^* = \phi_2(x_1^*) = 0$ and $x_3^* = \phi_3(x_1^*, x_2^*) = 1$. The optimal solution to the example is $x^* = (1, 0, 1)^T$ with $f(x^*) = -1$.

The key task in performing the basic algorithm is how to identify the polynomial expression of ϕ_k defied in (5). Techniques to obtain the polynomial expression ϕ_k are discussed in [33][42]. In principle, ϕ_k can be always constructed systematically. Let's consider the following instance, $\Delta_4(x_1, x_2, x_3) = 4x_1 - x_2 - 5x_3$. The first step is to find the mapping from all possible combinations of x_1, x_2 and x_3 to the value of Δ_4 which is given in the following table.

Table 4.1: Illustrative example of mapping Δ_k

x_1	x_2	x_3	$\Delta_4(x_1, x_2, x_3)$
0	0	0	0
1	0	0	4
0	1	0	-1
0	0	1	-5
1	1	0	3
1	0	1	-1
0	1	1	-6
1	1	1	-2

Using Boolean algebra and noticing that all possible combinations of x_1, x_2 and x_3 are mutually exclusive, we can get

$$\begin{aligned}\phi_4(x_1, x_2, x_3) &= (1 - x_1)x_2(1 - x_3) + (1 - x_1)(1 - x_2)x_3 + x_1(1 - x_2)x_3 \\ &\quad +(1 - x_1)x_2x_3 + x_1x_2x_3 \\ &= x_2 - x_3 - x_1x_2 - x_2x_3 + x_1x_2x_3.\end{aligned}$$

Note that if Δ_k involves s variables, then we need to examine 2^s combinations. In the worst case, if Δ_n involves $n - 1$ variables, calculating ϕ_n is more than enumerating 2^{n-1} possible solutions. The basic algorithm could become very powerful for (0-1QP) when interactions among variables are weak, for example, when matrix Q in (0-1QP) is tridiagonal.

4.2 Problem $(0\text{-}1QP)$ with Q being a tridiagonal Matrix

In this section, we consider the following special case of problem $(0\text{-}1QP)$:

$$(UP)\ \ f(x) = \min_{x\in\{0,1\}^n} \quad x'Qx + c'x,$$

where Q is a tridiagonal symmetric matrix with zero diagonal elements,

$$Q = \begin{pmatrix} 0 & q_{12} & 0 & \dots & 0 & 0 \\ q_{12} & 0 & q_{23} & \dots & 0 & 0 \\ 0 & q_{23} & 0 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & q_{n-1,n} \\ 0 & 0 & 0 & \dots & q_{n-1,n} & 0 \end{pmatrix}.$$

In this special case, it can be verified that both functions Δ_k and ϕ_k are linear functions of a single variable x_{k-1}. Thus, f_k remains a quadratic form all the way through the iteration. The basic algorithm becomes polynomial in such a special case.

Algorithm 4.2.1 (Exact Algorithm for $(0\text{-}1QP)$ with Q being Tridiagonal)

Step 0. Set $f_n(x) = f(x)$ and $k = n$.

Step 1. Calculate

$$\Delta_k(x_{k-1}) = \frac{\partial f_k}{\partial x_k} = 2q_{k-1,k}x_{k-1} + c_k,$$
$$\Theta_k(x_1, \dots, x_{k-1}) = f_k(x_1, \dots, x_{k-1}, 0).$$

Determine the polynomial expression of ϕ_k defined by

$$\phi_k(x_{k-1}) = \begin{cases} 1 & \text{if } 2q_{k-1,k} + c_k < 0 \text{ and } c_k < 0, \\ 0 & \text{if } 2q_{k-1,k} + c_k \geq 0 \text{ and } c_k \geq 0, \\ x_{k-1} & \text{if } 2q_{k-1,k} + c_k < 0 \text{ and } c_k \geq 0, \\ 1 - x_{k-1} & \text{if } 2q_{k-1,k} + c_k \geq 0 \text{ and } c_k < 0. \end{cases} \tag{6}$$

Step 2. Compute

$$f_{k-1}(x_1, \dots, x_{k-1}) = \phi_k(x_{k-1})\Delta_k(x_{k-1}) + \Theta_k(x_1, \dots, x_{k-1}),$$

and simplify the expression using $x_{k-1}^2 = x_{k-1}$.

Step 3. If $k > 1$, then set $k := k - 1$ and go to Step 1. Otherwise, set $x_1^* = 1$ if $f_1(1) < f_1(0)$ and $x_1^* = 0$ if $f_1(1) \geq f_1(0)$. Calculate x_k^* by $x_k^* = \phi_k(x_{k-1}^*)$ for $k = 2, \ldots, n$.

Note that at every iteration, $\phi_k(x_{k-1})\Delta_k(x_{k-1})$ will always be a linear function of x_{k-1}. Therefore, $f_{k-1}(x_1, \ldots, x_{k-1})$ is also a quadratic function with Q being a tri-diagonal matrix. Then we can further calculate $\phi_{k-1}(x_{k-2})$ and $\Delta_{k-1}(x_{k-2})$ very easily. As a result, the algorithm we proposed is very efficient.

4.3 Problem $(0\text{-}1QP)$ with Q being a five-diagonal Matrix

In this section, we consider the following special binary quadratic programming problems:

$$(UP) \quad f(x) = \min_{x \in \{0,1\}^n} \qquad x'Qx + c'x,$$

where Q is a five-diagonal matrix with zero diagonal elements, and $c \in \mathbb{R}^n$.

When Q is an n-dimensionsl five-diagonal matrix, Q takes the following form:

$$Q = \begin{pmatrix} 0 & q_{12} & q_{13} & 0 & \ldots & 0 & 0 & 0 \\ q_{12} & 0 & q_{23} & q_{24} & \ldots & 0 & 0 & 0 \\ q_{13} & q_{23} & 0 & q_{34} & \ldots & 0 & 0 & 0 \\ 0 & q_{24} & q_{34} & 0 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & 0 & q_{n-2,n-1} & q_{n-2,n} \\ 0 & 0 & 0 & 0 & \ldots & q_{n-2,n-1} & 0 & q_{n-1,n} \\ 0 & 0 & 0 & 0 & \ldots & q_{n-2,n} & q_{n-1,n} & 0 \end{pmatrix}$$

Based on the basic algorithm and the algorithm for solving (UP) with Q is a tri-diagonal matrix stated in Section 4.2, an algorithm can be developed for solving (UP) where Q is a five-diagonal matrix.

By applying the method described in Section 4.2, we let $s_1 = c_k$, $s_2 = q_{k-1,k} + c_k$, $s_3 = q_{k-2,k} + c_k$ and $s_4 = q_{k-2,k} + q_{k-1,k} + c_k$. And by assigning different value to s_1, s_2,

s_3 and s_4, we can easily get the definitions of ϕ_k under different situation shown in Table 4.2. Then, we figure out the algorithm for UP with Q is a five-diagonal matrix as follows:

Table 4.2: Definitions for $\phi_k(x_{k-2}, x_{k-1})$ according to the value of s_1, s_2, s_3 and s_4

s_1	s_2	s_3	s_4	$\phi_k(x_{k-2}, x_{k-1})$
< 0	< 0	< 0	< 0	1
< 0	< 0	< 0	≥ 0	$1 - x_{k-2}x_{k-1}$
< 0	< 0	≥ 0	< 0	$1 - x_{k-2} + x_{k-2}x_{k-1}$
< 0	< 0	≥ 0	≥ 0	$1 - x_{k-2}$
< 0	≥ 0	< 0	< 0	$1 - x_{k-1} + x_{k-2}x_{k-1}$
< 0	≥ 0	< 0	≥ 0	$1 - x_{k-1}$
< 0	≥ 0	≥ 0	< 0	$1 - x_{k-2} - x_{k-1} + 2x_{k-2}x_{k-1}$
< 0	≥ 0	≥ 0	≥ 0	$1 - x_{k-2} - x_{k-1} + x_{k-2}x_{k-1}$
≥ 0	< 0	< 0	< 0	$x_{k-2} + x_{k-1} - x_{k-2}x_{k-1}$
≥ 0	< 0	< 0	≥ 0	$x_{k-2} + x_{k-1} - 2x_{k-2}x_{k-1}$
≥ 0	< 0	≥ 0	< 0	x_{k-1}
≥ 0	< 0	≥ 0	≥ 0	$x_{k-1} - x_{k-2}x_{k-1}$
≥ 0	≥ 0	< 0	< 0	x_{k-2}
≥ 0	≥ 0	< 0	≥ 0	$x_{k-2} - x_{k-2}x_{k-1}$
≥ 0	≥ 0	≥ 0	< 0	$x_{k-2}x_{k-1}$
≥ 0	≥ 0	≥ 0	≥ 0	0

Algorithm 4.3.1 (Exact Algorithm for $(0\text{-}1QP)$ with Q being Five-Diagonal)

Step 0. Set $f_n(x) = f(x)$ and $k = n$.

Step 1. Calculate

$$\Delta_k(x_{k-2}, x_{k-1}) = \frac{\partial f_k}{\partial x_k} = q_{k-2,k}x_{k-2} + q_{k-1,k}x_{k-1} + c_k,$$
$$\Theta_k(x_1, \ldots, x_{k-1}) = f_k(x_1, \ldots, x_{k-1}, 0).$$

Let $s_1 = c_k$, $s_2 = q_{k-1,k} + c_k$, $s_3 = q_{k-2,k} + c_k$ and $s_4 = q_{k-2,k} + q_{k-1,k} + c_k$. Determine the polynomial expression of ϕ_k defined by Table 4.2

Step 2. Compute

$$f_{k-1}(x_1, \ldots, x_{k-1}) = \phi_k(x_{k-2}, x_{k-1})\Delta_k(x_{k-2}, x_{k-1}) + \Theta_k(x_1, \ldots, x_{k-1}),$$

and simplify the expression using $x_{k-2}^2 = x_{k-2},\ x_{k-1}^2 = x_{k-1}$.

Step 3. If $k > 3$, then set $k := k-1$ and go to Step 1. Otherwise, determine the value of x_1^* and x_2^* that minimize $f_2(x_1, x_2)$. Calculate x_k^* by $x_k^* = \phi_k(x_{k-2}^*, x_{k-1}^*)$ for $k = 3, \ldots, n$.

Note that at every iteration, $\phi_k(x_{k-2}, x_{k-1})\Delta_k(x_{k-2}, x_{k-1})$ will always be a linear combination of $x_{k-2}x_{k-1}$, x_{k-2} and x_{k-1}. Therefore, $f_{k-1}(x_1, \ldots, x_{k-1})$ is also a quadratic function with Q being a five-diagonal matrix. Then we can further calculate $\phi_{k-1}(x_{k-3}, x_{k-2})$ and $\Delta_{k-1}(x_{k-3}, x_{k-2})$ very easily. As a result, the algorithm we proposed is very efficient.

4.4 Algorithms to Linearly Constrained BQP with Q being a Tri-diagonal Matrix Based on Dynamic Programming Method

Here we combine the dynamic programming method with the previous exact solution algorithm to solve the following linearly and singly constrained quadratic 0-1 problem:

$$\begin{aligned} (CP)\ \ f(x) = \min_{x\in\{0,1\}^n} \quad & x'Qx + c'x, \\ \text{Subject to:} \quad & a'x \le b \end{aligned}$$

where Q is a tri-diagonal symmetric matrix with zero diagonal elements, $c \in \mathbb{R}^n,\ a \in \mathbb{Z}_+^n$, and $b \in \mathbb{Z}_+$.

To apply dynamic programming, we first introduce a stage variable k, $0 < k \le n$, and a state at stage k, $s_k \in \mathbb{R}$, satisfying the following recursive equation:

$$s_{k+1} = s_k + a_k x_k,\ k = 1, \ldots, n-1,$$

with an initial condition $s_1 = 0$.

Since the constraints are integer-valued, we only need to consider integer points in the state space. Furthermore, the feasible region of the state vector at stage k with $2 \le k \le n$ can be confined as follows:

$$\underline{s}_k \le s_k \le \bar{s}_k$$

where

$$\underline{s}_k = \sum_{t=1}^{k-1} \min_{x_t\in\{0,1\}} a_t x_t = 0$$

and

$$\bar{s}_k = \min\{\sum_{t=1}^{k-1} \max_{x_t \in \{0,1\}} a_t x_t,\ b - \sum_{t=k}^{n} \min_{x_t \in \{0,1\}} a_t x_t\} = \min\{\sum_{t=1}^{k-1} a_t,\ b\}.$$

Then for each stage k, we can write out all its possible states of s_k. Then we can define $\{s_k\} = \{s | \underline{s}_k \leq s_k \leq \bar{s}_k, s_k \in \mathbb{Z}\}$. For every state, we can define its cost-to-go function, $\hat{t}_k(s_k)$, which is a function defined over $\{x_1, \ldots, x_{k-1}\}$ with the following procedure.

Procedure 4 (Define the cost-to-go function for state s_k)

- Case 1. If $k = n$

  We need to consider the following two cases:

  - Case 1a. If $s_n + a_n > b$. It means that, at this state, x_n must be set as zero in order to satisfy the linear constraint, so we set $x_n^* = 0$ and define $\hat{t}_n(s_n) = f(x_1, \ldots, x_{n-1}, 0)$.
  - Case 1b. If $s_n + a_n \leq b$. It means, at this state, x_n can be either 0 or 1. Therefore we apply the exact solution algorithm introduced in previous section to $f_n(x)$ and finally get $x_n^* = \phi_n(x_{n-1})$ and $\hat{t}_n(s_n) = f_{n-1}(x_1, \ldots, x_{n-1})$

  Applying the calculation procedure described above, we will generate a series of cost-to-go functions $\hat{t}_n(s_n)$, and they have the property that all the cost-to-go functions $\hat{t}_k(s_k)$ generated at each stage k are only different in the linear term coefficient before x_{k-1} and the constant coefficient. The proof is given after the procedure.

- Case 2. If $k < n$

  We also have two cases to consider

  - Case 2a. If $s_k + a_k \notin \{s_{k+1}\}$. Similar to the situation described in Case 1a, x_k must be set as zero in this case. So we set $x_k^* = 0$ and define $\hat{t}_k(s) = \hat{t}_{k+1}(s)|_{x_k=0}$. Here, we should note that, for the state s_k in the stage of $(k+1)$, there might exist multiple $\hat{t}_{k+1}(s_k)$. Therefore, we need to find out the minimum one from these candidates. It has already been proved that all these candidate functions have the same nonlinear term coefficients and linear term coefficients except the linear term coefficient before x_{k+1} and the constant coefficient. For this reason, the best candidate function is the one with the minimum constant coefficient.

– Case 2b. If $s_k + a_k \in \{s_{k+1}\}$. Similar to the situation described in Case 1b, x_k can be either 0 or 1 in this case. But here we have following two different situations to discuss:

(i) At stage $k+1$, there exists only one cost-to-go function at $s_k + a_k$ and s_k respectively, i.e. $\hat{t}_{k+1}(s_k)$ and $\hat{t}_{k+1}(s_k + a_k)$, and the constant term coefficient as well as the sum of the constant term coefficient and the linear term coefficient before x_k of $\hat{t}_{k+1}(s_k + a_k)$ are less or equal to the counterparts of $\hat{t}_{k+1}(s_k)$. From the discussion in Case 2(a), we know that $\hat{t}_{k+1}(s_k + a_k)$ is either equivalent to or better than $\hat{t}_{k+1}(s_k)$ under this situation. We then apply the exact solution algorithm to $\hat{t}_{k+1}(s_k + a_k)$ and set x_k^* and $\hat{t}_k(s_k)$ using the procedure introduced in case 1(b).

(ii) For other situations except Situation (i), we shall set $x_k = 0$ in $\hat{t}_{k+1}(s_k)$ and set $x_k = 1$ in $\hat{t}_{k+1}(s_k + a_k)$, and assign these functions to be the cost-to-go functions $\hat{t}_k(s_k)$. Obviously, in this progress, multiple x_k^* and $\hat{t}_k(s_k)$ will be generated in a single state. Therefore the number of cost-to-go functions should be reduced to the minimum. For the cost-to-go function generated by setting $x_k = 0$ in $\hat{t}_{k+1}(s_k)$, since $\hat{t}_{k+1}(s_k)$ have the same nonlinear term coefficients and linear term coefficients except the linear term coefficient before x_{k+1} and the constant coefficient, then by setting $x_k = 0$ will result in the new functions that just different at the constant coefficient. Therefore, just the function that has the minimum constant coefficient should be chosen as the cost-to-go function for s_k. For the same reason, for the cost-to-go function that generated by setting $x_k = 1$ in $\hat{t}_{k+1}(s_k + a_k)$, we can also easily find out the best cost-to-go function for s_k. So, in this case, only two cost-to-go function will be generated with respect to s_k.

Lemma 4.4.1 *All the cost-to-go functions $\hat{t}_n(s_n)$ generated at stage n are only different in the linear term coefficient before x_{n-1} and the constant coefficient. Furthermore, this property holds for $\hat{t}_k(s)$ at stage k, $(k = n-1, \ldots, 2)$.*

Proof. From the description of the basic algorithm given in the previous section, we know that $\hat{t}_n(s_n)$ is calculated with the following equation:

$$\hat{t}_n(s_n) = f_n(x_1, \ldots, x_{n-1}, 0) + \phi_n(x_{n-1})(q_{n-1,n}x_{n-1} + c_n)$$

where $\phi_n(x_{n-1})$ can be one of the following four expressions, i.e. 0, 1, x_{n-1} and $1 - x_{n-1}$. Since both $q_{n-1,n}x_{n-1} + c_n$ and $\phi_n(x_{n-1})$ are linear functions of x_{n-1}, and in addition, x_{n-1}^2

can be replaced by x_{n-1}, then $\phi_n(x_{n-1})(q_{n-1,n}x_{n-1} + c_n)$ is also a linear function of x_{n-1}. Different assignments of $\phi_n(x_{n-1})$ will result in different linear term coefficient before x_{n-1}, i.e. $q_{n-1,n}$, and the constant coefficient , i.e. c_n, in the expression of $\phi_n(x_{n-1})(q_{n-1,n}x_{n-1} + c_n) \triangleq qx_{n-1} + c$. Therefore, we can get the conclusion that all the cost-to-go functions generated in stage n, $\hat{t}_n(s_n)$, have the same nonlinear term coefficients and linear term coefficients except the linear term coefficient before x_{n-1} and the constant coefficient.

With this property, we can express $\hat{t}_n(s_n)$ as

$$\begin{aligned}\hat{t}_n(s_n) &= f_n(x_1, \ldots, x_{n-2}, 0, 0) + q_{n-2,n-1}x_{n-2}x_{n-1} + c_{n-1}x_{n-1} + qx_{n-1} + c \\ &= f_n(x_1, \ldots, x_{n-2}, 0, 0) + x_{n-1}(q_{n-2,n-1}x_{n-2} + c_{n-1} + q) + c\end{aligned}$$

By assigning x_{n-1} with $\phi_{n-1}(x_{n-2})$, where $\phi_{n-1}(x_{n-2})$ can be one of the following four expressions, i.e. 0, 1, x_{n-2} and $1 - x_{n-2}$, we can get the cost-to-go function, $\hat{t}_{n-1}(s_{n-1})$, at stage $n-1$. Similar to the proof of stage n, we can get the conclusion that $\hat{t}_{n-1}(s_{n-1})$, have the same nonlinear term coefficients and linear term coefficients except the linear term coefficient before x_{n-2} and the constant coefficient.

Then apply the above proof to stage $(n-1)$ till stage 2, we can get the same conclusion that, at any stage k, $(k = 2, \ldots, n)$, $\hat{t}_k(s_k)$ are only different in the linear term coefficient before x_{k-1} and the constant coefficient. □

In the previous procedure, the maximum number of cost-to-go functions generated under different situations can be summarized with the following table.

Case	Maximum number of cost-to-go functions generated
Case 1a	1
Case 1b	1
Case 2a	1
Case 2b(i)	1
Case 2b(ii)	2

After this procedure, we get a complete table for all the states. Then by applying the tracking procedure, we can obtain the optimal solution to (CP). And it can be found from the above table that for each state at any stage, at most two cost-to-go functions will be generated.

Therefore, the number of cost-to-go functions generated at any stage will be less than the double of then number of stages at this stage.

4.4.1 A Simple Example

Consider the problem:

$$\min f(x) = 9x_1x_2 + 18x_2x_3 + 47x_3x_4 - 21x_4x_5 - 32x_1 + 42x_2 - 23x_3 - 12x_4 + 14x_5$$

$$\text{Subject to: } 3x_1 + 2x_2 + 3x_3 + 2x_4 + x_5 \leq 3$$

The optimal solution to the above problem is known to be $x = (1\ 0\ 0\ 0\ 0)^T$. Now, We use our algorithm to find out how it works to get the above optimal solution.

By applying the algorithm described above, we can get the following five tables.

We first generate the table for s_5. Since in this example $n = 5$, the table should be constructed according to the procedure described in Case 1 of the algorithm.

s_5	feasible region for x_5	x_5^*	$\hat{t}_5(s_5)$
0	{0,1}	x_4	$9x_1x_2 + 18x_2x_3 + 47x_3x_4 - 32x_1 + 42x_2 - 23x_3 - 19x_4$
1	{0,1}	x_4	$9x_1x_2 + 18x_2x_3 + 47x_3x_4 - 32x_1 + 42x_2 - 23x_3 - 19x_4$
2	{0,1}	x_4	$9x_1x_2 + 18x_2x_3 + 47x_3x_4 - 32x_1 + 42x_2 - 23x_3 - 19x_4$
3	0	0	$9x_1x_2 + 18x_2x_3 + 47x_3x_4 - 32x_1 + 42x_2 - 23x_3 - 12x_4$

In this table, we have four possible states, i.e. $s_5 = 0, 1, 2$ and 3. For $s_5 = 0, 1$ and 2, $s_5 + a_5 = s_5 + 1 \leq b = 3$ is satisfied. Therefore, we can get the value of x_5^* and $\hat{t}_5(s)$ from Case 1 (b). Since $s_5 = 3$, $s_5 + a_5 \leq b$ can not be satisfied, x_n^* and $\hat{t}_5(s)$ should be determined with the procedure described in Case 1 (a).

Then we generate the table for s_4. Since $k = 4 < n = 5$, the remaining tables should be constructed according to the procedure described in Case 2 of the algorithm.

s_4	feasible region for x_4	x_4^*	$\hat{t}_4(s_4)$
0	{0,1}	$1 - x_3$	$9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 - 4x_3 - 19$
1	{0,1}	0	$9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 - 23x_3$
1	{0,1}	1	$9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 + 24x_3 - 12$
2	0	0	$9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 - 23x_3$
3	0	0	$9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 - 23x_3$

In this table, for $s_4 = 0$, $s_4 + a_4 = s_4 + 3 \in \{s_5\} = \{0, 1, 2, 3\}$ is satisfied. And it can be easily found out that when $s_4 = 0$, the conditions coincide with the Situation (i) in Case 2(b). Therefore, we can get the value of x_4^* and $\hat{t}_4(s)$ from Situation (i) in Case 2(b). Similarly, two cost-to-go functions can be defined for $s_4 = 1$ with the procedure described in Situation (ii) in Case 2(b). While for $s_4 = 2$ and 3, the condition $s_4 + a_4 \in \{s_5\}$ can not be satisfied. Therefore, for these two states, x_4^* and $\hat{t}_4(s)$ can be determined by the approach introduced in Case 2(a).

We further generate the table for s_3.

s_3	feasible region for x_3	x_3^*	$\hat{t}_3(s_3)$
0	{0,1}	0	$9x_1x_2 - 32x_1 + 42x_2 - 19$
0	{0,1}	1	$9x_1x_2 - 32x_1 + 42x_2 - 23$
1	0	0	$9x_1x_2 - 32x_1 + 42x_2 - 12$
2	0	0	$9x_1x_2 - 32x_1 + 42x_2$
3	0	0	$9x_1x_2 - 32x_1 + 42x_2$

For $s_3 = 0$, x_3^* and $\hat{t}_3(s)$ can be determined by the method described in Situation (ii) in Case 2(b). Then for $s_3 = 1$, x_3^* and $\hat{t}_3(s)$ can be determined by the procedure described in Case 2(a). Here we should be noted that, for this state, we have two candidates to define the cost-to-go function. Then apply the method described in the algorithm, one candidate, i.e., $9x_1x_2 + 18x_2x_3 - 32x_1 + 42x_2 + 24x_3 - 12$, is determined to be the best one. For $s_3 = 2$ and 3, the method described in Case 2(a) should be applied.

We then generate the table for s_2.

s_2	feasible region for x_2	x_2^*	$\hat{t}_2(s_2)$
0	{0,1}	0	$-32x_1 - 23$
1	{0,1}	0	$-32x_1 - 12$
2	0	0	$-32x_1$
3	0	0	$-32x_1$

For $s_2 = 0$, three cost-to-go functions will be generated by using the algorithm described in Situation (ii) in Case 2(b). The three functions are $-32x_1 - 19$, $-32x_1 - 23$ and $-32x_1$. Only the best one $-32x_1 - 23$ is kept. Likewise, for $s_2 = 1$, two cost-to-go functions $-32x_1 - 12$ and $-32x_1$ can be generated, and only $-32x_1 - 12$ is kept.

We finally generate the table for s_1.

s_1	feasible region for x_1	x_1^*	$\hat{t}_1(s_1)$
0	{0,1}	0	-23
0	{0,1}	1	-32

From the last table, we can find that when $x_1 = 1$ the value of $\hat{t}_1(s_1)$ is the smallest. Then we use the tracking procedure to find out the optimal value of x_2, x_3, x_4 and x_5 as follows:

$$
\begin{aligned}
s_2^* &= s_1 + a_1 x_1^* = 0 + 3 \times 1 = 3 \\
x_2^* &= x_2^*(s_2^*) = x_2^*(3) = 0 \\
s_3^* &= s_2 + a_2 x_2^* = 3 + 2 \times 0 = 3 \\
x_3^* &= x_3^*(s_3^*) = x_3^*(3) = 0 \\
s_4^* &= s_3 + a_3 x_3^* = 3 + 3 \times 0 = 3 \\
x_4^* &= x_4^*(s_4^*) = x_4^*(3) = 0 \\
s_5^* &= s_4 + a_4 x_4^* = 3 + 2 \times 0 = 3 \\
x_5^* &= x_5^*(s_5^*) = x_5^*(3) = 0
\end{aligned}
$$

Then we get the optimal result, i.e. $x = (1\,0\,0\,0\,0)^T$.

4.5 Computational Results

In this section, we present numerical results for the algorithms proposed in Section 4.3. The main purpose of our computational experiments is to test the efficiency of methods discussed in the previous sections.

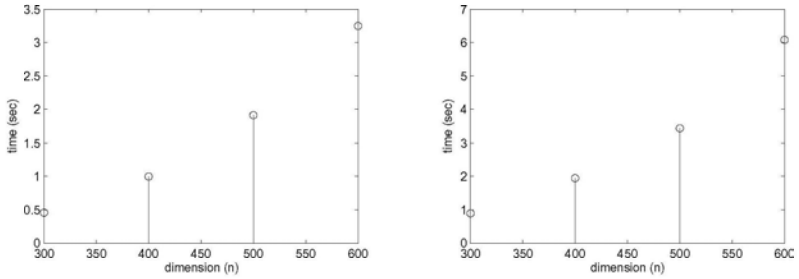
The algorithm was coded by Matlab R14 and run on a Pentium IV PC. Numerical results are reported in Tables 4.3, 4.4 and 4.5. Table 4.3 and 4.4 provide the computation time of our algorithm for solving different dimensional (UP) where Q is tri-diagonal matrix and five-diagonal matrix respectively. Table 4.5 provides the computation tim of our algorithm for solving (CP). These three tables indicate that our algorithms have considerably good performance in terms of computational time.

Table 4.3: Experimental results for UP where Q is a tri-diagonal matrix

$n =$	300	400	500	600
Round 1	0.4531	1.0625	1.9531	3.2656
Round 2	0.4219	0.9531	1.8594	3.2031
Round 3	0.4063	0.9844	1.9063	3.3125
Round 4	0.3906	0.9531	1.8906	3.2656
Round 5	0.4844	1.0625	1.8594	3.2188
Round 6	0.4688	0.9688	1.8750	3.1875
Round 7	0.4375	1.0625	1.8594	3.3438
Round 8	0.5625	0.9219	1.9219	3.1719
Round 9	0.5000	1.0000	1.9531	3.1875
Round 10	0.4375	1.0156	2.0156	3.3438
Max time	0.5625	1.0625	2.0156	3.3438
Min time	0.3906	0.9219	1.8594	3.1719
Average time	0.4563	0.9984	1.9094	3.2500

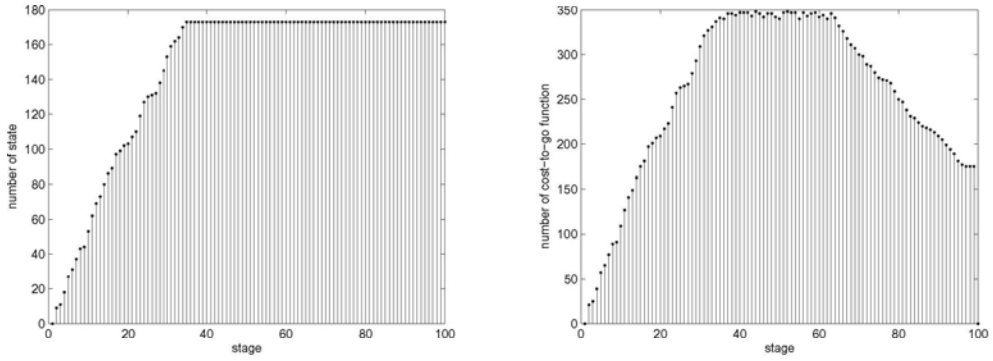
Based on the information listed in Table 4.3 and 4.4, the average computational times for problems with different dimension are plotted in Figure 4.1(a) and 4.1(b). It is obvious that when the dimension of problem increases the computational time increased linearly.

For the algorithm to (CP), it has been proved that for each state at every stage, at most two cost-to-go functions will be assigned. Figure 4.2(a) and 4.2(b) can obviously reveal this



(a) Q being a tri-diagonal matrix (b) Q being a five-diagonal matrix

Figure 4.1: Average computational time for different dimensional UP with Q being a tri-diagonal or five-diagonal matrix



(a) Number of states (b) Number of cost-to-go functions

Figure 4.2: Number of states and cost-to-go functions at every stage for one example with $n = 100$

Table 4.4: Experimental results for UP where Q is a five-diagonal matrix

$n =$	300	400	500	600
Round 1	0.9531	2.0313	3.5000	6.0156
Round 2	0.9219	1.9375	3.3750	5.8594
Round 3	0.8594	1.9531	3.5313	5.8750
Round 4	0.8750	1.9531	3.4375	5.8906
Round 5	0.8438	1.8906	3.4375	5.8594
Round 6	0.9063	1.9688	3.3906	5.9219
Round 7	0.8750	1.9219	3.5000	5.9375
Round 8	0.8594	1.9219	3.3906	5.9063
Round 9	0.9219	1.9063	3.3594	6.0156
Round 10	0.8906	2.0156	3.3594	7.5156
Max time	0.9531	2.0313	3.5000	7.5156
Min time	0.8438	1.8906	3.3594	5.8594
Average time	0.8906	1.9500	3.4281	6.0797

important property. We apply our algorithm to a problem with 100 variables. Figure 4.2(a) shows the number of states of each stage for this problem. Figure 4.2(b) shows the number of cost-to-go functions that generated at each stage. It can be obviously found out that, at each stage, the number of cost-to-go function is less than or equal to the double of the number of state. That is to say, the complexity of the algorithm will not increase exponentially with respect to the dimension of the problem. The computational results reveal that our algorithms are very effective and efficient.

□ End of chapter.

Table 4.5: Experimental results for CP

$n =$	10	25	40	100
Round 1	0.2500	2.6406	2.8438	13.2969
Round 2	0.1875	1.6406	1.5469	62.8594
Round 3	0.1563	0.8750	1.5000	51.6875
Round 4	0.1406	0.4688	1.6094	18.2344
Round 5	0.1875	1.9531	3.6250	60.2344
Round 6	0.2813	0.6094	5.1719	12.5156
Round 7	0.1563	1.0156	10.2031	65.5625
Round 8	0.1250	0.6719	5.8750	49.4375
Round 9	0.1406	2.5625	8.4688	59.5781
Round 10	0.1094	0.9219	2.3438	37.7031
Max time	0.2813	2.6406	10.2031	65.5625
Min time	0.1094	0.4688	1.5000	12.5156
Average time	0.1735	1.3360	4.3188	43.1109

Chapter 5

A New Algorithm to Find Integer Solutions to Linear Equations

5.1 Brief Introduction to the Methods for Finding Integer Solutions to Linear Equations

Consider the linear Diophantine equations: $Ax = b, x \in \mathbb{Z}^n$ and the Diophantine equations on a bounded integer set: $Ax = b, x \in X = \{x \in \mathbb{Z}^n | l \leq x \leq u\}$, where $\mathbb{Z}^n$ denotes the set all integer vectors in $\mathbb{R}^n$, A is an $m \times n$ integral matrix with $m < n$ and rank$(A) = m$, and $b \in \mathbb{R}^m$ is integral. It is well known that linear diophantine equations are polynomially solvable, while linear diophantine equations on a bounded integer set are NP-complete, as the special case of linear Diophantine Equations with $m = 1$, $x \in \{-1, 1\}^n$ and $b = 0$ is NP-complete.

The most classical method in solving linear Diophantine equations is the Smith normal form, which invokes the identification of two unimodular matrices $U \in \mathbb{Z}^{m \times m}$ and $V \in \mathbb{Z}^{n \times n}$ such that $UAV = [S, 0]$ with $S = diag(s_1, s_2, \ldots, s_m), s_i | s_{i+1}, 1 \leq i \leq m - 1$. Furthermore, there exists integer solution to the linear Diophantine equations if and only if $y_i^0 = d_i / s_i, i = 1, \ldots, m$, are integers, and if yes, every integer solution takes the form of $x = Vy = Vy^0 + \sum_{i=1}^{n-m} k_i V y^i, k_i \in \mathbb{Z}$, where $y_i^0 = 0, i = m + 1, \ldots, n$ and $y^1 = e_{m+1}, \ldots, y^{n-m} = e_n$.

The most popular method in solving linear Diophantine equations is the so called Hermite Normal Form, which states that there exists an $n \times n$ unimodular matrix C such that $AC = (H, 0)$, with H being in Hermite normal form. Furthermore, there exists integer solution to

the linear Diophantine equations if and only if $H^{-1}b \in \mathbb{Z}^m$, and, if yes, every integer solution takes the form of $x = C_1 H^{-1} b + C_2 z, z \in \mathbb{Z}^{n-m}$, where C_1 and C_2 are the first m rows and last $n - m$ rows of C, respectively.

Based on the Hermite normal form, many algorithms have been designed for finding integer solutions to linear Diophantine equations. As the computation of the Smith normal form and the Hermite normal form of integer matrix A plays a central role in finding integer solutions to linear Diophantine equations, some methods, such as the Euclidean algorithm in [Nemhauser and Wolsey 1988], have been devised to improve the algorithmic efficiency. However, a notorious phenomenon of coefficient explosion gives rise a major obstacle in such computations. Various strategies have been proposed to alleviate this computational difficulty.

Under the linear transformation $x = y + l$, solving linear Diophantine equations over bounded integer set $[l, u]$ is equivalent to solving the following problem: $Ay = d, 0 \leq y \leq \beta$, $y \in \mathbb{Z}^n$, where $d = b - Al$ and $\beta = u - l$. Based on lattice basis reduction, an algorithm was developed to identify if there exists a $y \in \mathbb{Z}^n$ satisfying bound constraints. In fact, the algorithm is to firstly derive the Hermite normal form, based on the Lovassz basis reduction algorithm which possesses good capability in avoiding coefficient explosion, to obtain a short solution x_d and a short basis x_λ to system $Ay = d$. Secondly, branching on integer linear combinations of x_λ to obtain a solution that satisfies bound constraints, or to prove an infeasibility.

Utilizing the results of [21] to express explicitly integer solutions to a linear equation of two variables, an algorithm based on the Euclid's algorithm for computing the set of integer solution of $Ax = b$ on bounded set $X$is developed [22]. In its first phase, the algorithm of Ramachandran reduces the problem recursively by aggregating two variables into an artificial variable with calculated lower and upper bounds, finally into a linear system with only two variables whose integer solutions can be specified. In the second phase of the algorithm, by repeating using the results for a linear equation of two variables, the solution set is expanded by determining progressively integer values for remaining undecided elements of x.

5.2 Finding Integer Solution with Cell Enumeration Method

In this section, we propose a new algorithm with cell enumeration method for finding integer solution to linear equations. Let $U = (U_1, \ldots, U_{n-m})$ be an orthogonal basis for the null

space of A and x_0 be a special solution to $Ax = b$ in R^n, then $C = \{x \in R^n \mid Ax = b\}$ can be expressed by

$$\begin{aligned} C &= \{x \in R^n \mid x = x_0 + \sum_{i=1}^{n-m} z_i U_i, \\ &\qquad z_i \in R,\ i = 1, \ldots, n-m\}. \end{aligned} \tag{1}$$

Consider the Euclidian distance from C to $X = \{x \in Z^n \mid 0 \le x_i \le u_i,\ i = 1, \ldots, n\}$:

$$\delta = \mathrm{dist}(C, X) = \min\{\|x - y\| \mid x \in X,\ y \in C\}. \tag{2}$$

It is obvious that $Ax = b$ has a solution in X if and only if $\delta = 0$. Furthermore, when $\delta = 0$, any $x^* \in X$ that achieves the zero distance is an integer solution to the linear equations $Ax = b$ in X.

Note that combining (1) and (2) yields

$$\begin{aligned} \delta &= \mathrm{dist}(C, X) \\ &= \min_{y \in C} \min\{\|y - x\| \mid 0 \le x_i \le u_i,\ x_i \in Z,\ i = 1, \ldots, n\} \\ &= \min_{y \in C} \|y - \varphi(y)\|, \end{aligned}$$

where, for $j = 1, \ldots, n$, $\varphi(y)_j$ is determined by the following equation:

$$\varphi(y)_j = \begin{cases} 0, & y_j \in (-\infty, \frac{1}{2}], \\ i, & y_j \in (i - \frac{1}{2}, i + \frac{1}{2}),\ i \in \{1, \ldots, u_j - 1\}, \\ u_j, & y_j \in [u_j - \frac{1}{2}, \infty). \end{cases}$$

Let's consider the hyperplane arrangement generated by the following $\sum_{j=1}^{n} u_j$ hyperplanes in R^{n-m}:

$$\begin{aligned} h_{ij} &= \{z \in R^{n-m} \mid g_{ij}(z) := (x_0)_j \\ &\qquad + \sum_{l=1}^{n-m} z_l U_{lj} - (i - \frac{1}{2}) = 0\} \end{aligned} \tag{3}$$

for $i = 1, \ldots, u_j$, $j = 1, \ldots, n$. Note that for each j, h_{ij} $(i = 1, \ldots, u_j)$ are parallel hyperplanes. A cell E of the hyperplane arrangement corresponding to h_{ij}'s is an $(n - m)$-dimensional polyhedral set formed by the half-spaces induced by h_{ij}'s and can be characterized by a $\sum_{j=1}^{n} u_j$-dimensional sign vector, $\mathrm{sign}(E) = (w^1, \ldots, w^n)$, where $w^j =$

$(w_{1,j}, \ldots, w_{u_j,j})$ is specified by

$$w_{ij} = \begin{cases} +, & \text{if } g_{ij}(\pi) > 0 \\ -, & \text{if } g_{ij}(\pi) < 0 \end{cases}, \quad i = 1, \ldots, u_j, \tag{4}$$

with π being an interior point of E. For fixed j, since h_{ij} $(i = 1, \ldots, u_j)$ are parallel hyperplanes, $w^j = (w_{1,j}, \ldots, w_{u_j,j})$ must take the following form:

$$(\overbrace{+, \ldots, +}^{i}, -, \ldots, -), \quad i = 0, 1, \ldots, u_j.$$

The one-to-one map between all distinct $\varphi(y)$ for $y \in C$ and the sign vectors of all the cells of the hyperplane arrangement can be established as follows:

$$w = (w^1, \ldots, w^n) \longleftrightarrow \varphi = (\varphi_1, \ldots, \varphi_n),$$

where for $j = 1, \ldots, n$,

$$\begin{aligned} w^j = (-, \ldots, -) &\longleftrightarrow \varphi_j = 0, \\ w^j = (+, \ldots, +) &\longleftrightarrow \varphi_j = u_j, \\ w^j = (\overbrace{+, \ldots, +}^{i}, -, \ldots, -) &\longleftrightarrow \varphi_j = i, \\ \text{where } i \in \{1, \ldots, u_j - 1\}&. \end{aligned}$$

It has been known that the number of cells of the hyperplane arrangement generated by (3) is bounded by $O((n\omega)^{n-m})$, where $\omega = \max_{i=1,\ldots,n} u_i$ [56, 58]. Moreover, using the cell enumeration methods in [24] and [54], we can find all the cells of the hyperplane arrangement generated by (3) in $O((n\omega)^{n-m})$ time. Listing all the distinct $\varphi(y)$ for $y \in C$ as $\varphi^1, \ldots, \varphi^k$, the distance $\delta = \text{dist}(C, X)$ can then be calculated as follows:

$$\begin{aligned} \delta &= \min_{j=1,\ldots,k} \delta_j \equiv \min_{j=1,\ldots,k} \text{dist}(C, \varphi^j) \\ &= \min_{j=1,\ldots,k} \|(UU^T - I_n)(\varphi^j - x_0)\|. \end{aligned} \tag{5}$$

Therefore, finding a solution of the linear system $Ax = b$ over the integer set X or checking its infeasibility can be done in $O((n\omega)^{n-m})$ time. Especially, when $r = n - m$ is fixed $(0 \leq r \leq n - 1)$, the linear system $Ax = b$ over X is polynomially solvable.

5.3 An Illustrative Example

Let us consider now a simple example as follows to illustrate the solution algorithm for the following set of linear equations using the cell enumeration method.

$$\begin{pmatrix} -6 & 3 & 9 & -2 & -2 \\ 9 & -5 & -3 & 3 & -1 \\ 2 & 1 & 4 & 4 & 4 \end{pmatrix} x = \begin{pmatrix} 6 \\ 1 \\ 7 \end{pmatrix}.$$

where $x \in X = \{x \in Z^6 \mid -1 \leq x_1 \leq 2, -1 \leq x_2, x_3 \leq 1, -2 \leq x_4, x_5 \leq 0\}$.

Since the matrix is of rank 3 in this example, solving the fundamental equation in R^5 yields

$$\begin{aligned} C &= \{x \in R^5 \mid Ax = b\} \\ &= \{x \in R^5 \mid x = x_0 + z_1 U_1 + z_2 U_2, z_i \in R, i = 1, 2\}, \end{aligned}$$

where

$$x_0 = (0.3182, 0.0043, 1.0069, 0.4404, 0.1425)^T$$

and

$$U = (U_1, U_2) = \begin{pmatrix} -0.4513 & -0.3197 \\ -0.1833 & -0.8280 \\ -0.1468 & 0.1181 \\ 0.7808 & -0.1649 \\ -0.3625 & 0.4136 \end{pmatrix}.$$

With the information of U, x_0 and the bound of X, we generates the following 11 hyperplanes

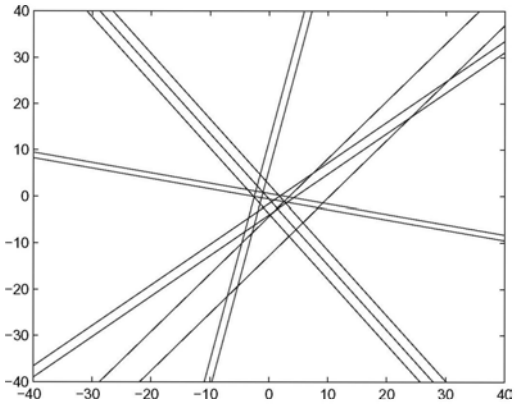


Figure 5.1: Cell arrangement for the example problem.

on C defined in (3),

$$
\begin{aligned}
j=1: &\quad \begin{cases} g_{11}(z) = -0.4513z_1 - 0.3197z_2 = -0.8182, \\ g_{21}(z) = -0.4513z_1 - 0.3197z_2 = 0.1818, \\ g_{31}(z) = -0.4513z_1 - 0.3197z_2 = 1.1818, \end{cases} \\
j=2: &\quad \begin{cases} g_{12}(z) = -0.1833z_1 - 0.8280z_2 = -0.5043, \\ g_{22}(z) = -0.1833z_1 - 0.8280z_2 = 0.4957, \end{cases} \\
j=3: &\quad \begin{cases} g_{13}(z) = -0.1468z_1 + 0.1181z_2 = -1.5069, \\ g_{23}(z) = -0.1468z_1 + 0.1181z_2 = -0.5069, \end{cases} \\
j=4: &\quad \begin{cases} g_{14}(z) = 0.7808z_1 - 0.1649z_2 = -1.9404, \\ g_{24}(z) = 0.7808z_1 - 0.1649z_2 = -0.9404, \end{cases} \\
j=5: &\quad \begin{cases} g_{15}(z) = -0.3625z_1 + 0.4136z_2 = -1.6425, \\ g_{25}(z) = -0.3625z_1 + 0.4136z_2 = -0.6425, \end{cases}
\end{aligned}
$$

As illustrated in Fig. 5.1, these 11 hyperplanes partition set C into 60 cells, each of which corresponds to a unique sign vector defined in (4), $\text{sign}(E) = (w^1, \ldots, w^5)$, where $w^1 = (w_{11}, w_{21}, w_{31})$ and $w^j = (w_{1j}, w_{2j})$, $(j = 2, \ldots, 5)$. Table 1 lists the integer vector according to all these 60 cells.

Let us take Cell 1 in Fig. 5.1 as an examplt. As any interior point π of Cell 1 satisfies $g_{11}(\pi) > 0$, $g_{21}(\pi) < 0$, $g_{31}(\pi) < 0$, $g_{12}(\pi) < 0$, $g_{22}(\pi) < 0$, $g_{13}(\pi) > 0$, $g_{23}(\pi) > 0$, $g_{14}(\pi)$

> 0, $g_{24}(\pi) > 0$, $g_{15}(\pi) > 0$, and $g_{25}(\pi) > 0$, the sign vector for any interior point π of Cell 5 is thus specified by $((+--)(--)(++)(++)(++))$. Furthermore, for all points in Cell 1, the closest integer point in $X = \{0, 1, 2\}^6$ is clearly given by $\varphi^1 = (0, -1, 1, 0, 0)$.

Then we calculate the distance between φ^i and Cell i $(i = 1, \ldots, 60)$ using the following function:

$$\delta_i = \text{dist}(X, \text{Cell } i) = \|(UU^T - I_5)(\varphi^i - x_0)\|$$

We find that, for Cell 21, $\delta_{21} = 0$. That is to say $\varphi^{21} = (1, 1, 1, 0, 0)^T$ is the solution to $Ax = b$.

□ End of chapter.

Table 5.1: A full list of cells and their closest integer points.

Cell j	$\wp^j$
1	$(0,-1,1,0,0)^T$
2	$(-1,-1,1,0,0)^T$
3	$(-1,0,1,0,0)^T$
4	$(-1,0,1,0,-1)^T$
5	$(-1,0,0,0,-1)^T$
6	$(-1,1,0,0,-1)^T$
7	$(-1,0,0,0,-2)^T$
8	$(-1,1,0,0,-2)^T$
9	$(-1,0,-1,0,-2)^T$
10	$(-1,-1,0,0,0)^T$
11	$(-1,-1,-1,0,0)^T$
12	$(-1,-1,0,0,-1)^T$
13	$(-1,-1,-1,0,-1)^T$
14	$(-1,-1,0,0,-2)^T$
15	$(-1,-1,-1,0,-2)^T$
16	$(-1,-1,1,-1,0)^T$
17	$(-1,-1,1,0,-1)^T$
18	$(1,-1,1,0,0)^T$
19	$(0,0,1,0,0)^T$
20	$(1,0,1,0,0)^T$
21*	$(1,1,1,0,0)^T$
22	$(2,1,1,0,0)^T$
23	$(2,1,1,0,-1)^T$
24	$(2,1,1,0,-2)^T$
25	$(2,1,0,0,-2)^T$
26	$(2,1,-1,0,-2)^T$
27	$(2,1,-1,-1,-2)^T$
28	$(2,1,0,-1,-2)^T$
29	$(1,1,1,0,-1)^T$
30	$(1,1,1,0,-2)^T$
31	$(1,0,1,-1,0)^T$
32	$(1,1,1,-1,0)^T$
33	$(2,1,1,-1,0)^T$
34	$(2,1,1,-2,0)^T$
35	$(2,1,1,-2,-1)^T$
36	$(2,1,1,-2,-2)^T$
37	$(2,1,0,-2,-2)^T$
38	$(2,1,-1,-2,-2)^T$
39	$(2,1,1,-1,-1)^T$
40	$(2,1,1,-1,-2)^T$
41	$(1,1,1,-2,0)^T$
42	$(1,0,1,-2,0)^T$
43	$(0,1,1,0,0)^T$
44	$(0,0,1,0,-)^T$
45	$(0,1,1,0,-1)^T$
46	$(0,1,0,0,-1)^T$
47	$(1,1,0,0,-1)^T$
48	$(0,1,0,0,-2)^T$
49	$(1,1,0,0,-2)^T$
50	$(0,1,-1,0,-2)^T$
51	$(-1,1,-1,0,-2)^T$
52	$(1,1,-1,0,-2)^T$
53	$(0,0,0,0,-1)^T$
54	$(0,-1,1,-1,0)^T$
55	$(1,-1,1,-1,0)^T$
56	$(0,-1,1,-2,0)^T$
57	$(-1,-1,1,-2,0)^T$
58	$(1,-1,1,-2,0)^T$
59	$(2,-1,1,-2,0)^T$
60	$(2,0,1,-2,0)^T$

$*$ denotes the cells corresponding to the integer solutions of $Ax = b$.

Chapter 6

Reachability Analysis of Petri Nets

6.1 Brief Introduction to Petri Nets and Reachability Analysis

Petri net, introduced by Petri in his seminal work "Communication with Automata" [64], has been a promising mathematical formalism for modeling, analyzing and designing discrete event systems, especially by its remarkable capability in modeling process synchronization, asynchronous events, concurrent and distributed operations and conflicts or resource sharing. The past four decades have witnessed innumerable successful applications of Petri nets in various areas, such as i) modeling and analysis of communication protocols and networks, production systems, sequence controllers and software development, and ii) performance evaluation of the modeled systems.

The Petri net is a particular kind of bipartite directed graph consisting of three types of elements: places, transitions, and directed arcs connecting places and transitions, while each place may hold a non-negative number of tokens. In a graphical representation of a Petri net, places are depicted by circles and transitions as bars. With these three types of elements, Petri net may be used to model various systems. See Fig. 6.1 for an illustrative example of Petri nets with 4 places and 6 transitions. The distribution of tokens on places, called Petri net marking, defines the state of the modeled system. A marking for a Petri net with m places is represented by an $(m \times 1)$ vector M, where M_j, $j = 1, 2, \ldots, m$, are nonnegative integers representing the number of tokens in the corresponding places.

A generalized Petri net is a five-tuple defined as follows:

PN$= (P, T, D_I, D_O, M_0)$; where

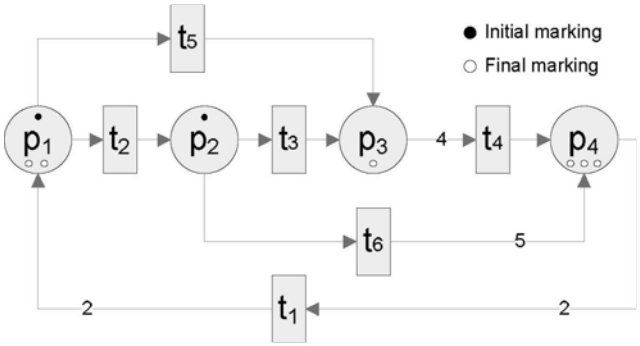


Figure 6.1: Example of a Petri net

1. $P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,

2. $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$,

3. D_I: $(P \times T) \mapsto Z_+^{m \times n}$ is an input function that defines weights associated with directed arcs from places to transitions, where $Z_+^{m \times n}$ (Z_+^m) is the set of $(m \times n)$ dimensional matrices (m dimensional vector) with all entries being in Z_+,

4. D_O: $(P \times T) \mapsto Z_+^{m \times n}$ is an output function that defines weights associated with directed arcs from transitions to places , and

5. $M_0 : P \mapsto Z_+^m$ is the initial marking.

For the Petri net given in Figure 6.1, we have

$$D_I = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 4 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$D_O = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 5 \end{pmatrix},$$

and $M_0 = (1, 1, 0, 0)^T$.

The change of the distribution of the tokens represents the dynamics of the modeled system, while the distribution of tokens on places may change according to the following enabling rule and firing rule [69]:

Enabling Rule: A transition t is said to be enabled if each input place p of t contains at least the number of tokens equal to the weight of the arc connecting p to t.

Firing Rule: (a) An enabled transition t may or may not fire depending on the additional interpretation, and (b) A firing of an enabled transition t removes form each input place p the number of tokens equal to the weight of the directed arc connecting p to t, and deposits in each output place p the number of tokens equal to the weight of the arc connecting t to p.

Petri nets have been used in many well known engineering application fields including modeling flexible manufacturing systems, workflow systems and sequence controllers, analysis of communications protocols and software development, and performance evaluation of multiprocessor systems and parallel computer architectures.

6.2 Solving the Reachability Analysis Problem by Finding the Integer Solutions to the Fundamental Equation

Reachability analysis is no doubt one of the most important behavioral properties of Petri nets. Given both an initial state M_0 and a target terminal state M, a natural question to ask is whether or not we have a sequence of firing rules such that the system can reach the specific target state within finite steps. There are two primary approaches in investigating reachability: reachability graph analysis and reachability algebraic analysis. Reachability graph analysis is based on the creation and investigation of a reachability graph or a reduced reachability graph. As this approach relies on exhaustively generating all the reachable markings from a given initial marking, it suffers the state explosion phenomenon, while transferring the reachability graph to a reduced counterpart is NP-hard [60]. The second approach is based on methods of linear algebra. It is well known that a necessary condition for reachability of marking M from an initial marking M_0 of a Petri net is that there exists a nonnegative integer vector solution of the following system of linear equations [63, 68],

$$Ax = b, \tag{1}$$

where $b = M - M_0$, $x = (x_1, x_2, \ldots, x_m)^T$ is a firing count vector and A is an $(m \times n)$ dimensional incidence matrix given by $A = [a(p_i, t_j)] = [D_O(p_i, t_j) - D_I(p_i, t_j)]$. When the Petri net is acyclic, the above condition becomes both necessary and sufficient [61]. Equation (1) is also termed the *fundamental equation* in Petri nets. For example, if we set in Figure 6.1 the final marking to be $M = (2, 0, 1, 3)^T$, the fundamental equation for this example is then

given by

$$\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & -1 \\ 0 & 0 & 1 & -4 & 1 & 0 \\ -2 & 0 & 0 & 1 & 0 & 5 \end{pmatrix} x = \begin{pmatrix} 1 \\ -1 \\ 1 \\ 3 \end{pmatrix}.$$

The above discussion reveals that the investigation of reachability can be achieved, to certain degree, by finding out nonnegative integer solution(s) to a system of linear equations. Thererfore, the first step in the reachability algebraic analysis is to find firing count vectors, which can be performed either by solving the fundamental equation directly, or by solving the corresponding integer programming problem. Unfortunately, finding nonnegative integer solutions to the fundamental equation in Petri nets is NP-complete [62]. In essence, solving the fundamental equation is an integer programming problem (see, for example, [66] [47] and [44]). Generally speaking, the capability of the existing methods for finding the nonnegative integer solution to the fundamental equation (1) is still very limited. In particular, the Petri nets community has been largely dependent on some existing softwares to obtain nonnegative integer solutions to the fundamental equation. More efficient solution schemes that invoke the state-of-the-art from some other related subjects, especially from discrete optimization and integer programming, deserve further investigation. There have been a few attempts in the Petri net communities in this direction. For example, [62] applied Groebner Bases method from integer programming to solve reachability problems in Petri nets.

The algorithm of finding the negative integer solutions to $Ax = b$ with Groebner Bases, where A is a $m \times n$ matrix and b is a column vector with m elements, is given as follows:

1. Define $\varphi_i(w) = \prod_{j=1}^{n} z_i^{a_{ij}}$, $(i = 1, \ldots, m)$;

2. If $a_{ij} < 0$, then apply $t = \prod_{k=1}^{m} z_k^{-1}$ to $z_i^{a_{ij}}$ such that all the powers appears in $\varphi_i(w)$ are greater or equal to zero;

3. Define an ideal $J =< t\prod_{k=1}^{m} z_k - 1, \varphi_1(w) - w_1, \ldots, \varphi_m(w) - w_m >$;

4. Calculate the Groebner Bases, g, for J

5. Define $f = \prod_{i=1}^{m} z_i^{b_i}$. And if there exists $b_i < 0$, apply $t = \prod_{k=1}^{m} z_k^{-1}$ to $z_i^{b_i}$ such that all the powers appears in f are greater or equal to zero;

6. Calculate $\bar{f}^g$, If $\bar{f}^g$ is in the form $\bar{f}^g = \prod_{i=1}^{m} w_i^{\alpha_i}$ and α_i is nonnegative integer, then $(\alpha_1, \ldots, \alpha_m)$ is the solution to the original problem.

From the above algorithm, it can be easily found that the solution approach using Groebner Bases depends heavily on symbolic computing power, and its capability in solving large-scale problem is thus not promising.

We propose in Chapter 5 a novel method based on cell enumeration approach for finding integer solutions of a linear system $Ax = b$ over a bounded integer set X. This cell enumeration approach provides a promising platform for designing an efficient method with a complexity of $O((n\omega)^{n-m})$, where $\omega = \max_{i=1,\ldots,n} u_i$, to find integer solutions to linear equations on a bounded integer set. Although the original analytical reachability analysis is to find nonnegative integer solution(s) to the fundamental equation, a more practical problem is to find integer solution(s) to the fundamental equation on a bounded integer set, $X = \{x \in Z^n \mid 0 \leq x_i \leq u_i,\ i = 1, \ldots, n\}$, where Z^n denotes the set of all integer points in R^n. This confinement is reasonable as there always exist resource constraints in real-world applications of Petri nets. For example, in chemical processes, the operation of reactors (transitions in a corresponding Petri net) is expensive, thus the number of such operations should not exceed a given upper bound. As a result, the algorithm we proposed can be applied in reachability analysis in Petri net.

Let us consider now the example in Fig. 6.1 to illustrate the solution algorithm for the fundamental equation using the cell enumeration method. For this instance, we assume that the bounded integer set is given by $X = \{x \in Z^6 \mid 0 \leq x_i \leq 2,\ i = 1, \ldots, 6\}$.

Since the incidence matrix is of rank 4 in this example, solving the fundamental equation in R^6 yields

$$\begin{aligned} C &= \{x \in R^6 \mid Ax = b\} \\ &= \{x \in R^6 \mid x = x_0 + z_1 U_1 + z_2 U_2, z_i \in R, i = 1, 2\}, \end{aligned}$$

where

$$x_0 = (0.2912, -0.2138, 0.0100, -0.2984, -0.2038, 0.776)^T$$

and

$$U = (U_1, U_2) = \begin{pmatrix} 0.3508 & 0.3944 \\ -0.1605 & 0.6882 \\ -0.2713 & 0.5636 \\ 0.1477 & 0.1661 \\ 0.8622 & 0.1006 \\ 0.1108 & 0.1245 \end{pmatrix}.$$

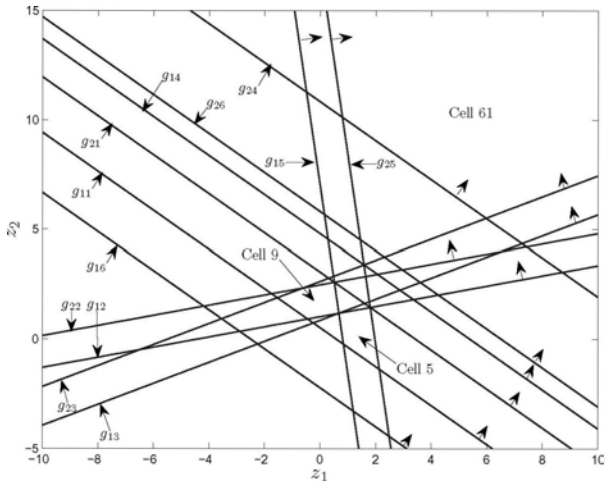


Figure 6.2: Cell arrangement for the example problem.

Note that u_j = 2, j = 1, . . ., 6. Setting each x_j, j = 1, . . ., 6, equal to $\frac{1}{2}$ and $\frac{3}{2}$, respectively, generates the following 12 hyperplanes on C defined in (3),

$$
\begin{aligned}
j=1: &\quad \begin{cases} g_{11}(z) = 0.3508z_1 + 0.3944z_2 = 0.2088, \\ g_{21}(z) = 0.3508z_1 + 0.3944z_2 = 1.2088, \end{cases} \\
j=2: &\quad \begin{cases} g_{12}(z) = -0.1605z_1 + 0.6882z_2 = 0.7138, \\ g_{22}(z) = -0.1605z_1 + 0.6882z_2 = 1.7138, \end{cases} \\
j=3: &\quad \begin{cases} g_{13}(z) = -0.2713z_1 + 0.5636z_2 = 0.4900, \\ g_{23}(z) = -0.2713z_1 + 0.5636z_2 = 1.4900, \end{cases} \\
j=4: &\quad \begin{cases} g_{14}(z) = 0.1477z_1 + 0.1661z_2 = 0.7984, \\ g_{24}(z) = 0.1477z_1 + 0.1661z_2 = 1.7984, \end{cases} \\
j=5: &\quad \begin{cases} g_{15}(z) = 0.8622z_1 + 0.1006z_2 = 0.7038, \\ g_{25}(z) = 0.8622z_1 + 0.1006z_2 = 1.7038, \end{cases} \\
j=6: &\quad \begin{cases} g_{16}(z) = 0.1108z_1 + 0.1245z_2 = -0.2762, \\ g_{26}(z) = 0.1108z_1 + 0.1245z_2 = 0.7238. \end{cases}
\end{aligned}
$$

As illustrated in Fig. 6.2, these 12 hyperplanes partition set C into various cells, each of which corresponds to a unique sign vector defined in (4), $\mathrm{sign}(E) = (w^1, \ldots, w^6)$, where $w^j = (w_{1j}, w_{2j})$. Note that the arrow attached to each hyperplane points to the positive half space generated by this hyperplane. Table 1 lists all the 61 cells identified with their sign vectors and the corresponding distances to $X = \{0, 1, 2\}^6$.

Let us consider Cells 5 and 61 labeled in Fig. 6.2. As any interior point π of Cell 5 satisfies $g_{11}(\pi) > 0$, $g_{21}(\pi) < 0$, $g_{12}(\pi) < 0$, $g_{22}(\pi) < 0$, $g_{13}(\pi) < 0$, $g_{23}(\pi) < 0$, $g_{14}(\pi) < 0$, $g_{24}(\pi) <$

0, $g_{15}(\pi) > 0$, $g_{25}(\pi) < 0$, $g_{16}(\pi) > 0$, and $g_{26}(\pi) < 0$, the sign vector for any interior point π of Cell 5 is thus specified by $((+-)(--)(--)(--)(+-)(+-))$. Furthermore, for all points in Cell 5, the closest integer point in $X = \{0, 1, 2\}^6$ is clearly given by $\varphi^5 = (1, 0, 0, 0, 1, 1)$ and the distance between X and Cell 5 is given by

$$\delta_5 = \text{dist}(X, \text{Cell } 5) = \|(UU^T - I_6)(\varphi^5 - x_0)\| = 0.$$

We observe from Table 1 that there exist two solutions, $(1, 0, 0, 0, 1, 1)^T$ and $(1, 1, 1, 0, 0, 1)^T$, to the fundamental equation of this example, as they both achieve the zero distance. We need to emphasize that our solution scheme enables us to identify all solutions to the fundamental equation on a finite integer set. Note also that the total number of cells in this example is 61, which is much smaller than the upper bound of the cell numbers $O((6 \times 2)^{(6-4)})$.

6.3 Convertion of the Firing Vector to Firing Sequence in Petri Nets

The second step in the reachability algebraic analysis is to translate a firing count vector into a firing sequence, if there is a reachable path. Fig. 6.5 presents a flow diagram of this conversion, which is of a combinatorial nature. Applying this algorithm to two firing vector we got from the previous section, i.e. $(1, 0, 0, 0, 1, 1)^T$ and $(1, 1, 1, 0, 0, 1)^T$, we finally find two feasible firing sequences exist. For $(1, 0, 0, 0, 1, 1)^T$, the firing sequence is $t_5 \rightarrow t_6 \rightarrow t_1$. And for $(1, 1, 1, 0, 0, 1)^T$, the firing sequence is $t_6 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3$. The whole progresses for these two firing sequences are displayed in Fig. 6.3 and Fig. 6.4 respectively.

□ End of chapter.

Table 6.1: A full list of cells, their sign vectors and the closest integer points.

Cell j	$(w^1, w^2, w^3, w^4, w^5, w^6)$	φ^j	δ_j
1	$((+-)(--)(--)(--)(--)(+-))$	$(1,0,0,0,0,1)^T$	0.496
2	$((--)(--)(--)(--)(--)(+-))$	$(0,0,0,0,0,1)^T$	0.532
3	$((+-)(+-)(--)(--)(--)(+-))$	$(1,1,0,0,0,1)^T$	0.780
4	$((+-)(--)(+-)(--)(--)(+-))$	$(1,0,1,0,0,1)^T$	0.708
5*	$((+-)(--)(--)(--)(+-)(+-))$	$(1,0,0,0,1,1)^T$	0.000
6	$((--)(--)(+-)(--)(--)(+-))$	$(0,0,1,0,0,1)^T$	0.890
7	$((--)(--)(--)(--)(+-)(+-))$	$(0,0,0,0,1,1)^T$	0.849
8	$((--)(--)(--)(--)(--)(--))$	$(0,0,0,0,0,0)^T$	0.929
9*	$((+-)(+-)(+-)(--)(--)(+-))$	$(1,1,1,0,0,1)^T$	0.000
10	$((+-)(+-)(--)(--)(+-)(+-))$	$(1,1,0,0,1,1)^T$	0.708
11	$((+-)(--)(--)(--)(++)(+-))$	$(1,0,0,0,2,1)^T$	0.496
12	$((--)(+-)(+-)(--)(--)(+-))$	$(0,1,1,0,0,1)^T$	0.849
13	$((--)(--)(+-)(--)(--)(--))$	$(0,0,1,0,0,0)^T$	1.205
14	$((--)(--)(--)(--)(++)(+-))$	$(0,0,0,0,2,1)^T$	1.285
15	$((--)(--)(--)(--)(+-)(--))$	$(0,0,0,0,1,0)^T$	1.232
16	$((+-)(++)(+-)(--)(--)(+-))$	$(1,2,1,0,0,1)^T$	0.708
17	$((+-)(+-)(++)(--)(--)(+-))$	$(1,1,2,0,0,1)^T$	0.780
18	$((+-)(+-)(+-)(--)(+-)(+-))$	$(1,1,1,0,1,1)^T$	0.496
19	$((++)(+-)(--)(--)(+-)(+-))$	$(2,1,0,0,1,1)^T$	0.890
20	$((+-)(+-)(--)(--)(++)(+-))$	$(1,1,0,0,2,1)^T$	0.941
21	$((++)(--)(--)(--)(++)(+-))$	$(2,0,0,0,2,1)^T$	0.532
22	$((--)(+-)(++)(--)(--)(+-))$	$(0,1,2,0,0,1)^T$	1.259
23	$((--)(+-)(+-)(--)(--)(--))$	$(0,1,1,0,0,0)^T$	1.232
24	$((--)(--)(++)(--)(--)(--))$	$(0,0,2,0,0,0)^T$	1.805
25	$((--)(--)(--)(--)(++)(--))$	$(0,0,0,0,2,0)^T$	1.632
26	$((++)(++)(+-)(--)(--)(+-))$	$(2,2,1,0,0,1)^T$	0.890
27	$((+-)(++)(++)(--)(--)(+-))$	$(1,2,2,0,0,1)^T$	0.496
28	$((+-)(++)(+-)(--)(+-)(+-))$	$(1,2,1,0,1,1)^T$	0.941
29	$((++)(+-)(+-)(--)(+-)(+-))$	$(2,1,1,0,1,1)^T$	0.532
30	$((++)(+-)(--)(--)(++)(+-))$	$(2,1,0,0,2,1)^T$	0.702
31	$((++)(--)(--)(+-)(++)(+-))$	$(2,0,0,1,2,1)^T$	0.843
32	$((--)(++)(++)(--)(--)(+-))$	$(0,2,2,0,0,1)^T$	1.285
33	$((--)(+-)(++)(--)(--)(--))$	$(0,1,2,0,0,0)^T$	1.569
34	$((++)(++)(++)(--)(--)(+-))$	$(2,2,2,0,0,1)^T$	0.532
35	$((++)(++)(+-)(--)(+-)(+-))$	$(2,2,1,0,1,1)^T$	0.702
36	$((++)(+-)(+-)(--)(++)(+-))$	$(2,1,1,0,2,1)^T$	0.582
37	$((++)(+-)(--)(+-)(++)(+-))$	$(2,1,0,1,2,1)^T$	0.860
38	$((++)(--)(--)(+-)(++)(++))$	$(2,0,0,1,2,2)^T$	1.103
39	$((--)(++)(++)(--)(--)(--))$	$(0,2,2,0,0,0)^T$	1.632
40	$((++)(++)(++)(+-)(--)(+-))$	$(2,2,2,1,0,1)^T$	0.843
41	$((++)(++)(++)(--)(+-)(+-))$	$(2,2,2,0,1,1)^T$	0.582
42	$((++)(++)(+-)(+-)(+-)(+-))$	$(2,2,1,1,1,1)^T$	0.860
43	$((++)(++)(+-)(--)(++)(+-))$	$(2,2,1,0,2,1)^T$	0.828
44	$((++)(+-)(+-)(+-)(++)(+-))$	$(2,1,1,1,2,1)^T$	0.692
45	$((++)(+-)(--)(+-)(++)(++))$	$(2,1,0,1,2,2)^T$	1.053
46	$((++)(--)(--)(++)(++)(++))$	$(2,0,0,2,2,2)^T$	1.863
47	$((++)(++)(++)(+-)(+-)(+-))$	$(2,2,2,1,1,1)^T$	0.692
48	$((++)(++)(++)(+-)(--)(++))$	$(2,2,2,1,0,2)^T$	1.103
49	$((++)(++)(+-)(+-)(++)(+-))$	$(2,2,1,1,2,1)^T$	0.803
50	$((++)(+-)(+-)(+-)(++)(++))$	$(2,1,1,1,2,2)^T$	0.876
51	$((++)(++)(--)(+-)(++)(++))$	$(2,2,0,1,2,2)^T$	1.415
52	$((++)(+-)(--)(++)(++)(++))$	$(2,1,0,2,2,2)^T$	1.784
53	$((++)(++)(++)(+-)(++)(+-))$	$(2,2,2,1,2,1)^T$	0.859
54	$((++)(++)(++)(+-)(+-)(++))$	$(2,2,2,1,1,2)^T$	0.876
55	$((++)(++)(++)(++)(--)(++))$	$(2,2,2,2,0,2)^T$	1.863
56	$((++)(++)(+-)(+-)(++)(++))$	$(2,2,1,1,2,2)^T$	0.894
57	$((++)(++)(--)(++)(++)(++))$	$(2,2,0,2,2,2)^T$	1.974
58	$((++)(++)(++)(+-)(++)(++))$	$(2,2,2,1,2,2)^T$	0.901
59	$((++)(++)(++)(++)(+-)(++))$	$(2,2,2,2,1,2)^T$	1.654
60	$((++)(++)(+-)(++)(++)(++))$	$(2,2,1,2,2,2)^T$	1.608
61	$((++)(++)(++)(++)(++)(++))$	$(2,2,2,2,2,2)^T$	1.578

$*$ denotes the cells corresponding to the integer solutions of $Ax = b$.

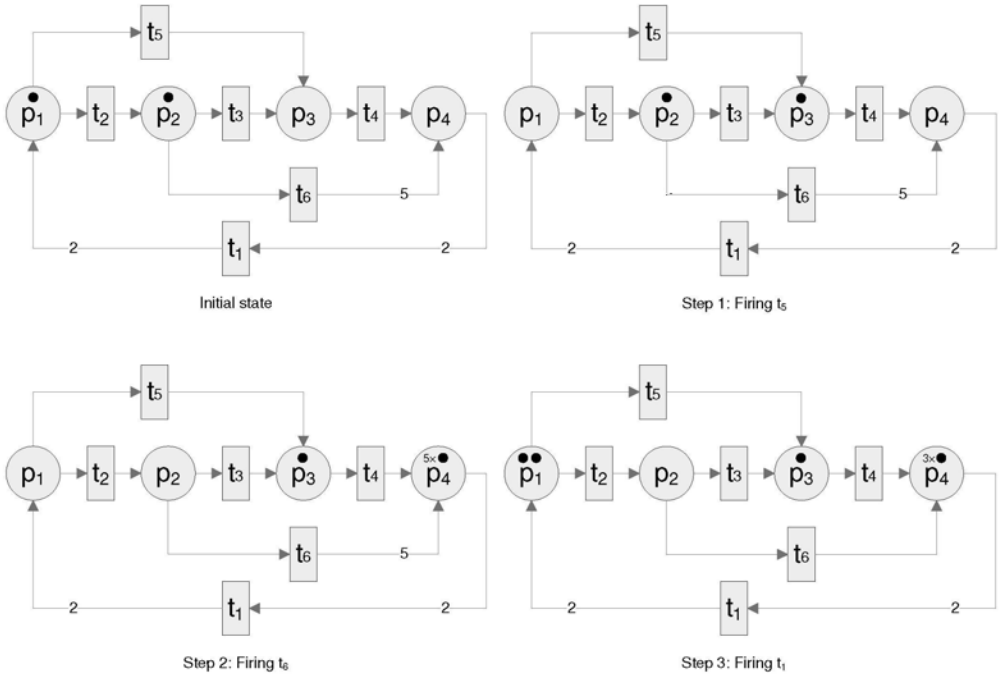


Figure 6.3: Firing sequence with respect to the firing vector $(1, 0, 0, 0, 1, 1)^T$

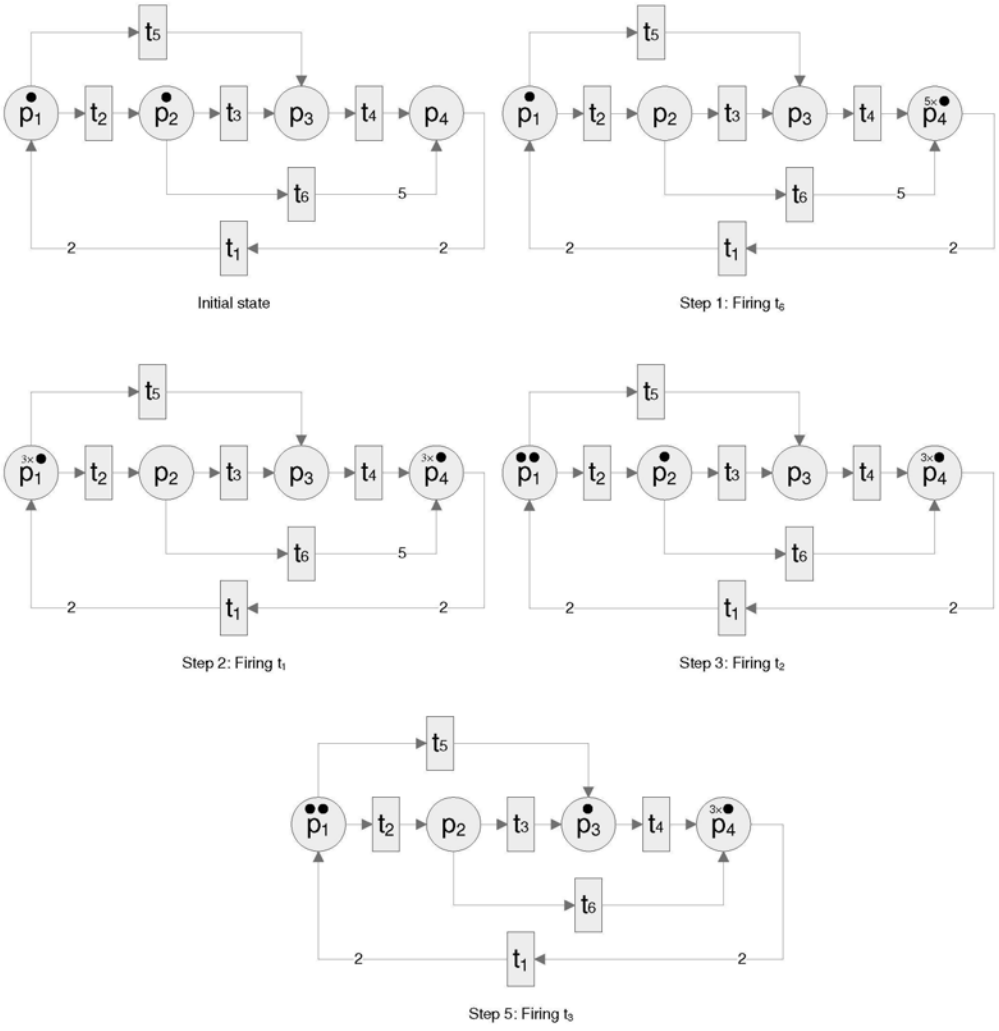


Figure 6.4: Firing sequence with respect to the firing vector $(1, 1, 1, 0, 0, 1)^T$

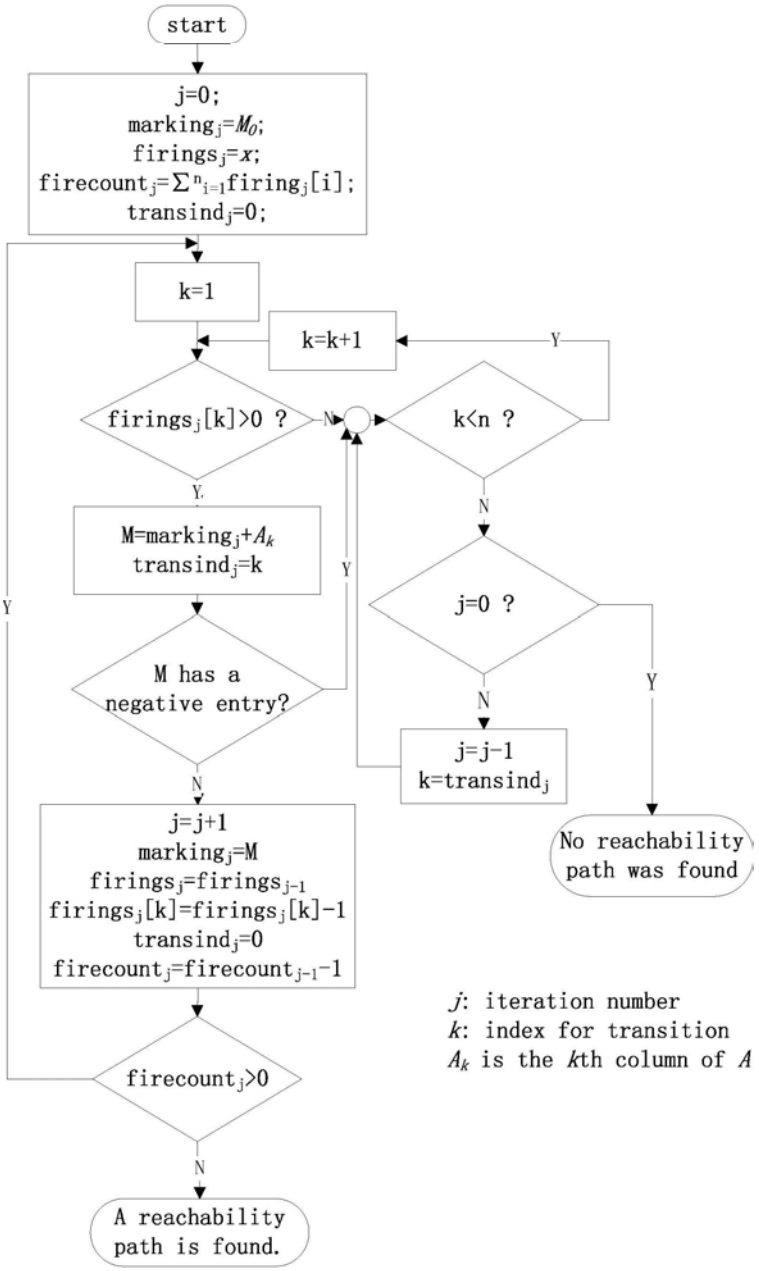


Figure 6.5: Flowchart of converting the firing vector to firing sequence in petri nets

Chapter 7

Conclusion and Future Work

Binary quadratic programming problem is a both classical and long-standing challenging research topic in the realm of operations research. Numerous algorithms have been proposed to solve this problem. These algorithms can be divided into two main categories, i.e. heuristic algorithms and exact solution algorithms. Although heuristic algorithms are more time efficient than exact solution algorithms, the fatal disadvantage that there is no guarantee for them to find the global optimal prevents them from many important applications. In the foreseeable future, the exact solution algorithms will continue to be the dominating force in solving quadratic programming. In Chapter 3 of this thesis, we have proposed a new algorithm by investigating the geometric properties of the original problem. This new algorithm seems to be efficient from our preliminary numerical results. More importantly, with the optimality condition we derived, we can estimate the difficulty in solving a certain problem. In the future, there is still potential to further extend our results along this direction. In Chapter 4, we find out some special cases of the binary quadratic programming problems and develop polynomially solvable algorithms for them so that these special cases can be solved efficiently. By exploring situations where function ϕ is ensured to be linear, more polynomially solvable cases can be identified. Some other polynomially solvable cases of binary quadratic programming problems are reported in Chapter 2. We find that, in these special cases, special matrix structures have close relationships with graph structures. In the future, efforts will be placed to investigate the relationship between the algebraic structure and their graph representation, in order to benefit us to figure out more polynomially solvable cases.

Finding integer solutions to linear equations is another challenging and interesting work in

operations research. In Chapter 5, we have proposed a new algorithm to this problem on the basis of cell enumeration method. Unlike other methods, our approach is applicable to situations with real-valued matrix A. In addition, the complexity of our algorithm can be estimated with the information of n, m and ω. In detail, the complexity of this algorithm is closely related with the difference between the number of variables and the number of equations. This crucial phenomenon will lead us to fight for further improvement.

□ End of chapter.

Bibliography

[1] P. L. Hammer and S. Rudeanu. (1968). *Boolean Methods in Operations Research and Related Areas*, Springer-Verlag, Berlin, Heidelberg, New York.

[2] F. Glover and R. E. D. Woolsey. (1973). Further reduction of zero-one polynomial programs to zero-one linear programming problems. *Oper. Res.*, 21 , pp. 156-161.

[3] F. Glover and R. E. D. Woolsey. (1974). Note on converting the 0-1 polynomial programming problem into a 0-1 linear program. *Oper. Res.*, 22, pp. 180-181.

[4] L. J. Watters. (1967). Reduction of integer polynomial programming problems to zero-one linear programming problems, *Oper. Res.*, 15, pp. 1171-1174.

[5] W. P. Adams and H. D. Sherali. (1986). A tight linearization and an algorithm for zero-one quadratic programming problems, *Manage. Sci.*, 32, pp. 1274-1290.

[6] Jin Y. Yen. (1971). Finding the K Shortest Loopless Paths in a Network, *MANAGEMENT SCIENCE*, Vol. 17, No. 11, pp. 712–716.

[7] L. T. H. An and P. D. Tao. (1998). A branch and bound method via d. c. optimization algorithms and ellipsoidal technique for box constrained nonconvex quadratic problems, *J. Global Optim.*, 13 , pp. 171-206.

[8] B. Kalantari and A. Bagchi. (1990). An algorithm for quadratic zero-one programs, *Naval Res. Logist.*, 37, pp. 527-538.

[9] H. Kinno. (1980). Maximizing a convex quadratic function over a hypercube, J. *Oper. Res. Soc*. Japan, 23, pp. 171-189.

[10] N. V. Thoai. (1998). Global optimization techniques for solving the general quadratic integer programming problem, *Comput. Optim. Appl.*, 10, pp. 149-163.

[11] D. Vanderbussche and G. L. Nemhauser. (2005). GA branch-and-cut algorithm for nonconvex quadratic programs with box constraints, *Math. Program.*, 102, pp. 371-405.

[12] Y. Yajima and T. Fujie. (1998). A polyhedral approach for nonconvex quadratic programming problems with box constraints, *J. Global Optim.*, 13 , pp. 151-170.

[13] A. Billionnet and A. Sutter. (1994). Minimization of a quadratic pseudo-Boolean function, *European J. Oper. Res.*, 78, pp. 106-115.

[14] M. W. Carter. (1984). The indefinite zero-one quadratic problem Discrete *Appl. Math.*, 7, pp. 23-44.

[15] V. P. Gulati, S. K. Gupta, and A. K. Mittal. (1984). Unconstrained quadratic bsivalent program- ming problem, *European J. Oper. Res.*, 15, pp. 121-125.

[16] H. X. Huang and P. M. Pardalos. (2006). Lower bound improvement and forcing rule for quadratic binary programming, *Comput. Optim. Appl.*, 33, pp. 187-208.

[17] E. Beasley. (1998). Heuristic algorithms for the unconstrained binary quadratic programming problem, *tech. report, Imperial College*.

[18] E. Boros, P. L. Hammer, and G. Tavares. (2005). Local search heuristics for unconstrained quadratic binary optimization, *tech. report, RUTCOR, Rutgers University*, Rutcor Research Report.

[19] W. Liu, D. Wilkins, and B. Alidaee. (2005). A hybrid multi-exchange local search for unconstrained binary quadratic program, *tech. report, Hearin Center for Enterprose Science, The University of Mississippi*, Working Paper, HCES-09-05.

[20] Newmann, M. (1972). *Integral Matrices*. Academic Press, New York and London.

[21] Kertzner, S. (1981). The linear diophantine equation. *American Mathematical Monthly*, 88, pp.200–203.

[22] Ramachandran, P. (2006). Use of extended euclidean algorithm in solving a system of linear diophantine equations with bounded variables. *Lecture notes in computer science*, 4076, pp.182.

[23] K. Allemand, K. Fukuda, T.M. Liebling, and E. Steiner. (2001). A polynomial case of unconstrained zero-one quadratic optimization. *Math. Program.*, 91:49–52.

[24] D. Avis and K. Kukuda. (1996). Reverse search for numeration. *Discrete Appl. Math.*, 65:21–46.

[25] F. Barahona. (1986). A solvable case of quadratic 0-1 programming. *Discrete Appl. Math.*, 13:23–26.

[26] F. Barahona, M. Jünger, and G. Reinelt. (1989). Experiments in quadratic 0-1 programming. *Math. Program.*, 44:127–137.

[27] A. Beck and M. Teboulle. (2000). Global optimality conditions for quadratic optimization problems with binary constraints. *SIAM J. Optim.*, 11:179–188.

[28] A. Ben-Tal. (2002). Conic and Robust Optimization. Lecture Notes, Universita di Roma La Sapienzia, Rome, Italy.

[29] A. Billionnet and S. Elloumi. (2007). Using a mixed integer quadratic programming solver for the unconstrained quadratic 0-1 problem. *Math. Program.*, 109:55–68.

[30] P. Chaillou, P. Hansen, and Y. Mahieu. (1986). Best network flow bounds for the quadratic knapsack problem. *Lecture Notes in Mathematics*, 1403:226–235.

[31] S. T. Chakradhar and M.L. Bushnell. (1992). A solvable class of quadratic 0–1 programming. *Discrete Appl. Math.*, 36:233–251.

[32] P. Chardaire and A. Sutter. (1995). A decomposition method for quadratic zero-one programming. *Manage. Sci.*, 41:704–712.

[33] Y. Crama, P. Hansen, and B. Jaumard. (1990). The basic algorithm for pseudo-Boolean programming revisited. *Discrete Appl. Math.*, 29:171–185.

[34] C. Delorme and S. Poljak. (1993). Laplacian eigenvalues and the maximum cut problem. *Math. Program.*, 62:557–574.

[35] J. A. Ferrez, K. Fukuda, and T.M. Liebling. (2005). Solving the fixed rank convex quadratic maximization in binary variables by a parallel zonotope construction algorithm. *European J. Oper. Res.*, 166:35–50.

[36] G. Gallo, M. Grigoridis, and R. E. Tarjan. (1989). A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18:30–55.

[37] M. R. Garey and D. S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. WH Freeman & Co. New York, NY, USA.

[38] M. X. Goemans and D. P. Williamson. (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42:1115–1145.

[39] A. V. Goldberg and R. E. Tarjar. (1986). A new approach to the maximum flow problem. *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pp. 136–146.

[40] P. L. Hammer, P. Hansen, and B. Simeone. (1984). Roof duality, complementation and persistency in quadratic 0-1 optimization. *Math. Program.*, 28:121–155.

[41] P. L. Hammer and S. Rudeanu. (1968). *Boolean Methods in Operations Research and Related Areas*. Springer-Verlag, Berlin, Heidelberg, New York.

[42] P. Hansen, B. Jaumard, and V. Mathon. (1993). Constrained nonlinear 0-1 programming. *ORSA J. Computing*, 5:97–119.

[43] C. Helmberg and F. Rendl. (1998). Solving quadratic (0,1)-problems by semidefinite programs and cutting planes. *Math. Program.*, 82:291–315.

[44] D. Li and X. L. Sun. (2006). *Nonlinear Integer Programming*. Springer, New York.

[45] D. Li, X. L. Sun, and C. L. Liu. (2006). An exact solution method for quadratic 0-1 programming: A geometric approach. Technical report, Chinese University of Hong Kong, Department of Systems Engineering and Engineering Management.

[46] R. D. Mcbride and J. S. Yormark. (1980). An implicit enumeration algorithm for quadratic integer programming. *Manage. Sci.*, 26:282–296.

[47] G. L. Nemhauser and L. A. Wolsey. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, New York.

[48] P. M. Pardalos and G. P. Rodgers. (1990). Computational aspects of a branch-and-bound algorithm for quadratic zero-one programming. *Computing*, 45:131–144.

[49] A. T. Phillips and J. B. Rosen. (1994). A quadratic assignment formulation of the molecular conformation problem. *J. Global Optim.*, 4:229–241.

[50] J. C. Picard and H. D. Ratliff. (1975). Minimum cuts and related problems. *Networks*, 5:357–370.

[51] F. Rendl, G. Rinaldi, and A. Wiegele. (2007). Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. *Lecture Notes Comput. Sci.*, 4513:295–309.

[52] J. Rhys. (1970). A selection problem of shared fixed costs and network flows. *Manage. Sci.*, 17:200–207.

[53] N. Z. Shor. (1987). Quadratic optimization problems. *Sov. J. Comput. Syst. Sci.*, 25:1–11.

[54] N. Sleumer. (1999). Output-Sensitive Cell Enumeration in Hyperplane Arrangements. *Nordic Journal of Computing*, 6:137–161.

[55] X. L. Sun, C. L. Liu, D. Li, and J. J. Gao. (2007). On duality gap in binary quadratic optimization. Technical report, Chinese University of Hong Kong, Department of Systems Engineering and Engineering Management.

[56] T. Zaslavsky. (1975). Facing up to arrangements: face-count formulas for partitions of space by hyperplanes. *Mem. Amer. Math. Soc*, 1:1–101.

[57] Aardal, K., Hurkens, C.A.J., and Lenstra, A.K. (2000). Solving a system of linear diophantine equations with lower and upper bounds on the variables. *Mathematics of Operations Research*, 25, 427–442.

[58] Edelsbrunner, H. (1987). *Algorithms in Combinatorial Geometry*. Springer.

[59] Kertzner, S. (1981). The linear diophantine equation. *American Mathematical Monthly*, 88, 200–203.

[60] Kostin, A.E. (2003). Reachability analysis in t-invariant-less petri nets. *IEEE Transactions on Automatic Control*, 48, 1019–1024.

[61] Matsumoto, T., Miyano, Y., and Jiang, Y. (1997). Some useful sufficient criteria for the basic reachability problem in general petri nets. In *Proceedings of the 36th IEEE Conference on Decision and Control*, volume 4, 4104–4109. San Diego, CA, USA.

[62] Matsumoto, T., Takata, M., and Moro, S. (2002). Reachability analyses in petri nets by groebner bases. In *Proceeding of SICE 2002*, 841–846.

[63] Murata, T. (1977). State equation, controllability, and maximal matchings of petri nets. *IEEE Transactions on Automatic Control*, 22, 412–416.

[64] Petri, C.A. (1962). *Kommunikation mit Automaten*. Ph.D. thesis, Schriften des Rheinisch-Westfalischen Instituts fur Instrumentelle Mathematik, Bonn, Germany.

[65] Ramachandran, P. (2006). Use of extended euclidean algorithm in solving a system of linear diophantine equations with bounded variables. *Lecture Notes in Computer Science*, 4076, 182–192.

[66] Schrijver, A. (1986). *Theory of Linear and Integer Programming*. John Wiley & Sons, New York.

[67] Sun, X.L., Liu, C.L., Li, D., and Gao, J.J. (2008). On duality gap in binary quadratic optimization. In *Working paper series*. Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong.

[68] Yen, H.C. (2006). Introduction to petri net theory. *In Recent Advances in Formal Languages and Applications, Studies in Computational Intelligence*, 25, 343–373.

[69] Zurawski, R. and Zhou, M.C. (1994). Petri nets and industrial applications: A tutorial. *IEEE Transactions on Industrial Electronics*, 41, 567–583.