The P-norm Surrogate-constraint Algorithm for Polynomial Zero-one Programming

A THESIS

SUBMITTED TO THE DEPARTMENT OF SYSTEMS ENGINEERING

AND ENGINEERING MANAGEMENT OF

THE CHINESE UNIVERSITY OF HONG KONG

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

By

WANG JUN

August, 1999

Acknowledgment

I would like to express my greatest gratitude to my supervisor, Professor Duan Li, for his excellent guidance in the development of the ideas in this thesis, as well as in writing. His high standard has much enhanced the quality of this work. I would also like to thank Professor Xun-yu Zhou, Professor C. H. Cheung and Professor Sydney C. K. Chu for their expert serving as members of my defense committee.

Moreover, I would like to thank Dr. Xiao-lin Sun and Ms. Cui-zhen Yao for their professional suggestions in the course of this work.

I would like to also take this opportunity to thank all computing staff of the Department of Systems Engineering and Engineering Management of The Chinese University of Hong Kong for their assistance and maintenance of the computing facilities. I also thank all other staff in the department for their support.

Finally, special thanks must be devoted to my wife, Ping Zhu, and my daughter, for their continuing support, endless love and constant encouragement.

摘要

许多现实生活中的决策问题可以模型化为一个多项式0-1规划．借助二进制变量的一些特性，多项式0-1规划问题能够转换成为一个等价的线性主问题与多项式非线性第二约束．我们注意到多项式0-1规划是NP-HARD问题．根据多项式0-1规划问题的可行集是它主问题可行集的子集，TAHA [1972]提出了一个著名算法，即从满足第二约束的主问题可行解里寻找最优解．本论文主要的研究工作是基于TAHA的算法和李端[1999]提出的P次范数替代约束方法，进而发展一个高效数值算法来解决多项式0-1规划问题．

运用P次范数替代约束方法，多项式0-1规划中的多个约束可以被单一的替代约束来代替．当P充分大时，替代松弛的可行集和原问题可行集精确匹配．由于0-1变量的幂仍是它本身，单一替代约束的复杂程度和原问题在同一水平．将一个多约束多项式0-1规划问题简化成一个单一替代约束问题会极大便利主问题可行解的确定．新算法利用这个突出的性质在搜寻过程中运用有效的"探寻"和"折返"策略．实例验证了P次范数替代约束算法在多项式0-1规划中的高效率．它在集合覆盖问题上的应用也在论文中被研究．

Abstract

Many real-world decision making problems can be modeled by a polynomial zero-one programming formulation. By some special properties of binary variables, a polynomial zero-one programming problem can be converted into an equivalent linear zero-one programming problem, called the *master problem*, with nonlinear *secondary constraints* representing the polynomial terms. Since the polynomial zero-one programming problem is NP-hard in the strong sense, several numerical solution algorithms have been suggested in the literature in solving it. Observing the fact that the feasible set of the polynomial zero-one programming problem is a subset of its master problem, Taha [1972], proposed a well-known algorithm for polynomial zero-one programming in which the optimal solution is sought from among the set of the feasible solutions to the master problem while it satisfies the secondary constraints. The major research task in this thesis is to develop a more efficient numerical solution algorithm for polynomial zero-one programming based on both Taha's previous work and the p-norm surrogate constraint method recently proposed by Li [1999].

Adopting the p-norm surrogate constraint method, a single surrogate constraint can be constructed for polynomial zero-one programming problems with multiple constraints such that the feasible sets in a surrogate relaxation and the original problem match exactly. Since a power of a zero-one variable is itself, the complexity degree in the single surrogate constraint is at the same level as in the original problem. Reducing a polynomial zero-one programming problem with multiple constraints into an equivalent one with a single surrogate constraint greatly facilitates the identification of the feasible solutions in the master prob-

lem. The new numerical solution scheme proposed in this thesis has been devised to take advantage of this prominent feature in carrying out the "fathoming" procedure and the "backtrack" procedure in a searching process of an implicit enumeration. Numerical testing has demonstrated the efficiency of the proposed p-norm surrogate-constraint algorithm in polynomial zero-one programming. Its application in the set covering problem has been also studied.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

The literature has clearly demonstrated the importance and wide applications of the linear zero-one programming. "However, it is often the case that a polynomial (nonlinear) zero-one model more accurately reflects the real-world by allowing for the interaction that frequently occurs between the decision variables" [30]. Many real-world problems, such as scheduling, facility allocation, and capital budgeting [11][25][29][28][27][33][35], have been modeled by a polynomial zero-one formulation. Unfortunately, the polynomial zero-one programming problem is NP-hard in the strong sense, i.e., no algorithm seems possible to find an exact optimal solution in polynomial time. So what we can do is to develop more efficient algorithms, under this limitation, to solve polynomial zero-one programming problems.

The majority of the algorithms for zero-one programming in the literature is devised to solve linear zero-one programming problems in which the objective function and the constraints are all linear. Until recently, many of them have been modified to fit the need to solve polynomial zero-one programming problems which can be converted into linear zero-one programming problems with their polynomial constraint systems by using some special properties of binary variables. The fact is that each term, a cross product of several variables (maybe to a high power), in a polynomial zero-one programming formulation is still a binary variable. The Balasian-based algorithm for polynomial zero-one programming proposed by Taha [31][32] in 1972 is one of the most typical and successful algorithms, where the additive algorithm for solving linear zero-one programming problem [1] was extended directly.

1.2 The polynomial zero-one programming problem

We consider in this thesis the following polynomial zero-one programming problem:

$$\min \sum_{j=1}^{2^{n'}-1} c_j' \prod_{k_j \in K_j} y_{k_j} \tag{1.1}$$

$$\text{s.t.} \quad g_i(y) = \sum_{j=1}^{2^{n'}-1} a_{ij}' \prod_{k_j \in K_j} y_{k_j} \leq b_i', i = 1, 2, \ldots, m.$$

where $y = [y_1, y_2, \ldots, y_{n'}] \in \{0,1\}^{n'}$ is the vector of *decision variable*, $K_j \subseteq N' = \{1, 2, \ldots, n'\}$, $2^{n'} - 1$ is the total possible number of K_j, and all a_{ij}', $i \in \{1, 2, \ldots, m\}$ and $j \in \{1, 2, \ldots, 2^{n'} - 1\}$, are assumed to be integers. Without loss of generality, $g_i(y)$, i = 1,2, ..., m, are assumed to be strictly positive for all

$y \in \{0,1\}^{n'}$. Problem (1.1) is referred as the *general form* in polynomial zero-one programming.

Let $n = 2^{n'} - 1$, $N = \{1, 2, \ldots, n\}$. Define

$$x_j = \begin{cases} \prod_{k_j \in K_j} y_{k_j}, & j \in J^+ = \{j | c'_j \geq 0, \ j \in N\}, \\ 1 - \prod_{k_j \in K_j} y_{k_j}, & j \in J^- = \{j | c'_j < 0, \ j \in N\}. \end{cases} \tag{1.2}$$

We call $x = [x_1, x_2, \ldots, x_n] \in \{0,1\}^n$ the vector of *decision term.* If we let

$$c_j = \begin{cases} c'_j, & j \in J^+, \\ -c'_j, & j \in J^-, \end{cases} \tag{1.3}$$

the general form (1.1) can then be expressed by the following form:

$$\min \quad z = \sum_{j=1}^{n} c_j x_j, \quad c_j \geq 0, \tag{1.4}$$

$$s.t. \quad \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i, \text{ for } i = 1, 2, \ldots, m,$$

where s_i, $i \in \{1, 2, \ldots, m\}$, are nonnegative slack variables, a_{ij} and b_i in (1.4) are deduced from (1.1) and (1.2), as well as the vector of decision term x satisfies (1.2). Until recently, the polynomial zero-one programming problem (1.1) has been transformed into an equivalent two-level problem, a *master problem* (1.4) with (1.2) as its *secondary constraints.* Clearly, master problem (1.4) is a linear zero-one programming problem while the second constraint (1.2) is a polynomial system.

1.3 Motivation

Taha once predicted in [31] that the efficiency of his algorithm may be further enhanced by taking the advantage of more efficient "fathoming" techniques than the additive algorithm [1].

The p-norm surrogate constraint method has been recently proposed in [22] for integer programming. Using the p-norm surrogate constraint method, a polynomial zero-one programming problem with multiple constraints can be converted into an equivalent one with a single surrogate constraint. Since a power of a zero-one variable is itself, the complexity degree in the single surrogate constraint is at the same level as in the original problem. The feature of a single constraint must greatly facilitate the identification of the feasible solutions to the master problem. So, it becomes possible to improve the efficiency of the Balasian-based algorithm by modifying both the "fathoming" procedure and the "backtrack" procedure.

Based on these considerations, we have devised a new solution scheme in this thesis to take the advantage of this prominent feature in carrying out both the "fathoming" procedure and "backtrack" procedure in a searching process of an implicit enumeration.

1.4 Thesis outline

The new algorithm, p-norm surrogate-constraints algorithm for polynomial zero-one programming, is mainly based on both the strengths of the Balasian-based algorithm for polynomial zero-one programming [31][32] and the contributions of the p-norm surrogate constrain method [22]. So, these two algorithms are first described briefly in Chapter 3 and Chapter 4 as preliminary. Chapter 5 is the most important chapter in this thesis, in which the new algorithm is presented in detail. Chapter 6 shows us how to solve two examples step by step in the new algorithm and Taha's algorithm respectively. From computation experiences, some comparisons are also made between them in that chapter. An application of this new algorithm to the set-covering problem is introduced in Chapter 7. Finally, Chapter 8 summarizes the thesis and gives suggestions for further study.

Chapter 2

Literature Survey

Depending on whether or not the problem (1.1) can be solved directly, the solution algorithms for polynomial zero-one programming reported in the literature can be classified into two groups.

The first group, including Lawler and Bell's algorithm [21] and the covering relaxation algorithm [19][20], directly solves the problem (1.1) without any transformation. The second group includes the following algorithms:

(i) The method of reducing polynomial integer problems to linear zero-one problems [36],

(ii) Pseudo-Boolean programming [13],

(iii) The Balasian-based algorithm for zero-one polynomial programming [31] [32], and

(iv) Hybrid algorithm for zero-one polynomial programming [30].

The common character of these four algorithms is that they first reduce the problem (1.1) to a master problem (linear) with its secondary constraints (polynomial) before tackling it.

In the following sections, we introduce these six algorithms concerned above in six subsections.

2.1 Lawler and Bell's method

In 1966, Lawler and Bell [21] developed a method for polynomial zero-one programming that is of a nature of implicit enumeration. Since this method directly solves the general form of the polynomial zero-one programming problem (1.1), it belongs to the first group.

Lawler and Bell's method first converts the general form of the polynomial zero-one programming problem (1.1) into a standard form with a *monotone nonincreasing* objective function subject to the constraints constructed as the differences between two monotone noninceasing parts, then initializes the solution vector $X_0 = (0, 0, \cdots, 0)$, sets an infinite upper bound as well. Based on the fact that the objective function and the constraints are monotone nonincreasing, three rules are designed to check whether the solution vector X_s in the sth iteration is a potential candidate for the optimal solution or not by the means of the *numerical ordering.* If a solution vector X_s satisfies the conditions of the rule 1 or the rule 3, some solution vectors in the numerical ordering can be skipped and the algorithm goes to a more promising solution while assuring no optimal solutions will be by-passed. If X_s satisfies the conditions of the rule 2, i.e., if it is both feasible and superior than the previous solutions, it can be substituted for the current optimal solution, and the upper bound is updated by the value of the corresponding objective function associated with X_s. The procedure terminates when the numerical ordering of the solution vector is overflown.

In the polynomial zero-one programming problem, because the variables in a solution vector are assigned at 0 or 1 in a fixed order, the ability of excluding hopeless solutions and the flexibility in searching the optimal solution become weak. On the other hand, the nature of a fixed order simplifies the computer programming and saves a great amount of storage.

2.2 The covering relaxation algorithm for polynomial zero-one programming

The algorithm in [19] [20] is a cutting plane algorithm, i.e., it is not a branch and bound or implicit enumeration algorithm. It especially fits to solve the polynomial zero-one problem with linear objective function subject to polynomial constraints as follows:

$$\begin{aligned} \max \quad & z = \sum_{j=1}^{n} c_j y_j \\ \text{s.t.} \quad & \sum_{j=1}^{2^n-1} a_{ij} \prod_{k_j \in K_j} y_{k_j} \leq b_i, \ \text{for } i = 1, 2, \ldots, m, \end{aligned} \tag{2.1}$$

where $c_j \geq 0$, $b_i \geq 0$, and $a_{ij} \geq a_{ij+1} \geq 0$. Associated with each constraint violated by a given solution, an *ordinary cut* is generated as follows:

$$\sum_{j \in S} y_j \leq |S| - 1, \tag{2.2}$$

where $S = \cup_{j=1}^{l} K_j$ for l is the smallest index such that $\sum_{j=1}^{l} a_{ij} > b_i$.

From the concept of the ordinary cut stated above, the authors devised a covering relaxation algorithm dealing with the problem (2.1). The algorithm starts with solving the initial covering relaxation problem, the problem (2.1) without any constraint, to obtain a candidate for optimal solutions using four different greedy heuristic algorithms [18] [34]. If the candidate is feasible to the problem (2.1), it will be the optimal solution thereof and the solution process terminates. Otherwise, all ordinary cuts for violated constraints are constructed as (2.2), and a new covering relaxation problem is generated with these ordinary cuts added to the old covering relaxation problem as constraints. The new covering relaxation problem can be dealt with similarly to the initial covering relaxation problem. The authors have proved that, after at most 2^n iterations, the procedure will terminate with an optimal solution or a certificate of no feasible solution existing.

A promising feature of this algorithm is that no additional variable is introduced in the solution procedure. In return, a nested sequence of linear covering relaxation problems have to be solved. As the covering relaxation algorithm has been derived primarily for polynomial zero-one programming problems with linear objective functions, its efficiency of solving polynomial zero-one programming problems with nonlinear objective function is expected to be very low.

2.3 The method of reducing polynomial integer problems to linear zero-one problems

With the improvements suggested in [16] [17], Watters [36] proposed a method to solve the polynomial zero-one programming problem. He designed an appropriate technique, making full use of the properties of binary variables, to linearize the secondary constraints (1.2) such that the polynomial zero-one programming problem can be equivalently transformed into a complete linear zero-one problem.

Based on the relationships among the values of x_j and y_{k_j} in the secondary constraints (1.2), $x_j = \prod_{k_j \in K_j} y_{k_j}$ can be equivalently replaced by the following linear constraints:

$$\begin{aligned} \sum_{k_j \in K_j} y_{k_j} - x_j &\leq q_j - 1, \\ -\sum_{k_j \in K_j} y_{k_j} + q_j x_j &\leq 0, \\ x_j, y_{k_j} &= 0 \text{ or } 1, \end{aligned} \tag{2.3}$$

where q_j is the number of the elements in K_j. The secondary constraint (1.2) can be thus enforced by the following linear constraints:

$$\prod_{k_j \in K_j} y_{k_j} - (q_j - 1) \leq \left\{ \begin{array}{c} x_j \\ \\ 1 - x_j \end{array} \right\} \leq \frac{1}{q_j} \sum_{k_j \in K_j} y_{k_j} \left\{ \begin{array}{l} j \in J^+, \\ \\ j \in J^-. \end{array} \right. \tag{2.4}$$

The polynomial secondary constraints are therefore linearized by the inequalities (2.4). The problem (1.1) is equivalently converted into a linear master problem (1.4) with linear constraints (2.4) and can be solved in Balasian algorithm or other methods for linear zero-one programming.

The limitation of this algorithm rests on that the number of additional variables and the number of the inequalities (2.4), generated in the linearization to the secondary constraints, may often become quite large so that the transformed problem becomes intractable in practice. The primary reason for this dimensionality problem is that the constraints (2.4) and the variables y_{k_j} are considered explicitly at the same time in the solution process. In other words, the size of the problem will be dictated by both the dimensions of the master problem and the secondary constraints.

2.4 Pseudo-boolean programming

Hammer and Rudeanu [13] proposed an algorithm, termed Pseudo-Boolean programming, for polynomial zero-one programming.

For each constraint in the master problem (1.4), Pseudo-Boolean programming starts by determining its *basic solutions*, and further finds the *families of feasible solutions*. The *characteristic function* of the master problem is denoted by φ, and

$$\varphi = \varphi_1\varphi_2\cdots\varphi_m, \tag{2.5}$$

where each φ_i, $i \in \{1, 2, \ldots, m\}$, is the characteristic function of the ith constraint generated from the corresponding families of feasible solutions. φ now is a function of decision terms x_j. The characteristic function of the original problem, denoted by ψ, is derived from φ with x_j replaced by y_i according to the secondary constraints (1.2). After simplification, ψ can be always expressed by

$$\psi = \psi_1 \cup \psi_2 \cup \psi_3 \cup \ldots. \tag{2.6}$$

Conversely, all the families of feasible solutions for the problem (1.1) can be derived from ψ_1, ψ_2, ..., among which the optimal solution can be sought by comparing the objective function values.

This algorithm is not efficient in the sense that the feasibility check is not integrated with the optimality check, Because the objective function only plays a passive role in checking whether a feasible solution is optimal or not.

2.5 The Balasian-based algorithm for polynomial zero-one programming

Based on a result in Hammer-Rudeanu [13], Taha developed an algorithm [31] [32] for polynomial zero-one programming by modifying Balas' additive algorithm [1]. After converting the original problem into the master problem with the secondary constraints, it becomes clear that the optimal solution to the original polynomial problem must be a feasible solution to the master problem. Taha's algorithm starts with searching the feasible solutions to the linear master problem implicitly by a modified Balas's algorithm and then determines whether the current solution is better than the previous ones while satisfying the secondary constraints. In finite iterations, either an optimal solution is obtained or no feasible solution exists.

This algorithm is efficient in determining whether a feasible solution to the master problem is optimal to the original problem, but the process in searching for all the feasible solutions could be very time consuming.

2.6 The hybrid algorithm for polynomial zero-one programming

In 1990, having absorbed solution concepts from both the Balasian-based algorithm [31] [32] and pseudo-boolean programming [13], Snyder and Chrissis proposed a hybrid algorithm [30]. This algorithm is still an implicit enumeration algorithm while possessing two new solution strategies different from its predecessors in [31] [32] and [13]. The first is the *length-one minimum cover method*, and the second is the *term ranking strategy*.

The procedure in [30] to obtain the optimal solution(s) is composed of a series of iterations. At each iteration, the algorithm generates a *partial solution* which fixes a subset of the decision variables at either zero or one. Simultaneously, the algorithm fathoms these partial solutions according to the three rules given by Chrissis [10]. In addition, the authors develop the length-one minimum cover method to incorporate with the fathoming procedure.

Consider a modified version of the ith constraint in the master problem (1.4),

$$\sum_{j=1}^{n} a_{ij}x_j \leq b_i, \tag{2.7}$$

where all the coefficients $a_{ij}$are assumed to be strictly positive. If $a_{ij} > b_i$, the index j is called a length-one minimum cover to the ith constraint. That is to say, if j is a length-one minimum cover, inequality (2.7) is only true when $x_j = 0$. If the ith constraint has no length-one minimum cover, we temporarily remove it from the master problem. If j is a length-one minimum cover to at least one constraint, the term x_j should be fixed at the level of 0 immediately; If a fixation is found to be inconsistent with the previous ones, this problem has no feasible solution. The procedure repeats until either no minimum cover remains or it can be concluded that no feasible solution exists.

The computational experience in [30] has shown that the term ranking strategy, restructuring the polynomial zero-one programming problem according to a descending order of the costs, can significantly reduce the computational time. It seems that the hybrid algorithm is efficient for polynomial zero-one programming, especially in solving large-scale problems.

Chapter 3

The Balasian-based Algorithm

As presented in Chapter 1, a polynomial zero-one programming problem can be converted into an equivalent mater problem (linear) (1.4) with its secondary constraints (nonlinear) (1.2) representing the polynomial terms. Observing the fact that the feasible set of the problem (1.1) is a subset of the master problem (1.4), Taha [31][32] proposed a well-known algorithm, the Balasian-based algorithm, for polynomial zero-one programming in which the optimal solution is sought from among the set of the feasible solutions to the master problem while satisfying the secondary constraints. Adopting the modified additive algorithm [1] for linear zero-one programming, Taha's algorithm starts by finding out all the feasible solutions to the master problem, checks whether they are consistent to the secondary constraints and finally, chooses the optimal solution among both feasible and consistent solutions.

To understand the Balasian-based algorithm, the additive algorithm for linear zero-one programming is first sketched at the beginning of this chapter.

3.1 The additive algorithm for linear zero-one programming

In 1965, Balas [1] proposed an implicit enumeration method or branch and bound method to solve linear zero-one programming problems directly. Since only additions and subtractions are used in the solution procedure, this method is named as the additive algorithm. Although Balas did not give enough evidence to prove its efficiency in his paper [1], many algorithms proposed later, including Taha's algorithm, were developed based on Balas' work.

Consider linear zero-one programming problems of the following form,

$$\begin{aligned} \min \quad & z = \sum_{j=1}^{n} c_j x_j, \ c_j \geq 0, \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_{ij} x_j + s_i = b_i, \ i \in \{1, 2, \cdots, m\}, \end{aligned} \tag{3.1}$$

where $x_j \in \{0, 1\}$ for all $j \in N = \{1, 2, \ldots, n\}$ are decision variables, and s_i, $i = 1, 2, \ldots, m$, are nonnegative slack variables. $[x_1, x_2, \ldots, x_n, s_1, s_2, \ldots, s_m]$ is called *a solution vector*, and is denoted by U.

The algorithm starts with the solution vector $U^0 = [x_1^0, x_2^0, \ldots, x_n^0, s_1^0, s_2^0, \ldots, s_m^0]$ $= [0, 0, \cdots, 0, b_1, b_2, \cdots, b_m]$. Obviously, it is a dual-feasible solution to the linear programming problem corresponding to (3.1) since all $c_j \geq 0$. If all b_i, $i \in \{1, 2, \ldots, m\}$, are nonnegative, U^0 is the optimal solution to the problem (3.1); Otherwise, set $J_0 = \emptyset$ and introduce an index $j \in N$, according to a certain rule, into J_0.

At iteration t, the solution vector U^t, given by:

$$x_j^t = \begin{cases} 1 & (j \in J_t), \\ 0 & (j \in N - J_t), \end{cases} \qquad s_i^t = b_i - \sum_{j \in J_t} a_{ij}. \tag{3.2}$$

is still a dual-feasible solution to the corresponding relaxation of (3.1). If there exists $i \in \{1, 2, \ldots, m\}$ such that $s_i^t < 0$, form *the improving set for the solution vector* U^t, N_t, defined as follows:

$$N_t = N - (C^t \cup D^t \cup E^t), \tag{3.3}$$

where C^t stands for the set of those j introduced into J_k such that $k \leq t$ and $J_k \subset J_t$ before the solution U^t is obtained; D^t is the set of those $j \in (N - C^t)$ such that, if j is introduced into J_t, the value of the objective function would hit the ceiling for U^t; E^t is the set of those $j \in [N - (C^t \cup D^t)]$ such that, if j is introduced into J_t, no negative s_i^t would be increased in value. Thus, we can introduce an index $j \in N_t$, according to a certain rule, into J_t to improve the solution vector U^t and a new iteration starts.

If all slack variables s_i^t for $i \in \{1, 2, \ldots, m\}$ are nonnegative, U^t is a feasible solution to the problem (3.1). Let z_t denote the value of the objective function corresponding to U^t. When z_t is less than the current optimal value, $z_{\min}$ and $U_{\min}$ are replaced by z_t and U^t, respectively. The solution procedure then checks *the improving sets for the solution vector* U^k*, left after iteration* t, N_k^t, such that $k < s$ and $J_s \subset J_t$ in the decreasing order of the number k. N_k^t is defined as follows:

$$N_k^t = N_k - (C_k^t \cup D_k^t), \tag{3.4}$$

where C_k^t stands for the set of those j introduced into J_k before the solution U^t is obtained; D_k^t is the set of those $j \in (N_k - C_k^t)$ such that, if j is introduced into

J_k, the value of the objective function would hit the ceiling for U^t. If $N_k^t = \emptyset$ for all k such that $k < t$ and $J_k \subset J_t$, neglect this branch and a new iteration starts. Otherwise, we can introduce an index $j \in N_k^t$, according to a certain rule, into J_k to improve the solution vector U^t and a new iteration starts.

After finite iterations, either the optimal solution vector $U_{\min}$ is obtained or no feasible solution exists. The most prominent feature of this algorithm is that its operations only include additions and subtractions in the solution procedure, so computational round-off errors are totally avoided.

For satisfying Taha's algorithm to find all feasible solutions of the master problem, Taha modified this algorithm by fixing the upper bound as infinite, i.e., remove the set D^t and D_k^t form (3.3) and (3.4), respectively.

3.2 Some notations and definitions referred to the Balasian-based algorithm

Instead of the general form (1.1), the Balasian-based algorithm only considers the master problem (1.4) with its secondary constraints (1.3) in the solution process of the polynomial zero-one programming problem.

Partial solution A partial solution J_t is defined as a permutation of a subset of $\{\pm j | i \in N\}$, where $+j \in J_t$ implies that $x_j = 1$ and $-j \in J_t$ implies that $x_j = 0$ in the tth iteration. So J_t assigns binary values to a part of x_j for $j \in N$.

Completion and free term A completion of J_t is any vector of decision term x whose components are partially determined by J_t while the rest, called free terms, are chosen arbitrarily between 0 and 1.

Feasible and Consistent If the completion of J_t with all corresponding free terms set at zero constitutes a/an feasible/infeasible solution to the master problem (1.4), J_t is called feasible/infeasible. If there exists a completion of J_t satisfying the secondary constraints (1.2), J_t is called consistent; Otherwise, J_t is called inconsistent.

Fathoming Fathoming in [31] is a process that checks whether the branch represented by J_t is needed to be considered further. If (i) a given J_t is infeasible and J_t has no feasible completion, or (ii) a given J_t is feasible and any augmentation to J_t by one or more free terms set at one will invite an infeasibility, J_t is fathomed and the corresponding branch will be removed.

3.3 Identification of all the feasible solutions to the master problem

In the process of searching for all the feasible solutions to the master problem, the fathoming procedure is applied on-line.

When a given partial solution, J_t, is infeasible, the modified additive algorithm [1] is used to find a new feasible partial solution J_{t+1} by augmenting J_t with a subset of $\{+j|j \in N - J_t\}$ on the right, i.e., fixing some free terms at 1. If no feasible partial solution exists, J_t has no feasible completion and J_t is fathomed in case (i).

When J_t is feasible, a set is defined as follow:

$$Q_t = \{j \in (N - J_t) \mid a_{ij} \leq s_i^t \text{ for all } i \in N\}, \tag{3.5}$$

where s_i^t is the ith slack variable at the tth iteration. If $Q_t \neq \emptyset$, a new feasible partial solution J_{t+1} can be achieved by augmenting J_t with $\{+k | c_k = \min_{j \in Q_t} c_j\}$. If $Q_t = \emptyset$, J_t is augmented by $\{+k | w_k^t = \max_{j \in N - J_t} \{w_j^t | w_j^t = \sum_{i=1}^{m} \min(0, S_i^t - a_{ij})\}\}$, resulting in an infeasible partial solution, and the modified additive algorithm is performed again to find a new feasible partial solution J_{t+1}. In case that J_{t+1} doesn't exist, any augmentation to J_t by one or more free terms set at one will invite infeasibility and J_t is fathomed in case (ii).

A fathomed partial solution J_t indicates that its remaining completions are entirely infeasible. A "backtrack" procedure [15], changing the rightmost positive element of J_t into a negative one and then deleting all the elements to its right (if any), is carried out to abandon this useless branch and generates a nonredundant one. When no element left in a partial solution J_t is positive, all 2^n possible solutions to the master problem have been implicitly checked so that the feasible solutions to the master problem have been founded out completely. Thus, the fathoming process terminates.

3.4 Consistency check of the feasible partial solutions

When a feasible partial solution, J_t, is achieved, its consistency should be checked according to the secondary constraints.

If J_t is inconsistent, any augmentation to J_t also leads to inconsistency. The "backtrack" procedure is performed to generate a new partial solution which will be checked by the next round of the fathoming procedure.

If J_t is consistent, some of the decision variables y_k for $k \in N'$ are fixed at either 0 or 1 by both J_t and the secondary constraints. Conversely, these fixed variables can determine a set $B_t = \{+j | x_j \text{ is fixed at } 1, j \in N - J_t\}$. $B_t = \emptyset$ means J_t with all free terms set at 0 is a feasible solution to the original problem, and the current optimal solution is updated if the objective function value corresponding to J_t is better than the current optimal value. The "backtrack" procedure will be performed on J_t. When $B_t \neq \emptyset$, $J_t \cup B_t$ is still a consistent partial solution, and its feasibility will be checked again.

In finite iterations, either an optimal solution is obtained or it can be concluded that no feasible solution exists.

Chapter 4

The p-norm Surrogate Constraint Method

4.1 Introduction

The p-norm surrogate constraint method has been recently proposed by Li [22] for integer programming. Using this method, a polynomial zero-one programming problem with multiple constraints can be always converted into an equivalent polynomial zero-one programming problem with a single surrogate constraint if a positive integer p is selected to be large enough. One of the prominent properties of this method that is different from other surrogate constraint methods is that no assumption of convexity is required.

For a positive integer p, the p-norm surrogate constraint formulation of the problem (1.1) is given as

$$\begin{aligned} \min \quad & \sum_{j=1}^{2^{n'}-1} c_j' \prod_{k_j \in K_j} y_{k_j}, \\ \text{s.t.} \quad & \sum_{i=1}^{m} [\mu_i g_i(y)]^p \leq \sum_{i=1}^{m} [\mu_i b_i']^p, \end{aligned} \tag{4.1}$$

or

$$\begin{aligned} \min \quad & \sum_{j=1}^{2^{n'}-1} c_j' \prod_{k_j \in K_j} y_{k_j}, \\ \text{s.t.} \quad & \sum_{i=1}^{m} [\mu_i \sum_{j=1}^{2^{n'}-1} (a_{ij}' \prod_{k_j \in K_j} y_{k_j})]^p \leq \sum_{i=1}^{m} [\mu_i b_i']^p, \end{aligned} \tag{4.2}$$

where all $\mu_i > 0$, $i = 1, 2, \ldots, m$, and satisfy the following two constraints:

$$\mu_1 b_1' = \mu_2 b_2' = \ldots = \mu_m b_m', \tag{4.3}$$

$$\sum_{i=1}^{m} \mu_i = 1. \tag{4.4}$$

Denote by S the feasible region of the original problem (1.1) and S_p the feasible region of the problem (4.1),

$$S = \{y \mid g_i(y) \leq b_i', \ , i = 1, 2, \ldots, m, \ y \in \{0,1\}^{n'}\}, \tag{4.5}$$

$$S_p = \{y \mid \sum_{i=1}^{m} [\mu_i g_i(y)]^p \leq \sum_{i=1}^{m} [\mu_i b_i']^p, \ y \in \{0,1\}^{n'}\}. \tag{4.6}$$

Lemma 1 [22]

$$S = S_p \tag{4.7}$$

if we select

$$p = \lceil \frac{\ln(m)}{\ln(\min_{1 \leq i \leq m} \frac{b_i' + 1}{b_i'})} \rceil, \tag{4.8}$$

where $\lceil q \rceil$ denotes the integer that is greater than or equal to q.

We can conclude from Lemma 1 that the problem (1.1) and the problem (4.1) or (4.2) are exactly the same when p is chosen according to (4.8). Thus, in the following discussion, we only need to consider problem (4.2) and take the advantage of the prominent feature of a single constraint.

In the next section, an example will be solved to demonstrate how the p-norm surrogate constraint method reduces a polynomial zero-one programming problem with multiple constraints into a one with a single surrogate constraint.

4.2 Numerical example

Example 1 *Consider the following polynomial zero-one programming problem with three constraints in [31]:*

$$\min \quad z = 3y_4y_5 + 2y_1y_2 + y_2y_4 + 2y_1y_2y_3 + 8y_2y_3y_5, \tag{4.9}$$
$$\text{s.t.} \quad \begin{cases} g_1(y) = & -y_4y_5 & +y_1y_2 & -y_2y_4 & +y_1y_2y_3 & -y_2y_3y_5 & \leq & 1 \\ g_2(y) = & -7y_4y_5 & & +3y_2y_4 & -4y_1y_2y_3 & -3y_2y_3y_5 & \leq & -2 \\ g_3(y) = & 8y_4y_5 & -6y_1y_2 & -y_2y_4 & -3y_1y_2y_3 & -3y_2y_3y_5 & \leq & -1 \end{cases},$$

where $y = [y_1, y_2, y_3, y_4, y_5] \in \{0,1\}^5$. Note that $g_1(.)$, $g_2(.)$, and $g_3(.)$ are not strictly positive. To use the p-norm surrogate constraint method, a positive integer needs to be added to both sides of each constraint in (4.9) such that the constraint can satisfy the requirement of being strictly positive. After this kind of treatment is performed, the problem (4.9) becomes the following form:

$$\min \quad z = 3y_4y_5 + 2y_1y_2 + y_2y_4 + 2y_1y_2y_3 + 8y_2y_3y_5, \tag{4.10}$$

$$s.t. \begin{cases} -y_4y_5 & +y_1y_2 & -y_2y_4 & +y_1y_2y_3 & -y_2y_3y_5 & +4 & \leq & 5 \\ -7y_4y_5 & & +3y_2y_4 & -4y_1y_2y_3 & -3y_2y_3y_5 & +15 & \leq & 13 \\ 8y_4y_5 & -6y_1y_2 & -y_2y_4 & -3y_1y_2y_3 & -3y_2y_3y_5 & +14 & \leq & 13 \end{cases}.$$

Using (4.3), (4.4), and (4.8), we have $\mu_1 = \frac{13}{23}$, $\mu_2 = \mu_3 = \frac{5}{23}$, *and* $p = 15$. *Applying the p-norm surrogate constraint method we yield an equivalent problem of (4.10),*

$$\begin{aligned} \min \quad & z = 3y_4y_5 + 2y_1y_2 + y_2y_4 + 2y_1y_2y_3 + 8y_2y_3y_5, \qquad (4.11) \\ s.t. \quad & [\frac{13}{23}(-y_4y_5 + y_1y_2 - y_2y_4 + y_1y_2y_3 - y_2y_3y_5 + 4)]^{15} + \\ & [\frac{5}{23}(-7y_4y_5 + 3y_2y_4 - 4y_1y_2y_3 - 3y_2y_3y_5 + 15)]^{15} + \\ & [\frac{5}{23}(8y_4y_5 - 6y_1y_2 - y_2y_4 - 3y_1y_2y_3 - 3y_2y_3y_5 + 14)]^{15} \leq 3(\frac{65}{23})^{15} \end{aligned}$$

Essentially, we testify Lemma 1 with the results in Example 1, i.e., we testify whether the feasible region of the problem (4.9) is equal to that of the problem (4.11). Example 1 has 5 variables such that $y = [y_1, y_2, y_3, y_4, y_5]$ has 32 ($= 2^5$) combinations listed in Table 4.1.

In Table 4.1, the first column is the number of solutions, the second column shows us all 32 solutions. "F" means feasible and "I" means infeasible. In the column of the P-norm Surrogate Constraints Problem are the values of p from 1 to 15. O.P. stands for the original problem (4.9).

From Table 4.1, we can easily make a conclusion that if the problem (4.9) is feasible, the p-norm surrogate constraint problem is feasible no matter what value p is chosen. If the problem (4.9) is infeasible, only when $p \geq 5$ the p-norm surrogate constraint problem is infeasible. Thus, we have that, when $p \geq 15$, the feasible region of the problem (4.9) is equal to that of the problem (4.11), and Lemma 1 is testified to be right again in practice.

No.	y_1	y_2	y_3	y_4	y_5	The P-Norm Surrogate Constraint Problem (p:)															O. P.
						1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
1	0	0	0	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
2	0	0	0	0	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
3	0	0	0	1	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
4	0	0	0	1	1	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
5	0	0	1	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
6	0	0	1	0	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
7	0	0	1	1	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
8	0	0	1	1	1	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
9	0	1	0	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
10	0	1	0	0	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
11	0	1	0	1	0	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
12	0	1	0	1	1	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
13	0	1	1	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
14	0	1	1	0	1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
15	0	1	1	1	0	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
16	0	1	1	1	1	F	F	F	I	I	I	I	I	I	I	I	I	I	I	I	I
17	1	0	0	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
18	1	0	0	0	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
19	1	0	0	1	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
20	1	0	0	1	1	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
21	1	0	1	0	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
22	1	0	1	0	1	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
23	1	0	1	1	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
24	1	0	1	1	1	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
25	1	1	0	0	0	F	F	F	F	I	I	I	I	I	I	I	I	I	I	I	I
26	1	1	0	0	1	F	F	F	F	I	I	I	I	I	I	I	I	I	I	I	I
27	1	1	0	1	0	F	F	I	I	I	I	I	I	I	I	I	I	I	I	I	I
28	1	1	0	1	1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
29	1	1	1	0	0	F	F	F	F	I	I	I	I	I	I	I	I	I	I	I	I
30	1	1	1	0	1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
31	1	1	1	1	0	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I	I
32	1	1	1	1	1	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

Table 4.1: Compare the feasible region of the p-norm surrogate constraint problem and the original problem

Chapter 5

The P-norm Surrogate-constraint Algorithm

5.1 Main ideas

The Balasian-based algorithm [31][32] is efficient in determining whether a feasible solution to the master problem is optimal to the original problem, but the process in searching for all the feasible solutions, using the additive algorithm [1], could be very time consuming. An important reason for this because the total amount of computation in the additive algorithm depends linearly on both the number of constraints and the number of variables, i.e. $m \times n$.

The p-norm surrogate constraint method [22] can reduce the multiple constraints of the polynomial zero-one programming problem to a single one while the number of the decision terms in the master problem (1.4) is enlarged up to n^*. In general, $1 \times n^* < m \times n$ such that the efficiency of the searching process could be enhanced. When the general form of the polynomial zero-one programming problem (1.1) has all the $2^{n'} - 1$ decision terms, the number of the decision terms will remain unchanged after the transformation in the p-norm surrogate constraint method.

In addition, the prominent feature of a single constraint would assist us to devise a more efficient method searching all the feasible solutions than the additive algorithm in the Balasian-based algorithm and even would improve the "backtrack" technique [15].

Based on the above considerations, a new algorithm is sketched for polynomial zero-one programming. The original problem (1.1) is first reduced to an equivalent master problem with its secondary constraints. Then the algorithm searches a feasible solutions to the master problem using an improved "fathoming" technique and checks its consistency. A modified version of "backtrack" technique is adopted on the "fathomed" solution to generate a new nonredundant solution to the master problem and the algorithm goes to the next round of the iterations.

In the following sections, the new algorithm, the p-norm surrogate-constraint algorithm, will be presented in detail.

5.2 The standard form of the polynomial zero-one programming problem

Now, we reconsider the general form of the polynomial zero-one programming problems (1.1) with multiple constraints. It can be reduced to an equivalent polynomial single-constraint zero-one programming problem (4.2) by using the p-norm surrogate constraint method.

Since all y_i, $i = \{1, 2, \ldots, n'\}$, are binary, the surrogate constraint in (4.2) can be simplified to the following form after expansion and combining the similar terms,

$$\sum_{j=1}^{2^{n'}-1} a_j' \prod_{k_j \in K_j} y_{k_j} \le b', \tag{5.1}$$

where a_j' and b' are new constants generated in the process of simplification. Using the definitions of (1.2) and (1.3), the problem (4.2) can be now expressed by the following form:

$$\begin{aligned} \min \quad & z = \sum_{j=1}^{n} c_j x_j, \quad c_j \ge 0, \\ \text{s.t.} \quad & \sum_{j=1}^{n} a_j x_j + s = b, \end{aligned} \tag{5.2}$$

where s is a nonnegative slack variable, a_j and b are deduced from (5.1) and (1.2), and x satisfies (1.2). Up to this stage, the problem (4.2) has been transformed into an equivalent two-level problem referred as the standard form, a *master problem* (5.2) with its *secondary constraints* (1.2). Clearly, the master problem (5.2) is a linear zero-one programming problem with a single constraint while the secondary constraints (1.2) is a polynomial system.

In the following, we will concentrate on developing a numerical algorithm of a partial enumeration nature for the master problem (5.2) with secondary constraints (1.2).

5.3 Definitions and notations

This section will give a set of definitions and notations used in the statements of the new algorithm. Some of these concepts, similar to those adopted in the additive algorithm [31][32], have been redefined in this thesis while the others have been introduced for the first time.

5.3.1 Partial solution in x

A partial solution in x, denoted by J_t, is a permutation of a subset of $\{\pm j | j \in N\}$. The decision term x_j is set at 1 in the tth iteration if $+j \in J_t$, while x_j is set at 0 if $-j \in J_t$. Essentially, J_t determines an assignment of binary values to a subset of the decision terms.

5.3.2 Free term

We define an index set of J_t to be $I(J_t) = \{j \mid +j \text{ or } -j \in J_t\}$. The free term of J_t is any decision term x_j whose index j is not included in $I(J_t)$. Since all $c_j \geq 0$ for $j \in N$ in the master problem (4.4), the free terms are always set at 0.

5.3.3 Completion

A completion of J_t is any vector of decision term, x, in which a part of components are determined by J_t while the rest, all the free terms of J_t, are chosen arbitrarily between 0 and 1.

The partial solution, J_t, behaves exactly as its completion in which all the free terms equal zero. So, we use the partial solution J_t instead of its completion such that the solution process can be simplified.

5.3.4 Feasible partial solution

If the completion of J_t with all corresponding free terms set at zero value constitutes a feasible solution to the master problem (1.4), J_t is called feasible.

If the completion of J_t with all corresponding free terms set at zero value constitutes an infeasible solution to the master problem (1.4), J_t is called infeasible.

5.3.5 Consistent partial solution

A feasible partial solution, J_t, is said to be consistent if J_t determines a feasible solution in y to the secondary constraints in (1.2).

A feasible partial solution, J_t, is said to be inconsistent if J_t leads to an infeasible solution in y to the secondary constraints in (1.2).

If J_t is inconsistent, no matter how it is augmented by its free terms, J_t will remain inconsistent. This feature can improve the efficiency of the "fathoming" procedure since J_t is inconsistent indicates that it is fathomed. So, a new case is added as Case (iii) of the "fathoming" conditions in the new algorithm.

5.3.6 Partial solution in y

A partial solution in y, denoted by D_t, is a permutation of a subset of $\{\pm 1, \pm 2, \ldots, \pm n'\}$ and it is determined by J_t via the secondary constraints. The decision variable y_j is set at 1 in the tth iteration if $+j \in D_t$, while y_j is set at 0 if $-j \in D_t$.

D_t, similar to J_t, determines an assignment of binary values to a subset of the decision variables. If J_t can generate a D_t via the secondary constraints, J_t is consistent. In other words, to check the consistency of J_t is equal to check the existence of D_t.

5.3.7 Free variable

We define an index set of D_t to be $I(D_t) = \{i \mid +i \text{ or } -i \in D_t\}$. The free variable of D_t is any decision variable y_i whose index i is not included in $I(D_t)$.

5.3.8 Augmented solution in x

An augmented solution in x, denoted by B_t, is a subset of $\{+j | j \in N - I(J_t)\}$ and it is determined by D_t via the secondary constraints.

Augmented by B_t, the partial solution J_t must be consistent, but it may be infeasible.

Example 2 *A polynomial zero-one programming problem has been converted into the standard form, a master problem,*

$$\begin{aligned} \min \quad & z = f(x_1, x_2, x_3, x_4), && (5.3) \\ \text{s.t.} \quad & x_1 + 3x_2 + x_3 - 4x_4 + s = -2, \end{aligned}$$

and the secondary constraints,

$$\begin{cases} x_1 &= y_1 y_2, \\ x_2 &= y_2, \\ x_3 &= y_2 y_3 y_4, \\ x_4 &= 1 - y_1 y_2 y_3. \end{cases} \tag{5.4}$$

Clearly, $J^+ = \{1, 2, 3\}$ *and* $J^- = \{4\}$.

At the iteration t*, the feasible partial solution* $J_t = \{+1, -3, +4\}$ *with* x_2 *as a free term determines a following assignment of the decision terms,*

$$\begin{cases} x_1 &= 1, \\ x_3 &= 0, \\ x_4 &= 1, \end{cases} \tag{5.5}$$

and it further determines a following assignment of the decision variables,

$$\begin{cases} y_1 &= 1, \\ y_2 &= 1, \\ y_3 &= 0, \end{cases} \tag{5.6}$$

So the partial solution in y*,* $D_t = \{+1, +2, -3\}$ *and* y_4 *is a free variable.* J_t *is a consistent solution.*

Since $x_2 = y_2$ *in (5.6) and* $+2 \in D_t$*, the free term* x_2 *is fixed to be* 1*. Thus, the augmented solution* $B_t = \{+2\}$*, but* $\{+1, -3, +4, +2\}$*, generated from* J_t *augmented by* B_t*, is infeasible.*

Another feasible partial solution, $J_t = \{+1, +3, +4\}$*, can not satisfy the secondary constraints simultaneously. So it is an inconsistent partial solution.*

5.4 Solution concepts

The feasible solution set of the master problem (5.2) is a relaxation of S. All the optimal solutions to the original problem (1.1) must be feasible to the master problem (5.2). Thus, we are going to develop an approach to find the optimal solution to the problem (1.1) by searching for the best solution among the feasible solutions to (5.2) that satisfy the secondary constraints (1.2). The key point is how to implicitly enumerate the feasible solutions to (5.2). We have shown that, Using the p-norm surrogate constraint method [22], any multiple-constraint polynomial zero-one problem can be reformulated as an equivalent single-constraint one. Making use of this prominent result, we will develop a novel efficient search method that especially suits for singly-constrained polynomial zero-one problems.

5.4.1 Fathoming

Let J_t be a partial solution in x at iteration t. The concept of fathoming is redefined here as follows:

J_t is fathomed if one of the following conditions is satisfied:

(i) J_t is infeasible and J_t has no feasible completion; or

(ii) J_t is both feasible and consistent, and no B_t exists; or

(iii) J_t is feasible, but inconsistent.

In case (i), J_t is infeasible and there does not exist an augmentation to J_t such that the feasibility of the master problem (5.2) can be achieved. In case (ii), J_t with all free terms set at zero is a feasible solution to the original problem (1.1). No B_t exists implies that no necessary augmentation is needed. Since all $c_j \geq 0$, any augmentation to J_t will result in an objective function value which is no better than z_t, the current objective function value associated with J_t. In case (iii), J_t is inconsistent. So are all its completions.

In case where J_t is fathomed, it implies that there is no need to investigate further the remaining completions of J_t, and the "backtrack" process will be performed to generate a new nonredundant partial solution J_{t+1} from J_t.

The "fathoming" process consists of three stages: feasibility checking, consistency checking, and augmenting the partial solution in x. In the fathoming process, $J_{\min}$ and $z_{\min}$ denote the current incumbent solution in x and the corresponding optimal value of the objective function, respectively. At the beginning, set $J_{\min} = \emptyset$ and $z_{\min} = \infty$.

Feasibility checking

At iteration t, the partial solution, J_t, is given and s_t denotes the slack variable. The "fathoming" process starts with the feasibility checking.

If s_t is less than zero, J_t is infeasible and we have case (i); Otherwise, J_t is feasible and we have cases (ii) or (iii).

In case (i), a criterion is easy to be constructed to determine whether J_t has any feasible completion. When s_t is less than the summation of all negative coefficients of free terms, the slack variable cannot become nonnegative even we assign all free terms with negative coefficients at one. More specifically, if

$$\sum_{j \in N - J_t} \min(0, a_j) > s_t, \tag{5.7}$$

J_t has no feasible completion and J_t is fathomed.

If the inequality (5.7) does not hold, J_t has at least one feasible completion. Let $H_t = \emptyset$; find the most negative a_j, $j \in N - J_t - H_t$,; and augment H_t by

$\{+j\}$. This process repeats until $\sum_{j \in H_t} a_j < S_t$ is satisfied. Thus, a new feasible partial solution, $J_t \cup H_t$, is formed and the situation is converted to either case (ii) or (iii).

Consistency checking

If a partial solution, J_t, is feasible, we need to check its consistency for determining which case is used to fathom it. To test whether a partial solution in x, J_t, is consistent or not is equivalent to test if an associated partial solution in y, D_t, can be found.

A two-stage approach is designed to find D_t, which is closely related to the computer program that I coded. In the approach, some of components in D_t can be identified by directly checking both J_t and the secondary constraints and the others need to be identified by iteration.

Stage I. Direct fixation.

If $+j \in J_t$ and $j \in J^+$,

$$y_i = 1 \quad \text{for all } i \in K_j \ \cdot \tag{5.8}$$

If $-j \in J_t$ and $j \in J^-$,

$$y_i = 1 \quad \text{for all } i \in K_j \ \cdot \tag{5.9}$$

Stage II. Indirect fixation.

Step 0.

Set all y_i-variables not fixed at Stage I at an initial value of 2.

Step 1.

Calculate all the values of x_j, $\pm j \in J_t$, according to the secondary constraints (1.2).

Step 2.

For $-j \in J_t$ and $j \in J^+$,

$$\text{If } x_j \begin{cases} > 2, & \text{none to be fixed,} \\ = 2, & y_i \text{ is fixed at 0 for } y_i = 2 \ (i \in K_j), \\ = 1, & J_t \text{ is inconsistent and terminate,} \\ = 0, & \text{none to be fixed.} \end{cases} \tag{5.10}$$

For $+j \in J_t$ and $j \in J^-$,

$$\text{If } x_j \begin{cases} < -1, & \text{none to be fixed,} \\ = -1, & y_i \text{ is fixed at 0 for } y_i = 2 \ (i \in K_j), \\ = 0, & J_t \text{ is inconsistent and terminate,} \\ = 1, & \text{none to be fixed.} \end{cases} \tag{5.11}$$

Step 3.

If no fixation of y_i happens in the current iteration, the procedure terminates; Otherwise, go back to *Step 1.* $\square$

At Stage I, $+j \in J_t$ and $j \in J^+$ imply that $x_j = \prod_{i \in K_j} y_i = 1$. So all y_i, $i \in K_j$, can be fixed at 1, or an inconsistency occurs. $-j \in J_t$ and $j \in J^-$ imply that $x_j = 1 - \prod_{i \in K_j} y_i = 0$. So all y_i, $i \in K_j$, can be also fixed at 1, or an inconsistency occurs.

At Stage II, some other decision variables, y_i, can be fixed at zero by iteration. In Step 0, all y_i which have not been fixed at Stage 1 are set at 2. In other words, $y_i = 2$ indicates that it has not been fixed. So far, all of the decision variables have been fixed at 0, 1, or 2. Based on these new values of y_i, all the values of x_j, $\pm j \in J_t$, are updated in Step 1 of every iteration, which could be different from those determined by J_t.

In Step 2, $-j \in J_t$ and $j \in J^+$ imply that $x_j = \prod_{i \in K_j} y_i = 0$. If the value of x_j, calculated in Step 1, is larger than 2, at least two $y_i = 2$ for $i \in K_j$, i.e., they have not been fixed. Then, none can be determined; If the value of x_j is equal to 2, only one $y_i = 2$, i.e., it has not been fixed. Then, it will be set at zero, or an inconsistency occurs; If the value of x_j is 1, inconsistency occurs and the procedure of consistency checking terminates; If the value of x_j is 0, at least one $y_i = 0$ for $i \in K_j$. Then, no more y_i can be determined.

In Step 2, $+j \in J_t$ and $j \in J^-$ imply that $x_j = 1 - \prod_{i \in K_j} y_i = 1$. If the value of x_j, calculated in Step 1, is less than -1, at least two $y_i = 2$ for $i \in K_j$, i.e., they have not been fixed. Then, none can be determined; If the value of x_j is equal to -1, only one $y_i = 2$, i.e., it has not been fixed. Then, it will be set at zero, or an inconsistency occurs; If the value of x_j is 0, inconsistency occurs and the procedure of consistency checking terminates; If the value of x_j is 1, at least one $y_i = 0$ for $i \in K_j$. Then, no more y_i can be determined.

In Step 3, no fixation of y_i in the current iteration indicates that no fixation will happen in the following iterations. So, the procedure of consistency checking terminates.

Thus, the partial solution in y of J_t is generated in the two-stage approach as follows:

$$D_t = \{+i(-i) | y_i = 1(0) \text{ according to } (5.8), (5.9), (5.10) \text{ and } (5.11)\}.$$

Example 3 *Consider a partial solution* $J_t = \{-1, +3\}$ *with the secondary constraints of (5.4) of Example 2.*

Since $+3 \in J_t$ *and* $x_3 = y_2y_3y_4$, $y_2 = y_3 = y_4 = 1$ *is gotten from (5.8) at Stage I directly. Only* y_1 *has not been fixed.*

At stage II, y_1 *is first set at* 2 *in Step 0. Then* $x_1 = y_1y_2 = 2 \times 1 = 2$ *is calculated in Step 1. Since* $-1 \in J_t$ *and* $1 \in J^+$, y_1 *is fixed at* 0*. So far, all* y_i *for* $i \in N'$ *have been fixed, the procedure of consistency checking terminates.*

When J_t is feasible, if D_t does not exist, J_t is inconsistent and J_t is fathomed in case (iii). Otherwise, J_t is consistent and we have case (ii).

Augmenting the partial solution in x

In case (ii), J_t is both feasible and consistent. When no B_t exists, no necessary augmentation on J_t is needed and J_t is fathomed. Now the completion of J_t with all its free terms set at 0 is a feasible solution to the original problem (1.1). If $z_t < z_{\min}$, set incumbent $z_{\min} = z_t$ and $J_{\min} = J_t$. When B_t exists (there could be more than one B_t), J_t has to be augmented by B_t. A new partial solution, $J_t \cup B_t$, is formed for another round of checking.

An approach suitable to computer program is proposed for determining the augmented solutions in x.

We follow the data in two-level approach that some y_i are fixed at 0, some are fixed at 1 and the rest are equal to 2.

Step 0.

Set $k = 1$. Determine the following three sets:

$$B_t^1 = \{+j | x_j = 1, j \in N - I(J_t)\},$$

$$E_t^+ = \{j | x_j = 2, j \in J^+ \cap [N - I(J_t)]\},$$

$$E_t^- = \{j | x_j = -1, j \in J^- \cap [N - I(J_t)]\}.$$

If $E_t^+ \neq \emptyset$ and $E_t^- \neq \emptyset$, go to the next step. Otherwise, the approach terminates.

Step 1.

Choose an index u from E_t^+. Find v such that $v \in K_u$ and $y_v = 2$.

Step 2.

Delete u from E_t^+ and set $y_v = 0$.

Step 3.

Calculate x_j, $j \in E_t^-$, in accordance with the secondary constraints.

Step 4. Determine the set G_u defined by

$$G_u = \{+j | x_j = 1, \text{ and } j \in E_t^-\}.$$

If $G_u \neq \emptyset$, then $B_t^k = B_t^1 \cup G_u$, and go to the next step. Otherwise, go to Step 6.

Step 5.

Set $k = k + 1$. $B_t^k = B_t^1 \cup \{+u\}$.

Step 6.

If $E_t^+ = \emptyset$, the approach terminates. Otherwise, set $k = k+1$ and go back to Step 1. □

Obviously, $E_t^+ \subset J^+$ and $E_t^- \subset J^-$ are two index sets of the terms x_j, which only contain one y_i that is not fixed. If $G_u \neq \emptyset$, either x_u or x_j for $j \in G_u$ is set at 1 since y_v is binary.

Example 4 *Consider $J_t = \{+3\}$ with the secondary constraints of (5.4) of Example 2. $D_t = \{+2, +3, +4\}$ is obtained easily by the two-stage approach, in which y_1 is set at 2.*

Step 0.

Set $s = 1$, *and*

$$B_t^1 = \{2\},$$

$$E_t^+ = \{+1\} \neq \emptyset,$$

$$E_t^- = \{+4\} \neq \emptyset.$$

Step 1.

Choose $u = 1$ *and find* $v = 1$.

Step 2.

$E_t^+ = \emptyset$ *and* $y_1 = 0$.

Step 3.

$x_4 = 1$.

Step 4.

$G_1 = \{+4\} \neq \emptyset$.

$B_t^1 = B_t^1 \cup G_1 = \{+2, +4\}$.

Step 5.

$k = 2$ *and* $B_t^2 = B_t^1 \cup \{+1\} = \{+2, +1\}$.

Step 6.

Since $E_t^+ = \emptyset$, *the approach terminates.*

Having used the above approach on J_t, either k augmented solutions in y, denoted by B_t^k, are identified or no B_t^k exists. In the former case, although J_t is feasible and consistent, the completion of J_t with all free terms set at zero is not feasible to the secondary constraints until J_t is augmented by B_t^k.

J_{t+1}, generated from augmenting J_t with B_t^k, is obviously consistent and if it is feasible, then its completion with all the free terms set at 0 is a feasible solution to the original problem (1.1). If $z_{t+1} < z_{\min}$, set incumbent $z_{\min} = z_{t+1}$ and $J_{\min} = J_{t+1}$. Since no B_{t+1}^k exists, J_t is also fathomed in case(ii). If J_{t+1} is infeasible, the feasibility checking will go into the next iteration.

5.4.2 Backtracks

When the current partial solution J_t is fathomed, a modified version of Geoffrion's implicit enumeration technique [15] is used to generate a new potential partial solution in x.

The procedure of "backtrack" in the Geoffrion's method [15] makes the most-right positive element in J_t negative and then deletes all the elements to its right. When all the elements of a fathomed partial solution are negative, it means that all 2^n possible solutions to (5.2) have been checked implicitly. One simplification which this algorithm adopts is the treatment of the augmentation B_r^k in J_t at iteration t with $r \leq t$. We recognize that B_r^k is added to make the partial solution J_r, which consists of the components on the left of B_r^k in J_t, consistent. Changing any element of B_r^k from positive to negative while keeping J_r unchanged will result in an inconsistent solution. Thus, we should select the most-right element in J_t from among the elements which do not belong to any B_r^k with $r \leq t$. This modification leads to a significant saving in computation.

When all the elements of a fathomed partial solution are negative, the fathoming process terminates. The optimal partial solution in x is $J_{\min}$.

5.4.3 Determination of the optimal solution in y

In finite iterations, either an optimal value $z_{\min}$ is obtained or it can be concluded that no feasible solution exists from $z_{\min} = \infty$. The optimal partial solution in x is $J_{\min}$ associated with $z_{\min}$. According to the secondary constraints and J_t, we can determine the partial solution(s) in y, $D_{\min}$, by the two-stages approach.

When $N' = I(D_{\min})$, i.e., no free variable exists, the original problem has a single optimal solution in y determined by $D_{\min}$ exactly. When $N' \subset I(D_{\min})$, we choose a group of free variables and set them at 0 or 1 such that all other free variables can be fixed by indirect fixation in the two-stage approach, an optimal solution in y is achieved from the partial setting and the partial fixation. We repeat the above procedure until no such group exits. Finally, the optimal solutions are fully determined.

5.5 Solution algorithm

A polynomial zero-one programming problem has been converted into the standard form, the master problem (5.2) with the second constraints (1.2). The following new algorithm, p-norm surrogate-constraint algorithm, for polynomial zero-one programming is proposed based on the discussion in the previously sections. The detailed steps can be also followed on the flow chart in Figure 5.1.

Step 0. Initialization. Set $t = 0$, $z_{\min} = \infty$, $J_{\min} = J_0 = \emptyset$.

Step 1. Feasibility check. If $s_t > 0$ go to *Step 4*.

Step 2. If

$$\sum_{j \in N-J_t} \min(0, a_j) > s_t,$$

go to *Step 11.*

Step 3. Find the set of H_t. Augment J_t by H_t on the right.

Step 4. Consistency check. If J_t is inconsistent, go to *Step 11.*

Step 5. Find all the sets B_t^k. If B_t^k exist, go to *Step 7.*

Step 6. If $z_t < z_{min}$, replace $z_{\min}$ by z_t and $J_{\min}$ by J_t. Go to *Step 11.*

Step 7. If

$$z_t + \sum_{j \in B_t^k} c_j \geq z_{\min},$$

go to *Step 10.*

Step 8. If

$$S_t - \sum_{j \in B_t^k} a_j \geq 0,$$

replace $z_{\min}$ by $z_t + \sum_{j \in B_t^k} c_j$ and augment J_t on the right by B_t^k, and go to *Step 10.*

Step 9. Augment J_t on the right by B_t^k, then go back to *Step 2.*

Step 10. Set $k = k - 1$. If $k > 0$, return to *Step 7.*

Step 11. If all the elements in J_t are negative; terminate. Otherwise, perform a backtrack step and go back to *Step 1.*

Step 12. Determine all the optimal solutions. If $z_{\min}$ is still equal to infinity when the procedure terminates, the original problem has no feasible solution. Otherwise, those corresponding to $z_{\min}$ is the optimal solutions. □

The p-norm surrogate-constraint algorithm is coded in Visual C++ 5.0 for Pentium 166 CPU. The code of the new algorithm consists of two parts. The first part transforms a general polynomial zero-one problem into its standard form and the second part is the implementation of the above algorithm.

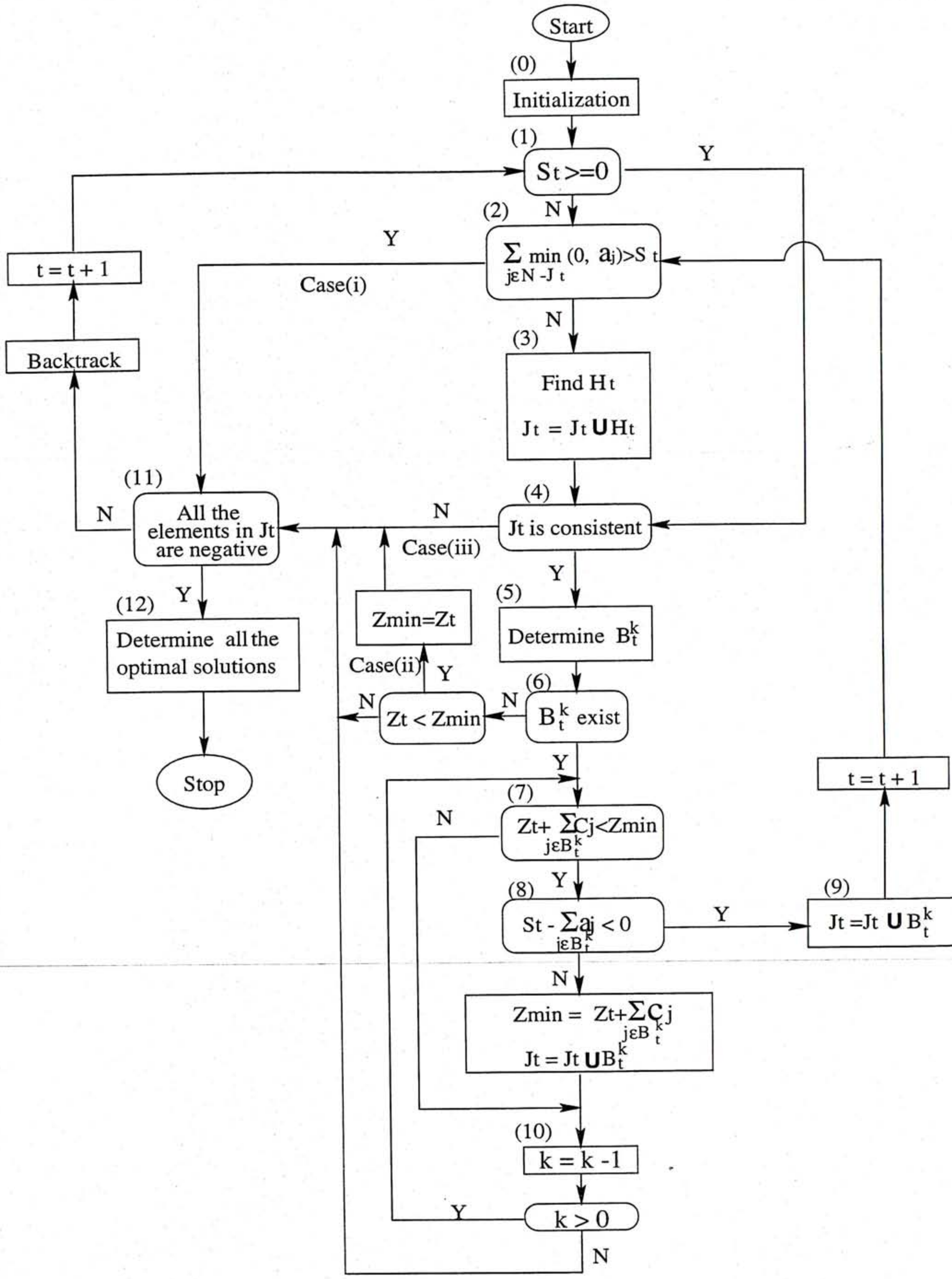

Figure 5.1: The flow chart of p-norm surrogate-constraint algorithm

Chapter 6

Numerical Examples

6.1 Solution process by the new algorithm

6.1.1 Example 5

We re-consider the polynomial zero-one programming problem which Taha [31][32] designed for illustrating the Balasin-based algorithm. The following is the general form of this problem:

$$\min \quad z = 3y_4y_5 + 2y_1y_2 + y_2y_4 + 2y_1y_2y_3 + 8y_2y_3y_5 \tag{6.1}$$

$$s.t. \quad \begin{cases} -y_4y_5 & +y_1y_2 & -y_2y_4 & +y_1y_2y_3 & -y_2y_3y_5 & \leq & 1 \\ -7y_4y_5 & & +3y_2y_4 & -4y_1y_2y_3 & -3y_2y_3y_5 & \leq & -2 \\ 8y_4y_5 & -6y_1y_2 & -y_2y_4 & -3y_1y_2y_3 & -3y_2y_3y_5 & \leq & -1 \end{cases},$$

where $y_i \in \{0,1\}$ for $i \in \{1,2,3,4,5\}$.

Transformation

Using the p-norm surrogate-constraint method [22], the problem (6.1) can be equivalently transformed into the surrogate-constraint formulation (4.11). After expanding and combining the similar terms, (4.11) can be expressed as a polynomial zero-one programming problem with a single surrogate constraint,

$$
\begin{array}{lll}
\min & z = 3y_4y_5 + 2y_1y_2 + y_2y_4 + 2y_1y_2y_3 + 8y_2y_3y_5 & \\
s.t. & +\,41590839946648644904896378599956 & y_4y_5 \\
 & -32393713291612264534830143 & y_1y_2 \\
 & +1892879530909938968922257 50581 & y_2y_4 \\
 & +92677145050818103273888 5152 & y_1y_2y_3 \\
 & -17567584010676854145371905669 & y_2y_3y_5 \\
 & -2089043208862717055203904 78550 & y_1y_2y_3y_4 \\
 & +206437899545917320863442962 6250 & y_1y_2y_3y_4y_5 \\
 & -50635915045620181632181 17738 & y_1y_2y_3y_5 \\
 & +17316800809249363195747 04562 & y_1y_2y_4 \\
 & -2064056286984483290942550140946 & y_1y_2y_4y_5 \\
 & -1855595870738783215333277428950 & y_2y_3y_4y_5 \\
 & -2287482270124103212658524077804 & y_2y_4y_5 \\
 & \leq -13479774448736286144056022 0768. &
\end{array}
\tag{6.2}
$$

Since the operations of the new algorithm to be carried out at each iteration consist solely of additions and subtractions, divided by a large integer, all the coefficients on both sides of the constraint in (6.2) could take the only integer parts without an impact on final result. Thus, the stand form of the problem (6.1) is given by a master problem,

$$\begin{aligned} \min \quad & z = 3x_1 + 2x_2 + x_3 + 2x_4 + 8x_5 \\ s.t. \quad & 41590x_1 - 32x_2 + 1893x_3 + 93x_4 - 176x_5 - 2089x_6 + 20644x_7 \\ & -50x_8 + 17x_9 - 20640x_{10} - 18555x_{11} - 22874x_{12} \leq -134, \end{aligned} \tag{6.3}$$

and its second constraints,

$$\left\{ \begin{array}{lcl} x_1 & = & y_4y_5 \\ x_2 & = & y_1y_2 \\ x_3 & = & y_2y_4 \\ x_4 & = & y_1y_2y_3 \\ x_5 & = & y_2y_3y_5 \\ x_6 & = & y_1y_2y_3y_4 \\ x_7 & = & y_1y_2y_3y_4y_5 \\ x_8 & = & y_1y_2y_3y_5 \\ x_9 & = & y_1y_2y_4 \\ x_{10} & = & y_1y_2y_4y_5 \\ x_{11} & = & y_2y_3y_4y_5 \\ x_{12} & = & y_2y_4y_5 \end{array} \right. . \tag{6.4}$$

Iteration

To clearly illustrate the new algorithm, we shall concentrate on the procedure of solving this example and the details of determining D_t and B_t^k will not presented here.

Step 0. Set $z_{min} = \infty$, $z_0 = 0$, $J_{min} = J_0 = \emptyset$, $s_0 = -134$.

Iteration 0.

Step 1. Feasibility check. $s_0 = -134 < 0$.

Step 2.

$$\sum_{j \in N-J_0} \min(0, a_j) = -32-176-2089-50-20640-18555-22874 = -64416 < s_0 = -134.$$

Step 3. $H_0 = \{12\}$, $J_0 = \{12\}$, and $z_0 = 0$, $s_0 = 22740$.

Step 4. Consistency check. J_0 is consistent and $D_0 = \{2, 4, 5\}$.

Step 5. $B_0^1 = \{1, 3\}$.

Step 7.

$$z_0 + \sum_{j \in B_0^1} c_j = 0 + 3 + 1 = 4 < z_{min} = \infty.$$

Step 8.

$$s_0 - \sum_{j \in B_0^1} a_j = 22740 - 41590 - 1893 = -20743 < 0.$$

Step 9. $J_1 = J_0 \cup B_0^1 = \{12, 1, 3\}$, $z_1 = 4$, and $s_1 = -20743$. Go back to *Step 2*

Iteration 1.

Step 2.

$$\sum_{j \in N-J_1} \min(0, a_j) = -32-176-2089-50-20640-18555 = -41542 < s_1 = -20743.$$

Step 3. $H_1 = \{10, 11\}$, $J_1 = \{12, 1, 3, 10, 11\}$, and $z_1 = 4$, $s_1 = 18452$.

Step 4. Consistency check. J_1 is consistent and $D_1 = \{1, 2, 3, 4, 5\}$.

Step 5. $B_1^1 = \{2, 4, 5, 6, 7, 8, 9\}$.

Step 7.

$$z_1 + \sum_{j \in B_1^1} c_j = 4 + 2 + 2 + 8 = 16 < z_{min} = \infty.$$

Step 8.

$$s_1 - \sum_{j \in B_1^1} a_j = 18452 - 32 + 93 - 176 - 2089 + 20644 - 50 + 17 = 36859 \geq 0,$$

$z_{min} = 16$ and $J_{min} = J_1 \cup B_1^1 = \{12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9\}$, goto *Step 10.*

Step 10. $k = 1 - 1 = 0$.

Step 11. All the elements in J_1 are not negative, backtrack. $J_2 = \{12, 1, 3, 10, -11\}$, $z_2 = 4$, $s_2 = -103$. Go to *Step 1.*

Iteration 2.

Step 1. Feasibility check. $s_2 = -103 < 0$.

Step 2.

$$\sum_{j \in N - J_2} \min(0, a_j) = -32 - 176 - 2089 - 50 = -2374 < s_2 = -103.$$

Step 3. $H_2 = \{6\}$, $J_2 = \{12, 1, 3, 10, -11, 6\}$, and $z_2 = 4$, $s_2 = 1986$.

Step 4. Consistency check. J_2 is inconsistent, go to *Step 11.*

Step 11. All the elements in J_2 are not negative, backtrack. $J_3 = \{12, 1, 3, 10, -11, -6\}$, $z_3 = 4$, $s_3 = -103$. Go to *Step 1.*

Iteration 3.

Step 1. Feasibility check. $s_3 = -103 < 0$.

Step 2.

$$\sum_{j \in N - J_3} \min(0, a_j) = -32 - 176 - 50 = -258 < s_3 = -103.$$

Step 3. $H_3 = \{5\}$, $J_3 = \{12, 1, 3, 10, -11, -6, 5\}$, and $z_3 = 12$, $s_3 = 73$.

Step 4. Consistency check. J_3 is inconsistent, go to *Step 11.*

Step 11. All the elements in J_3 are not negative, backtrack. $J_4 = \{12, 1, 3, 10, -11, -6, -5\}$, $z_4 = 4$, $s_4 = -103$. Go to *Step 1.*

Iteration 4.

Step 1. Feasibility check. $s_4 = -103 < 0$.

Step 2.

$$\sum_{j \in N - J_4} \min(0, a_j) = -32 - 50 = -82 > s_4 = -103,$$

go to *Step 11.*

Step 11. All the elements in J_4 are not negative, backtrack. $J_5 = \{12, 1, 3, -10\}$, $z_5 = 4$, $s_5 = -20743$. Go to *Step 1.*

Iteration 5.

Step 1. Feasibility check. $s_5 = -20743 < 0$.

Step 2.

$$\sum_{j \in N - J_5} \min(0, a_j) = -32 - 176 - 2089 - 50 - 18555 = -20902 < s_5 = -20743.$$

Step 3. $H_5 = \{11, 6, 5\}$, $J_5 = \{12, 1, 3, -10, 11, 6, 5\}$, $z_5 = 12$, $s_5 = 77$.

Step 4. J_5 is inconsistent, go to *Step 11.*

Step 11. All the elements in J_5 are not negative, backtrack. $J_6 = \{12, 1, 3, -10, 11, 6, -5\}$, $z_6 = 4$, $s_6 = -99$. Go to *Step 1.*

Iteration 6.

Step 1. Feasibility. $s_6 = -99 < 0$.

Step 2.

$$\sum_{j \in N - J_6} \min(0, a_j) = -32 - 50 = -82 > s_6 = -99,$$

go to *Step 11.*

Step 11. All the elements in J_6 are not negative, backtrack. $J_7 = \{12, 1, 3, -10, 11, -6\}$, $z_7 = 4$, $s_7 = -2188$. Go to *Step 1.*

Iteration 7.

Step 1. Feasibility check. $s_7 = -2188 < 0$.

Step 2.

$$\sum_{j \in N - J_7} \min(0, a_j) = -32 - 176 - 50 = -258 > s_7 = -2188,$$

go to *Step 11.*

Step 11. All the elements in J_7 are not negative, backtrack. $J_8 = \{12, 1, 3, -10, -11\}$, $z_8 = 4$, $s_8 = -20743$. Go to *Step 1.*

Iteration 8.

Step 1. Feasibility check. $s_8 = -20743 < 0$.

Step 2.

$$\sum_{j \in N - J_8} \min(0, a_j) = -32 - 176 - 2089 - 50 = -2347 > s_8 = -20743,$$

go to *Step 11.*

Step 11. All the elements in J_8 are not negative, backtrack. $J_9 = \{-12\}$, $z_9 = 0$, $s_9 = -134$. Go to *Step 1.*

Iteration 9.

Step 1. Feasibility check. $s_9 = -134 < 0$.

Step 2.

$$\sum_{j \in N - J_9} \min(0, a_j) = -32 - 176 - 2089 - 50 - 20640 - 18555 = -41542 < s_9 = -134.$$

Step 3. $H_9 = \{10\}$, $J_9 = \{-12, 10\}$, $z_9 = 0$, $s_9 = 20506$.

Step 4. Consistency check. J_9 is inconsistent, go to *Step 11.*

Step 11. All the elements in J_9 are not negative, backtrack. $J_{10} = \{-12, -10\}$, $z_{10} = 0$, $s_{10} = -134$. Go to *Step 1.*

Iteration 10.

Step 1. Feasibility check. $s_{10} = -134 < 0$.

Step 2.

$$\sum_{j \in N - J_{10}} \min(0, a_j) = -32 - 176 - 2089 - 50 - 18555 = -20902 < s_{10} = -134.$$

Step 3. $H_{10} = \{11\}$, $J_{10} = \{-12, -10, 11\}$, $z_{10} = 0$, $s_{10} = 18421$.

Step 4. Consistency check. J_{10} is inconsistent, goto *Step 11.*

Step 11. All the elements in J_{10} are not negative, backtrack. $J_{11} = \{-12, -10, -11\}$, $z_{11} = 0$, $s_{11} = -134$. Go to *Step 1.*

Iteration 11.

Step 1. Feasibility check. $s_{11} = -134 < 0$.

Step 2.

$$\sum_{j \in N - J_{11}} \min(0, a_j) = -32 - 176 - 2089 - 50 = -2347 < s_{11} = -134.$$

Step 3. $H_{11} = \{6\}$, $J_{11} = \{-12, -10, -11, 6\}$, $z_{11} = 0$, $s_{11} = 1955$.

Step 4. Consistency check. J_{11} is consistent and $D_{11} = \{1, 2, 3, 4\}$.

Step 5. $B_{11}^1 = \{2, 3, 4, 9\}$.

Step 7.

$$z_{11} + \sum_{j \in B_{11}^1} c_j = 0 + 2 + 1 + 2 + 0 = 5 < z_{min} = 16.$$

Step 8.

$$s_{11} - \sum_{j \in B_{11}^1} a_j = 1955 + 32 - 1893 - 93 - 17 = -16 < 0.$$

Step 9. $J_{12} = J_{11} \cup B_{11}^1 = \{-12, -10, -11, 6, 2, 3, 4, 9\}$, $z_{12} = 5$, and $s_{12} = -16$. Go back to *Step 2.*

Iteration 12.

Step 1. Feasibility check. $s_{12} = -16 < 0$.

Step 2.

$$\sum_{j \in N - J_{12}} \min(0, a_j) = -176 - 50 = -226 < s_{12} = -16.$$

Step 3. $H_{12} = \{5\}$, $J_{12} = \{-12, -10, -11, 6, 2, 3, 4, 9, 5\}$, $z_{12} = 13$, $s_{12} = 160$.

Step 4. Consistency check. J_{12} is inconsistent, goto *Step 11.*

Step 11. All the elements in J_{12} are not negative, backtrack. $J_{13} = \{-12, -10, -11, 6, 2, 3, 4, 9, -5\}$, $z_{13} = 5$, $s_{13} = -16$. Go to *Step 1.*

Iteration 13.

Step 1. Feasibility check. $s_{13} = -16 < 0$.

Step 2.

$$\sum_{j \in N - J_{13}} \min(0, a_j) = -50 < s_{13} = -16.$$

Step 3. $H_{13} = \{8\}$, $J_{13} = \{-12, -10, -11, 6, 2, 3, 4, 9, -5, 8\}$, $z_{13} = 5$, $s_{13} = 34$.

Step 4. Consistency check. J_{13} is inconsistent, go to *Step 11.*

Step 11. All the elements in J_{13} are not negative, backtrack. $J_{14} = \{-12, -10, -11, 6, 2, 3, 4, 9, -5, -8\}$, $z_{14} = 5$, $s_{14} = -16$. Go to *Step 1.*

Iteration 14.

Step 1. Feasibility check. $s_{14} = -16 < 0$.

Step 2.

$$\sum_{j \in N - J_{14}} \min(0, a_j) = 0 > s_{14} = -16,$$

goto *Step 11.*

Step 11. All the elements in J_{14} are not negative, backtrack. $J_{15} = \{-12, -10, -11, -6\}$, $z_{15} = 0$, $s_{15} = -134$. Go to *Step 1.*

Iteration 15.

Step 1. Feasibility check. $s_{15} = -134 < 0$.

Step 2.

$$\sum_{j \in N - J_{15}} \min(0, a_j) = -32 - 176 - 50 = -258 < s_{15} = -134.$$

Step 3. $H_{15} = \{5\}$, $J_{15} = \{-12, -10, -11, -6, 5\}$, $z_{15} = 8$, $s_{15} = 42$.

Step 4. Consistency check. J_{15} is consistent and $D_{15} = \{2, 3, 5\}$.

Step 5. No B_{15}^k exists.

Step 6. $z_{15} = 8 < z_{\min}$, $z_{\min} = z_{15} = 8$, $J_{\min} = J_{15} = \{-12, -10, -11, -6, 5\}$. Go to *Step 11.*

Step 11. All the elements in J_{15} are not negative, backtrack. $J_{16} = \{-12, -10, -11, -6, -5\}$, $z_{16} = 0$, $s_{16} = -134$. Go to *Step 1.*

Iteration 16.

Step 1. Feasibility check. $s_{16} = -134 < 0$.

Step 2.

$$\sum_{j \in N - J_{16}} \min(0, a_j) = -32 - 50 = -82 > s_{16} = -134,$$

goto *Step 11.*

Step 11. All the elements in J_{16} are negative, terminate.

Step 12. $z_{\min} = 8$, $J_{\min} = \{-12, -10, -11, -6, 5\}$, the optimal solution is $y_2 = y_3 = y_5 = 1$ and $y_1 = y_4 = 0$.

The solution process is summarized in Table 6.1:

Iteration	0	1	2	3	4	5
J_t	$\emptyset$	$12, 1, 3$	$12, 1, 3, 10, -11$	$12, 1, 3, 10, -11, -6$	$12, 1, 3, 10, -11, -6, -5$	$12, 1, 3, -10$
z_t	0	4	4	4	4	4
s_t	-134	-20743	-103	-103	-103	-20743
$\sum_{j \in N-J_t} \min(0, a_j)$	-64416	-41542	-2374	-258	-82	-20902
H_t	12	$10, 11$	6	5	Infeasible	$11, 6, 5$
D_t	$2, 4, 5$	$1, 2, 3, 4, 5$	Inconsistent	Inconsistent		Inconsistent
B_t^k	$1, 3$	$2, 4, 5, 6, 7, 8, 9$				
$z_t + \sum_{j \in B_t^k} c_j$	4	16				
$s_t - \sum_{j \in B_t^k} a_j$	-20743	36859				
$J_{\min}$	$\emptyset$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$
$z_{\min}$	∞	16	16	16	16	16

Iteration	6	7	8	9	10	11
J_t	$12, 1, 3, -10, 11, 6, -5$	$12, 1, 3, -10, 11, -6$	$12, 1, 3, -10, -11$	-12	$-12, -10$	$-12, -10, -11$
z_t	4	4	4	0	0	0
s_t	-99	-2188	-20743	-134	-134	-134
$\sum_{j \in N-J_t} \min(0, a_j)$	-82	-258	-2347	-41542	-20902	-2347
H_t	Infeasible	Infeasible	Infeasible	10	11	6
D_t				Inconsistent	Inconsistent	$1, 2, 3, 4$
B_t^k						$2, 3, 4, 9$
$z_t + \sum_{j \in B_t^k} c_j$						5
$s_t - \sum_{j \in B_t^k} a_j$						-16
$J_{\min}$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$
$z_{\min}$	16	16	16	16	16	16

Iteration	12	13	14	15	16	
J_t	$-12, -10, -11, 6, 2, 3, 4, 9$	$-12, -10, -11, 6, 2, 3, 4, 9, -5$	$-12, -10, -11, 6, 2, 3, 4, 9, -5, -8$	$-12, -10, -11, -6$	$-12, -10, -11, -6, -5$	
z_t	5	5	5	0	0	
s_t	-16	-16	-16	-134	-134	
$\sum_{j \in N-J_t} \min(0, a_j)$	-226	-50	0	-258	-82	
H_t	5	8	Infeasible	5	Infeasible	
D_t	Inconsistent	Inconsistent		$2, 3, 5$		
B_t^k				$\emptyset$		
$z_t + \sum_{j \in B_t^k} c_j$						
$s_t - \sum_{j \in B_t^k} a_j$						
$J_{\min}$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9$	$-12, -10, -11, -6, 5$	$-12, -10, -11, -6, 5$	
$z_{\min}$	16	16	16	8	8	

Table 6.1: The iterative solution process of the problem of example 5

6.1.2 Example 6

Now we solve another polynomial zero-one programming problem using the p-norm surrogate-constraint algorithm. The problem is given as follows:

$$\min \quad z = 3y_1 + 2y_2 + 3y_3 + 2y_1y_2 + 8y_1y_3 + 4y_2y_3 + y_1y_2y_3 \tag{6.5}$$

$$s.t. \begin{cases} -y_1 \ +y_2 \ -y_3 \ +y_1y_2 \ -y_1y_3 \ +y_1y_2y_3 \leq 1 \\ y_1 \ +3y_3 \ -4y_1y_2 \ +3y_1y_3 \ +2y_2y_3 \ -y_1y_2y_3 \leq -2 \\ 2y_1 \ +4y_2 \ -y_3 \ -3y_1y_2 \ +3y_1y_3 \ +y_2y_3 \ +2y_1y_2y_3 \leq -1 \end{cases},$$

where all $y_i \in \{0,1\}$ for $i \in \{1,2,3,4,5\}$. Note that this problem has 3 variables and all $2^3 - 1 = 7$ terms.

By the proposed equivalent transformations developed in this thesis, (6.9) is converted into a master problem,

$$\begin{aligned} \min \quad & z = 3x_1 + 2x_2 + x_3 + 2x_4 + 8x_5 + 4x_6 + x_7 \\ s.t. \quad & 10x_1 + 36x_2 + 34x_3 - 33x_4 + 480x_5 + 115x_6 - 78x_7 \leq -2, \end{aligned} \tag{6.6}$$

and its second constraints,

$$\begin{cases} x_1 = y_1 \\ x_2 = y_2 \\ x_3 = y_3 \\ x_4 = y_1y_2 \\ x_5 = y_1y_3 \\ x_6 = y_2y_3 \\ x_7 = y_1y_2y_3 \end{cases} \tag{6.7}$$

Step 0. Set $z_{min} = \infty$, $z_0 = 0$, $J_{min} = J_0 = \emptyset$, $s_0 = -2$.

Iteration 0.

Step 1. Feasibility check. $s_0 = -2 < 0$.

Step 2.

$$\sum_{j \in N - J_0} \min(0, a_j) = -111 < s_0 = -2.$$

Step 3. $H_0 = \{7\}$, $J_0 = \{7\}$, and $z_0 = 1$, $s_0 = 76$.

Step 4. Consistency check. J_0 is consistent and $D_0 = \{1, 2, 3\}$.

Step 5. $B_0^1 = \{1, 2, 3, 4, 5, 6\}$.

Step 7.

$$z_0 + \sum_{j \in B_0^1} c_j =< z_{min} = \infty.$$

Step 8.

$$s_0 - \sum_{j \in B_0^1} a_j = -466 < 0.$$

Step 9. $J_1 = J_0 \cup B_0^1 = \{7, 1, 2, 3, 4, 5, 6\}$, $z_1 = 21$, and $s_1 = -566$. Go back to *Step 2*

Iteration 1.

Step 2.

$$\sum_{j \in N - J_1} \min(0, a_j) = 0 > s_1 = -566.$$

Step 11. All the elements in J_1 are not negative, backtrack. $J_2 = \{-7\}$, $z_2 = 0$, $s_2 = -2$. Go to *Step 1.*

Iteration 2.

Step 1. Feasibility check. $s_2 = -2 < 0$.

Step 2.

$$\sum_{j \in N - J_2} \min(0, a_j) = -33 < s_2 = -2.$$

Step 3. $H_2 = \{4\}$, $J_2 = \{-7, 4\}$, and $z_2 = 2$, $s_2 = 31$.

Step 4. Consistency check. J_2 is consistent and $D_2 = \{1, 2\}$.

Step 5. $B_2^1 = \{1, 2\}$.

Step 7.

$$z_2 + \sum_{j \in B_2^1} c_j =< z_{min} = \infty.$$

Step 8.

$$s_2 - \sum_{j \in B_2^1} a_j = -15 < 0.$$

Step 9. $J_3 = J_2 \cup B_0^1 = \{-7, 4, 1, 2\}$, $z_3 = 7$, and $s_3 = -15$. Go back to *Step 2*

Iteration 3.

Step 1. Feasibility check. $s_3 = -15 < 0$.

Step 2.

$$\sum_{j \in N - J_3} \min(0, a_j) = 0 > s_3 = -15.$$

Step 11. All the elements in J_3 are not negative, backtrack. $J_4 = \{-7, -4\}$, $z_4 = 0$, $s_4 = -2$. Go to *Step 1.*

Iteration 4.

Step 1. Feasibility check. $s_4 = -2 < 0$.

Step 2.

$$\sum_{j \in N - J_4} \min(0, a_j) = 0 > s_4 = -2.$$

Step 11. All the elements in J_4 are negative, terminate. Since $z_{min} = \infty$, the problem has no feasible solution.

The solution process of Example 6 is summarized in Table 6.2.

Iteration	0	1	2	3	4
J_t	$\emptyset$	$7, 1, 2, 3, 4, 5, 6$	-7	$-7, 4, 1, 2$	$-7, -4$
z_t	0	21	0	7	0
s_t	-2	-566	-2	-15	-2
$\sum_{j \in N - J_t} \min(0, a_j)$	-111	0	-33	0	0
H_t	7	Infeasible	4	Infeasible	Infeasible
D_t	$1, 2, 3$		$1, 2$		
B_t^k	$1, 2, 3, 4, 5, 6$		$1, 2$		
$z_t + \sum_{j \in B_t^k} c_j$	21		5		
$s_t - \sum_{j \in B_t^k} a_j$	-466		-15		
$J_{\min}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
$z_{\min}$	∞	∞	∞	∞	∞

Table 6.2: The iterative solution process of the problem of example 6

6.2 Solution process by the Balasian-based algorithm

In this section, the Balasian-based algorithm will be used to solve Example 5 for comparing with the p-norm surrogate-constraint algorithm. We first transform (6.1) into a master problem,

$$\min \quad z = 3x_1 + 2x_2 + x_3 + 2x_4 + 8x_5, \tag{6.8}$$
$$s.t. \begin{cases} -x_1 & +x_2 & -x_3 & +x_4 & -x_5 & \leq & 1 \\ -7x_1 & & +3x_3 & -4x_4 & -3x_5 & \leq & -2 \\ 8x_1 & -6x_2 & -x_3 & -3x_4 & -3x_5 & \leq & -1 \end{cases},$$

and its secondary constraints,

$$\begin{cases} x_1 & = & y_4y_5 \\ x_2 & = & y_1y_2 \\ x_3 & = & y_2y_4 \\ x_4 & = & y_1y_2y_3 \\ x_5 & = & y_2y_3y_5 \end{cases}, \tag{6.9}$$

where x_i for $i \in \{1,2,3,4,5\}$ and y_j for $j \in \{1,2,3,4,5\}$ are binary.

Step 0. Set $z_{min} = \infty$, $J_0 = \emptyset$, $z_0 = 0$, $s^0 = (1,-2,-1)^T$.

Iteration 1.

Step 1. Applying the additive algorithm to (6.8), the first feasible partial solution is given by $J_1 = \{5\}$ with $z_1 = 8$ and $S^1 = (2,1,2)^T$.

Iteration 1.1

Step 1. $s_i^0 < 0$ for $i = 2, 3$.

Step 2. $C^0 = D_0 = E_0 = \emptyset$, $N_0 = N - (C^0 \cup D_0 \cup E_0) = N = \{1, 2, 3, 4, 5\}$.

Step 3. We check the following relations for $i = 2, 3$:

$$\sum_{j \in N_0} \overline{a}_{2j} = \overline{a}_{21} + \overline{a}_{24} + \overline{a}_{25} = -7 - 4 - 3 = -14 < s_2^0 = -2,$$

$$\sum_{j \in N_0} \overline{a}_{3j} = \overline{a}_{32} + \overline{a}_{33} + \overline{a}_{34} + \overline{a}_{35} = -6 - 1 - 3 - 3 = -13 < s_3^0 = -1.$$

Since all relations hold as strict inequalities, compute the values v_j^0 for $j \in N_0$:

$$v_1^0 = \sum_{i \in M_1^{0-}} (s_i^0 - a_{i1}) = -9,$$

$$v_2^0 = \sum_{i \in M_2^{0-}} (s_i^0 - a_{i2}) = -2,$$

$$v_3^0 = \sum_{i \in M_3^{0-}} (s_i^0 - a_{i3}) = -5,$$

$$v_4^0 = \sum_{i \in M_4^{0-}} (s_i^0 - a_{i4}) = 0,$$

$$v_5^0 = \sum_{i \in M_5^{0-}} (s_i^0 - a_{i5}) = 0.$$

We have $v_5^0 = \max_{j \in N_0}\{v_j^0\} = 0$, so cancel it and pass to

Step 8. $J_1 = J_0 \cup \{5\}$.

$$s_1^1 = s_1^0 - a_{15} = 2,$$

$$s_2^1 = s_2^0 - a_{25} = 1,$$

$$s_3^1 = s_3^0 - a_{35} = 2.$$

$s_i^1 \geq 0$ for $i = 1, 2, 3$, we then get a feasible partial solution $\{5\}$. □

Step 2. J_1 is consistent. The corresponding solution in y_{k_j} is $y_2 = y_3 = y_5 = 1$ and $y_1 = y_4 = 0$.

Step 3. From the solution in *Step 2*, all the free terms are equal to zero. Hence $B_1 = \emptyset$. Thus $z_{min} = 8$ and

$$J^* = \{5\},$$

$$y_2^* = y_3^* = y_5^* = 1, y_1^* = y_4^* = 0,$$

$$S^* = (2, 1, 2)^T.$$

Step 9. $J_2 = \{-5\}$, $z_2 = 0$, $S^2 = (1, -2, -1)^T$.

Iteration 2.

Step 1. Applying the additive algorithm to (6.8), the next feasible partial solution is given by $J_3 = \{-5, 4\}$ with $z_3 = 2$ and $S^3 = (0, 2, 2)^T$.

Iteration 2.1

Step 1. $s_i^0 < 0$ for $i = 2, 3$.

Step 2. $C^0 = D_0 = E_0 = \emptyset$, $N_0 = N - (C^0 \cup D_0 \cup E_0) = N = \{1, 2, 3, 4\}$.

Step 3. We check the following relations for $i = 2, 3$:

$$\sum_{j \in N_0} \overline{a}_{2j} = \overline{a}_{21} + \overline{a}_{24} = -7 - 4 = -11 < s_2^0 = -2,$$

$$\sum_{j \in N_0} \overline{a}_{3j} = \overline{a}_{32} + \overline{a}_{33} + \overline{a}_{34} = -6 - 1 - 3 = -10 < s_3^0 = -1.$$

Since all relations hold as strict inequalities, compute the values v_j^0 for $j \in N_0$:

$$v_1^0 = \sum_{i \in M_1^{0-}} (s_i^0 - a_{i1}) = -6,$$

$$v_2^0 = \sum_{i \in M_2^{0-}} (s_i^0 - a_{i2}) = -2,$$

$$v_3^0 = \sum_{i \in M_3^{0-}} (s_i^0 - a_{i3}) = -5,$$

$$v_4^0 = \sum_{i \in M_4^{0-}} (s_i^0 - a_{i4}) = -1.$$

We have $v_4^0 = \max_{j \in N_0}\{v_j^0\} = -1$, so cancel it and pass to

Step 8. $J_1 = J_0 \cup \{4\}$.

$$s_1^1 = s_1^0 - a_{14} = 0,$$

$$s_2^1 = s_2^0 - a_{24} = 2,$$

$$s_3^1 = s_3^0 - a_{34} = 2.$$

$s_i^1 \geq 0$ for $i = 1, 2, 3$, we then get a feasible partial solution $\{-5, 4\}$. $\square$

Step 2. J_3 is consistent. The corresponding solution in y_{k_j} is $y_1 = y_2 = y_3 = 1$ and $y_4 = y_5 = 0$.

Step 3. From the solution in *Step 2*, $x_2 = y_1 y_2 = 1$. Hence, $B_3 = \{2\}$.

Step 4. $z_3 + c_2 = 2 + 2 < z_{\min} = 8$.

Step 5.

$$\begin{aligned} s_1^3 - a_{12} &= \; 0 - 1 \; = -1 \; < 0, \\ s_2^3 - a_{22} &= \; 2 - 0 \; = 2 \quad > 0, \\ s_3^3 - a_{32} &= \; 2 + 6 \; = 8 \quad > 0. \end{aligned}$$

Hence augmentation of J_3 by $\{2\}$ can not result in a feasible partial solution to (6.8).

Step 6.

$$\begin{aligned} x_1 : \; z_3 + c_1 &= 2 + 3 \; < 8 (= z_{\min}), \\ x_2 : \; z_3 + c_2 &= 2 + 2 \; < 8, \\ x_3 : \; z_3 + c_3 &= 2 + 1 \; < 8. \end{aligned}$$

Hence $R_3 = \{1, 2, 3\} = \emptyset$.

Step 7.

$$x_1: \quad a_{31} = 8 > s_3^3 = 2,$$

$$x_2: \quad a_{12} = 1 > s_1^3 = 0,$$

$$x_3: \quad a_{23} = 3 > s_2^3 = 2.$$

Hence $Q_3 = \emptyset$.

Step 8.

$$\begin{aligned} x_1: \; w_1^3 &= 0 + 0 + (2-8) &&= -6, \\ x_2: \; w_2^3 &= (0-1) + 0 + 0 &&= \boxed{-1}, \\ x_3: \; w_3^3 &= 0 + (2-3) + 0 &&= -1. \end{aligned}$$

Thus $r = 2$ and $J_4 = \{-5, 4, 2\}$ with $z_4 = 4$ and $S^4 = (-1, 2, 8)^T$.

Iteration 3.

Step 1. Application of the additive algorithm yields a new feasible partial solution, $J_5 = \{-5, 4, 2, 1\}$ with $z_5 = 7$ and $S^5 = (0, 9, 0)^T$.

Iteration 3.1

Step 1. $s_i^0 < 0$ for $i = 1$.

Step 2. $N_0 = N - (C^0 \cup D_0 \cup E_0) = N = \{1, 3\}$.

Step 3. We check the following relations for $i = 1$:

$$\sum_{j \in N_0} \overline{a}_{1j} = \overline{a}_{11} + \overline{a}_{13} = -1 - 1 = -2 < s_1^0 = -1.$$

Since the relation holds as a strict inequality, compute the values v_j^0 for $j \in N_0$:

$$v_1^0 = \sum_{i \in M_1^{0-}} (s_i^0 - a_{i1}) = 0,$$

$$v_3^0 = \sum_{i \in M_3^{0-}} (s_i^0 - a_{i3}) = -1.$$

We have $v_3^0 = \max_{j \in N_0}\{v_j^0\} = 0$, so cancel it and pass to

Step 8. $J_1 = J_0 \cup \{1\}$.

$$s_1^1 = s_1^0 - a_{11} = 0,$$

$$s_2^1 = s_2^0 - a_{21} = 9,$$

$$s_3^1 = s_3^0 - a_{31} = 0.$$

$s_i^1 \geq 0$ for $i = 1, 2, 3$, we then get a feasible solution $\{-5, 4, 2, 1\}$. □

Step 2. J_5 is inconsistent.

Step 9. $J_6 = \{-5, 4, 2, -1\}$ with $z_6 = 4$ and $S^6 = (-1, 2, 8)^T$.

Iteration 4.

Step 1. Application of the additive algorithm can not find any partial feasible solution for J_6.

Iteration 4.1

Step 1. $s_i^0 < 0$ for $i = 1$.

Step 2. $N_0 = N - (C^0 \cup D_0 \cup E_0) = N = \{3\}$.

Step 3. We check the following relation for $i = 1$:

$$\sum_{j \in N_0} \overline{a}_{1j} = \overline{a}_{11} = -1 = s_1^0 = -1.$$

Step 4. We cancel it and get a partial solution $J_1 = \{3\}$.

$$s_1^1 = s_1^0 - a_{13} = 0,$$

$$s_2^1 = s_2^0 - a_{23} = -1,$$

$$s_3^1 = s_2^0 - a_{33} = 9.$$

Iteration 4.2

Step 1. $s_i^1 < 0$ for $i = 2$.

Step 2. $N_1 = \emptyset$.

Step 5. No feasible partial solution. Backtrack and get $J_2 = \{-3\}$ with $S^2 = (-1, 2, 8)^T$.

Iteration 4.3

Step 1. $s_i^2 < 0$ for $i = 1$.

Step 2. $N_2 = \emptyset$.

Step 5. No feasible partial solution. The process terminates. □

Step 9. Backtrack and get a partial solution $J_7 = \{-5, 4, -2\}$ with $z_7 = 2$ and $S^7 = (0, 2, 2)^T$.

Iteration 5.

Step 1. J_7 is a feasible partial solution.

Step 2. J_7 is inconsistent.

Step 9. $J_8 = \{-5, -4\}$ with $z_8 = 0$ and $S^8 = (1, -2, -1)^T$.

Iteration 6.

Step 1. Application of the additive algorithm can not find any feasible partial solution for J_8. Since all the elements in J_8 are negative, the procedure terminates.

Iteration 6.1

Step 1. $s_i^0 < 0$ for $i = 2, 3$.

Step 2. $N_0 = N - (C^0 \cup D_0 \cup E_0) = N = \{1, 2, 3\}$.

Step 3. We check the following relations for $i = 2, 3$:

$$\sum_{j \in N_0} \overline{a}_{2j} = \overline{a}_{21} = -7 < s_2^0 = -2,$$

$$\sum_{j\in N_0} \overline{a}_{3j} = \overline{a}_{32} + \overline{a}_{33} = -6 - 1 = -7 < s_3^0 = -1.$$

Since the relation hold as strict inequalities, compute the values v_j^0 for $j \in N_0$:

$$v_1^0 = \sum_{i\in M_1^{0-}} (s_i^0 - a_{i1}) = -9,$$

$$v_2^0 = \sum_{i\in M_2^{0-}} (s_i^0 - a_{i2}) = \boxed{-2},$$

$$v_3^0 = \sum_{i\in M_3^{0-}} (s_i^0 - a_{i3}) = -5.$$

We have $v_2^0 = \max_{j\in N_0}\{v_j^0\} = -2$. So, cancel it and pass to

Step 8. $J_1 = J_0 \cup \{2\} = \{2\}$.

$$s_1^1 = s_1^0 - a_{12} = 0,$$

$$s_2^1 = s_2^0 - a_{22} = -2,$$

$$s_3^1 = s_3^0 - a_{32} = 5.$$

Iteration 6.2

Step 1. $s_i^1 < 0$ for $i = 2$.

Step 2. $N_1 = \{1, 3\}$.

Step 3. We check the following relation for $i = 2$:

$$\sum_{j\in N_1} \overline{a}_{2j} = \overline{a}_{21} = -7 < s_2^1 = -2.$$

Since the relation holds as a strict inequality, compute the values v_j^1 for $j \in N_1$:

$$v_1^1 = \sum_{i\in M_1^{0-}} (s_i^1 - a_{i1}) = \boxed{-3},$$

$$v_3^1 = \sum_{i\in M_3^{0-}} (s_i^1 - a_{i3}) = -5.$$

We have $v_1^1 = \max_{j \in N_1}\{v_j^1\} = -3$, so cancel it and pass to

Step 8. $J_2 = J_1 \cup \{1\} = \{2, 1\}$.

$$s_1^2 = s_1^1 - a_{11} = 1,$$

$$s_2^2 = s_2^1 - a_{21} = 5,$$

$$s_3^2 = s_3^1 - a_{31} = -3.$$

Iteration 6.3

Step 1. $s_i^2 < 0$ for $i = 3$.

Step 2. $N_2 = \{3\}$.

Step 3. We check the following relation for $i = 3$:

$$\sum_{j \in N_2} \overline{a}_{3j} = \overline{a}_{33} = -1 > s_3^2 = -3.$$

Step 5. No feasible partial solution. Backtrack and get a partial solution $J_3 = \{2, -1\}$ with $S^3 = (0, -2, 5)^T$.

Iteration 6.4

Step 1. $s_i^3 < 0$ for $i = 2$.

Step 2. $N_3 = \emptyset$.

Step 5. No feasible solution. Backtrack and get a partial solution $J_4 = \{-2\}$ with $S^4 = (1, -2, -1)^T$.

Iteration 6.5

Step 1. $s_i^4 < 0$ for $i = 2, 3$.

Step 2. $N_4 = \{1, 3\}$.

Step 3. We check the following relations for $i = 2, 3$:

$$\sum_{j \in N_4} \overline{a}_{2j} = \overline{a}_{21} = -7 < s_2^4 = -2,$$

$$\sum_{j \in N_4} \overline{a}_{3j} = \overline{a}_{33} = -1 = s_3^4 = -1.$$

Step 4. We get a new partial solution, $J_5 = \{-2, 3\}$, with $S^5 = (2, -5, 0)^T$.

Iteration 6.6

Step 1. $s_i^5 < 0$ for $i = 2$.

Step 2. $N_5 = \{1\}$.

Step 3. We check the following relation for $i = 2$:

$$\sum_{j \in N_5} \overline{a}_{2j} = \overline{a}_{21} = -7 < s_2^5 = -5.$$

Since the relation holds as a strict inequality, compute the values v_j^5 for $j \in N_5$:

$$v_1^5 = \sum_{i \in M_1^{0-}} (s_i^5 - a_{i1}) = -8.$$

So, cancel it and pass to

Step 8. $J_6 = J_5 \cup \{1\} = \{-2, 3, 1\}$.

$$s_1^6 = s_1^5 - a_{11} = 3,$$

$$s_2^2 = s_2^1 - a_{21} = 2,$$

$$s_3^2 = s_3^1 - a_{31} = -8.$$

Iteration 6.7

Step 1. $s_i^6 < 0$ for $i = 3$.

Step 2. $N_6 = \emptyset$.

Step 3. No feasible partial solution. Backtrack and get $J_7 = \{-2, 3, -1\}$ with $S^7 = (2, -5, 0)^T$.

Iteration 6.8

Step 1. $s_i^7 < 0$ for $i = 2$.

Step 2. $N_7 = \emptyset$.

Step 3. No feasible partial solution. Backtrack and get $J_8 = \{-2, -3\}$ with $S^8 = (1, -2, -1)^T$.

Iteration 6.9

Step 1. $s_i^8 < 0$ for $i = 2, 3$.

Step 2. $N_8 = \{1\}$.

Step 3. We check the following relations for $i = 2, 3$:

$$\sum_{j \in N_8} \overline{a}_{2j} = \overline{a}_{21} = -7 < s_2^8 = -2,$$

$$\sum_{j \in N_8} \overline{a}_{3j} = 0 > s_3^8 = -1.$$

Step 5. No feasible partial solution. The process terminates. □

$z_{\min} = 8$, $J^* = \{5\}$, the optimal solution is $y_2^* = y_3^* = y_5^* = 1$ and $y_1^* = y_4^* = 0$.

6.3 Comparison between the p-norm surrogate constraint algorithm and the Balasian-based algorithm

We have solved the polynomial zero-one programming problem (6.1) using both the p-norm surrogate-constraint algorithm and the additive algorithm. We have also solved the problem (6.9) using the p-norm surrogate-constraint algorithm. Now, we will make some comparisons between these two algorithms.

The new algorithm solves the problem (6.1) within 16 iterations and the Balasian-based algorithm solves the problem (6.1) within only 6 iterations. It seems that the latter algorithm is much better than the former in the view of the number of iterations. However, each iteration in the Balasian-based algorithm except iteration 5 needs to apply the additive algorithm [1] to search for a feasible solution to the master problem. There are 17 iterations to be counted in for this purpose. Essentially, the Balasian-based algorithm needs total 21 iterations to solve the problem (6.1).

In the Balasian-based algorithm, the additive algorithm [1] plays the role of searching all the feasible solutions to the linear master problem. Its computational amount linearly depends not only on the number of the decision terms but also on the number of the constraints. The p-norm surrogate-constraint algorithm can easily finish this job with the help of checking all the coefficients of the surrogate constraint, so its computational amount only depends on the number of the decision terms. Especially, when a polynomial zero-one programming problem is in an all-combination form, i.e., it has all the $2^{n'}-1$ decision terms, the number of the decision terms will remain unchanged after the transformation in the p-norm surrogate constraint method. Example 6 is an all-combination problem. The number of decision terms is still $2^3-1=7$ after the transformation and only 4 iterations are performed in the solution process. Obviously, the searching strategy of the new algorithm is more efficient than the additive algorithm. Thus, we can conclude that the p-norm surrogate-constraint algorithm is more efficient.

The new version of the Geoffrion's implicit enumeration technique also accelerates the new algorithm. It may skip from a partial solution to the next one in a big backtrack step according to some rules. In Iteration 1 of Example 5, we backtrack $J_1 = \{12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9\}$ to $J_2 = \{12, 1, 3, 10, -11\}$ directly, but the traditional Geoffrion's implicit enumeration technique only backtracks $J_1 = \{12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, 9\}$ to $J_2 = \{12, 1, 3, 10, 11, 2, 4, 5, 6, 7, 8, -9\}$. In Iteration 1 of Example 6, we prefer to backtrack $J_1 = \{7, 1, 2, 3, 4, 5, 6\}$ to $J_2 = \{-7\}$ rather than to $J_2 = \{7, 1, 2, 3, 4, 5, -6\}$.

Chapter 7

Application to the Set Covering Problem

7.1 The set covering problem

Many real-word problems, such as the crew scheduling problem in railway and mass-transit transportation companies [3][8], could be modeled as the set covering problem (SCP). The general form of SCP may be expressed as follows:

$$\begin{aligned} \min z &= \sum_{j=1}^{n} c_j x_j, & c_j > 0, \\ s.t. & \quad \sum_{j=1}^{n} a_{ij} x_j \leq 1, \quad i = 1, 2, \ldots, m, & \end{aligned} \tag{7.1}$$

where the decision variables $x_j \in \{0, 1\}$ for $i = 1, 2, \ldots, n$. All coefficients a_{ij} are either 0 or 1. The right-hand-side of each constraint is always equal to 1. The coefficient matrix is denoted by $A = [a_{ij}]_{m \times n}$ in which m represents the number of the constraints in the problem (7.1) and n represents the number of n devision variables in the problem (7.1). If $a_{ij} = 1$, we say that the jth column covers the ith row. Let c_j represent the cost of column j. SCP can be interpreted as a problem to search for a minimum-cost subset $S \subseteq \{1, 2, \cdots, n\}$ of columns such that each row is covered by at least one column. The coefficient matrix A often has a large density of 0. In other words, the number of entries of 1 is much smaller than $m \times n$ in general.

SCP is NP-hard in the strong sense [14], and is difficult to solve from the point of view of the theoretical approximation [24]. However, due to the structure of certain real-world instances of the problem, many algorithms including both heuristic [4][2][23][9][12][7] and exact approaches [2][26][5][6] have been derived to perform efficiently on these instances. The current state of the art on the problem is that instances with a few hundred rows and a few thousand columns can be solved exactly, and instances with a few thousand rows and a few millions columns can be solved within about 1% of the optimum value with a reasonable computing time.

7.2 Solving the set covering problem by using the new algorithm

As stated before, SCP is a linear zero-one problem. We first reduce the multiple constraints into a single constraint using the p-norm surrogate constraint method, then model it as the standard form, i.e., the master problem with a single constraint and its secondary constraints. Finally, we solve it using the p-norm surrogate-constraint algorithm.

A series of test problems are given to demonstrate the solution procedure using the new algorithm. All the test problems have a same objective function,

$$\min z = 10y_1 + 2y_2 + 3y_3 + 4y_4 + 5y_5 + 6y_6 + 7y_7, \tag{7.2}$$

where decision variables $y_i \in \{0, 1\}$ for $i = 1, 2, \ldots, 7$, but they have different constraints. We compose 5 constraint matrices for the same objective function

(7.2) by choosing different row from a matrix A which is defined,

$$\begin{bmatrix} -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 \\ -1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 & 0 \end{bmatrix}, \tag{7.3}$$

The first problem picks up 3 rows on the top of the matrix A as its coefficient matrix with $m = 3$. The second one picks up 5 rows on the top as its coefficient matrix. The third and fourth choose the first 10 and 15 rows separately as

Problem	Constraints(m)	p	F. S.	Iteration
1	3	3	5	26
2	5	4	10	32
3	10	6	20	69
4	15	7	8	69
5	17	7	5	71

Table 7.1: The set covering problem

their coefficient matrices. The whole matrix A is the coefficient matrix of the last problem with $m = 17$. Computation results of these 5 test problems with different constraint matrix are listed in Table 7.1 where the column of constraints means the number of constraints a problem has, the column of p means the value of p taken in the p-norm surrogate constraint method [22], the column of F. S. means the number of feasible solutions checked in the linear master problem, and the column of iteration means the number of iterations to reach the optimal solution.

From Table 7.1, it is not difficult to get some crude observations. The value of p increases slower than the number of constraints. m becomes from 3 to 17, but p changes from 3 to 7 on a small scale. With the increasing of the value of m, the amount of checking feasible solution increases first and then decreases. When m is between 5 and 10, the amount of feasible solution checked is up to peak.

The number of iterations is an important index to measure the efficiency of an algorithm. We can see a pattern in the relationship between the iteration

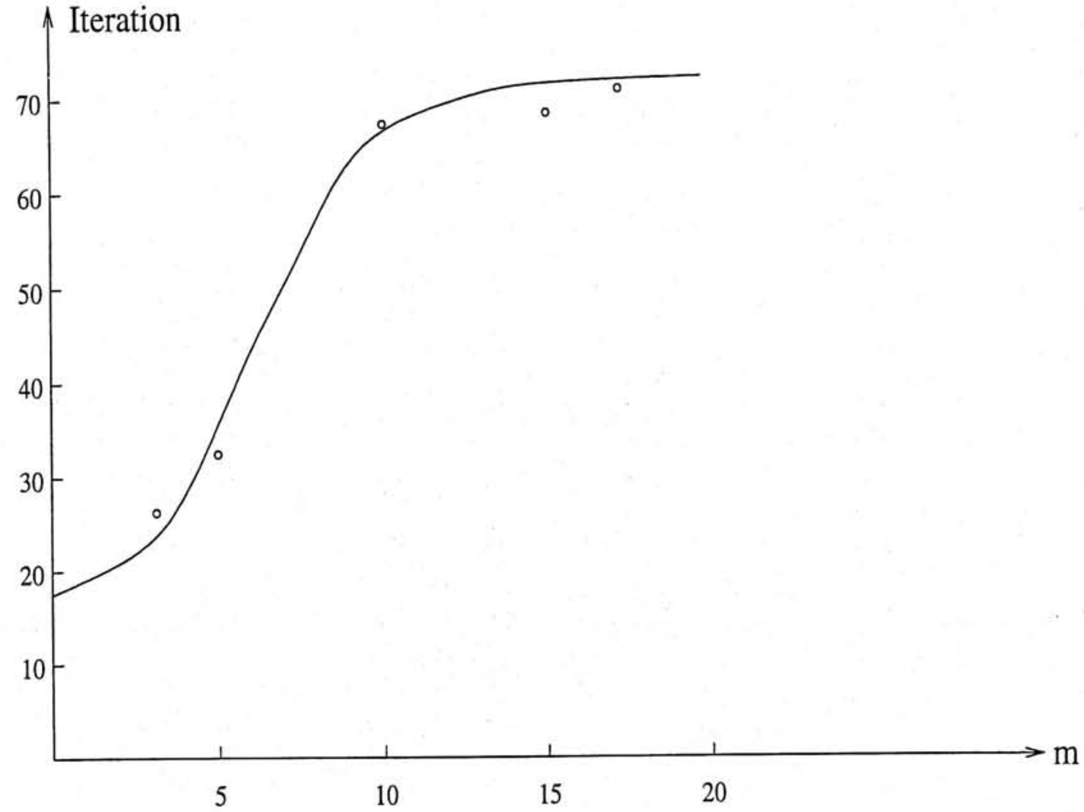

Figure 7.1: The relationship between iteration and m

number and the size of the problem by Figure 7.1.

From Figure 7.1, it seems that the iteration amount increases logarithmically when the size of the problem grows linearly. Further research is still required to check this observation.

Chapter 8

Conclusions and Future Work

A new algorithm for polynomial zero-one programming has been investigated in this thesis. The p-norm surrogate-constraint algorithm is an implicit enumeration method based on Taha's previous work [31][32] and the p-norm surrogate constraint method recently proposed in [22]. Up-to-date, implicit enumeration methods are one of the most efficient ways to solve the polynomial zero-one programming problems due to its flexibility and associability. By powering the implicit enumeration method with the p-norm surrogate constraint method, significant improvements have been made to increase the efficiency of the new algorithm. The modified version of the backtrack scheme proposed in this thesis enhances further the efficiency of the new algorithm via reducing the number in the candidate list for optimality.

The efficiency of this new algorithm is achieved primarily based on the derived single-constraint formulation. This feature will be most evident if the original problem formulation involves all the $2^{n'}$ - 1 terms. In other situations, there exists a trade-off between reducing the number of constraints and increasing the number of product terms. Explicit evaluation of this trade-off will be carried out in the near future to assess the degree of success of this new algorithm. From our computational experience, the saving in the computation for the standard formulation (5.2) with secondary constraints (1.2) is tremendous. However, great computation efforts in expansion and simplification are required to performing the p-norm surrogate constraint method to convert an original form to the standard form, especially when p is large.

The new algorithm can be used to solve not only the polynomial zero-one programming problem, but also the linear zero-one programming problem. An application to the set covering problem is demonstrated in this thesis. A rough observation indicates that the computational amount seems to increase logarithmically when the size of the problem grows linearly. One feature of the p-norm surrogate-constraint in the set covering problem is that the objective function of the set covering problem remains linear in the procedure of transformation and it often has much less terms than the terms in the surrogate constraint. This property seems to help in the implementation of the new algorithm.

In summary, the new algorithm seems promising. Of course, more numerical tests are needed to check its efficiency in solving the polynomial zero-one programming problems and more work is needed to evaluate its average performance against the existing ones in the literature.

Bibliography

[1] Balas, E., "An additive algorithm for solving linear programs with zero-one variables," *Operations Research,* Vol. 13, No. 4, pp.517-546 (1965).

[2] Balas, E. and Carrera, M. C., "A dynamic subgradient-based branch-and-bound procedure for set covering problem," *Operation Research,* Vol. 44, pp.875-890 (1996).

[3] Balas, E., "A class of location, distribution and scheduling problem: modeling and solution methods," *Proceedings of the Chinese-U.S. Symposium on System Analysis,* J. Wiley and Sons (1983).

[4] Beasley, J. E., "An algorithm for set covering problems," *European Journal of Operational Research,* Vol. 31, pp. 85-93 (1987).

[5] Beasley, J. E., " A Lagrangian heuristic for set covering problem," *Naval Research Logistics* Vol. 37, pp.151-164 (1990).

[6] Beasley, J. E. and Jornsten, K., "Enhancing an algorithm for the set covering problem," *European Journal of Operational Research,* Vol. 99, pp.293-300 (1992).

[7] Caprara, A., Fischetti, M., and Toth, P., "A heuristic method for the set covering problem," *Operations Research,* (1995).

[8] Ceria, S., Nobili, P. and Sassano, A., "Set covering problem," *Annotated Bibliographies in Combinatorial Optimization,* J. Wiley and Sons, pp.415-428 (1997).

[9] Ceria, S., Nobili, P. and Sassano, A., "A Lagrangian-based heuristic for large-scale set covering problem," *Mathematical Programming,* (1995).

[10] Chrissis, J. W., "The solution of nonlinear pseudo-boolean optimization problems subject to linear constraints," Doctoral Dissertation, Virginia Polytechnic Institute and State University, Blacksburg, Virginia (1980).

[11] Curry, G. and Skeith, R. W., "A dynamic programming algorithm for facility location and allocation," *AIIE Transactions,* Vol. 1, pp. 133-138 (1969).

[12] Haddadi, S., "Simple Lagrangian heuristic for the set covering problem," *European Journal of Operational Research,* Vol. 97, pp. 200-204 (1997).

[13] Hammer, P. L. and Rudeanu, S., "Pseudo-boolean programming," *Operations Research,* Vol.17, pp.233-261 (1969).

[14] Garey, M. R. and Johnson, D. S., "Computers and intractability: A guide to the theory of NP-completeness," Freeman (1979).

[15] Geoffrion, A., "Integer programming by implicit enumeration and Balas' method," *SIAM Review,* Vol. 9, pp.178-190 (1967).

[16] Glover, F. and Woolsey, R. E., "Further reduction of zero-one polynomial

programming problems to zero-one linear programming Problems," *Operations Research,* Vol.21, No. 1, pp.156-161 (1973).

[17] Glover, F. and Woolsey, R. E., "Converting the 0-1 polynomial programming problem to a 0-1 linear program," *Operations Research,* Vol. 22, No. 1, pp.141-161 (1974).

[18] Granot, D., Granot, F. and Vaessen, W., "An accelerated covering relaxation algorithm for solving 0-1 positive polynomial programs," Working Paper 718, Faculty of Commerce and Business Administration, U.B.C., Vancouver, B.C., March 1980.

[19] Granot, D., Granot, F. and Kallberg, J., "Covering relaxation for positive 0-1 polynomial programs," *Management Science,* Vol. 25, pp.760-772 (1977).

[20] Granot, D., Granot, F. and Vaessen, W., "An accelerated covering relaxation algorithm for solving 0-1 positive polynomial programs," *Mathematical Programming,* Vol. 22, No. 3, pp. 350-357 (1982).

[21] Lawler, E. L. and Bell, M. D., "A method for solving discrete optimization problems," *Operations Research,* Vol. 14, No. 6, pp. 1089-1112 (1966).

[22] Li, D., "Success guarantee of dual search in integer programming: p-th power Lagrangian method," To appear in *Operations Research Letters* (1999).

[23] Lorena, L. A. N. and Lopes, F. B., "A surrogate heuristic for set covering problem," *European Journal of Operational Research,* Vol. 79, pp.138-150 (1994).

[24] Lund, C. and Yannakakis, M., "On the hardness of approximating minimization problems," *Journal of the ACM,* Vol. 41, pp. 960-981 (1994).

[25] Mao, J. C. and Wallingford, B. A., "An extension of Lawler and Bell's method of discrete optimization with examples from capital budgeting," *Management Science,* Vol. 15, pp. 481 (1968).

[26] Nobili, P. and Sassno, A., "A separation routine for the set covering polytope," *Integer Programming and Combinatorial Optimization,* (1992).

[27] Peterson, D. E. and Laughhunn, D., "Capital expenditure programming and some alternative approaches to risk," *Management Science,* Vol. 17, pp.320-336 (1971).

[28] Pritsker, A. and Watters, L. J., "Mathematical formulation: a problem in design," The Rand Corporation, pp.3790 (1968).

[29] Pritsker, A. and Wolfe, P., "Multiproject scheduling with limited resources: A zero-one programming approach," *Management Science,* Vol. 16, pp. 93-108 (1969).

[30] Snyder, W. S. and Chrissis, J. W., "A hybrid algorithm for solving polynomial zero-one mathematical programming problem," *IIE Transactions,* Vol. 22, No. 2, pp. 161-167 (1990).

[31] Taha, H. A., "A Balasian-based algorithm for zero-one polynomial programming," *Management Science,* Vol. 18, No. 6, pp.328-343 (1972a).

[32] Taha, H. A., "Further improvements in the polynomial zero-one algorithm," *Management Science,* Vol. 19, No. 2, pp. B226-227 (1972b).

[33] Taha, H. A., "Sequencing by implicit ranking and zero-one polynomial programming," *Research Report No. 70-3,* Department of Industrial Engineering, University of Arkansas (1970).

[34] Vaessen, W., "Covering relaxation methods for solving the zero-one positive polynomial programming problems," M.Sc. thesis, Computer Science Department, U.B.C., Vancouver, B.C. (1980).

[35] Watters, L. J., "R&D project selection: interdependent and multiperiod probabilistic budget constraints," Doctoral Dissertation, Arizona State University, Tempe (1966).

[36] Watters, L. J., "Reduction of integer polynomial problems to zero-one linear programming problems," *Operations Research,* Vol. 15, No. 6, pp. 1171-1174 (1981).