Global Optimization of Monotonic Programs: Applications in Polynomial and Stochastic Programming

A Thesis
Presented to
The Academic Faculty

by

Myun-Seok Cheon

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

School of Industrial and Systems Engineering
Georgia Institute of Technology
May 2005

Global Optimization of Monotonic Programs: Applications in Polynomial and Stochastic Programming

To my wife and family,

for their love and patience.

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude to my two advisors, Dr. Faiz Al-Khayyal and Dr. Shabbir Ahmed for their guidance and encouragements throughout the entire course of my research. Without their excellent insights and invaluable suggestions, I could not be able to complete this thesis.

I am grateful to Dr. Earl Barnes, Dr. Alex Shapiro and Dr. Matthew Realf for serving as members of my committee and their valuable comments and suggestions.

Finally, I am deeply thankful to my parents, Bong-Nam Cheon and Sung-Hang Lee for their support and encouragement. I owe special thanks to my wife, Jin-Ah Lim, and my princess and prince, Esther and Ethan. Their support, patience, and love enabled me to complete this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Monotonic optimization consists of minimizing or maximizing a monotonic objective function over a set of constraints defined by monotonic functions. Many optimization problems in economics and engineering often have monotonicity while lacking other useful properties, such as convexity. This thesis is concerned with the development and application of global optimization algorithms for monotonic optimization problems.

First, we propose enhancements to an existing outer-approximation algorithm — called the Polyblock Algorithm — for monotonic optimization problems. The enhancements are shown to significantly improve the computational performance of the algorithm while retaining the convergence properties. Next, we develop a generic branch-and-bound algorithm for monotonic optimization problems. A computational study is carried out for comparing the performance of the Polyblock Algorithm and variants of the proposed branch-and-bound scheme on a family of separable polynomial programming problems. Finally, we study an important class of monotonic optimization problems — probabilistically constrained linear programs. We develop a branch-and-bound algorithm that searches for a global solution to the problem. The basic algorithm is enhanced by domain reduction and cutting plane strategies to reduce the size of the partitions and hence tighten bounds. The proposed branch-reduce-cut algorithm exploits the monotonicity properties inherent in the problem, and requires the solution of only linear programming subproblems. We provide convergence proofs for the algorithm. Some illustrative numerical results involving problems with discrete distributions are presented.

CHAPTER I

INTRODUCTION

The rapid growth of global optimization is observed in recent days for problems arising in science and engineering. Such problems include finance, allocation and location problems, operations research, statistics, structural optimization, engineering design, network and transportation problems, chip design and database problems, nuclear and mechanical design, chemical engineering design and control, and molecular biology [16]. Many important optimization problems contain nonconvexity due to either a nonconvex objective function or a nonconvex feasible region or both. One may want or need to have a best (global optimal) solution rather than a locally best (local optimal) solution for a nonconvex problem. According to Neumaier [15], a global optimal solution needs to be found in applications like hard feasibility problems, computer-assisted proofs, safety verification problems, problems in chemistry and semi-infinite programming. Often, local optimization techniques for these problems usually do not return any useful information; e.g., in robot arm design, a local solution gets stuck in local minimizers of the merit function, and does not provide feasible points. For another example, a local optimal solution, in safety verification problems, can be, in the worst case, a severe underestimation of the true risk.

The methods for global optimization can be categorized into deterministic and stochastic approaches [16]. In deterministic approaches, by exploiting the characteristics of any mathematical structure, we generate a deterministic sequence of points that converge to a global optimal solution. In stochastic approaches, the search is performed randomly and ensures that a global optimal solution will be found with a high probability. Deterministic approaches are suitable for d.c. optimization (differences of convex functions or sets), monotonic programs (monotone objective over a feasible set defined by monotone functions with inequalities), quadratic and polynomial programming and discrete problems. For stochastic approaches, there are two-phase methods, multistart methods and their traditional variants (e.g., random linkage and metaheuristics), simulated annealing, tabu search, and genetic algorithms.

Monotonic programs consist of minimizing or maximizing a monotone objective function over a feasible set defined by monotone inequalities. Many mathematical programming problems have monotonicity rather than convexity or concavity in either their objective or constraints (e.g., multiplicative programming, fractional programming, indefinite quadratic programming, polynomial programming, Lipschitz optimization, optimization under network constraints, Fekete points problem and Lennard-Jones potential energy function [32].) For monotonic programming problems, an outer approximation algorithm, called the Polyblock Algorithm [32], and pth power convexification and concavification schemes [12] have been developed. According to Tuy [32], monotonic programming approaches have been demonstrated to be efficient for solving multiplicative programming problems while alternative methods hardly handle them. The efficiency of the monotonic programming approaches has been reported with computational results on various classes of global optimization problems, such as linear or polynomial fractional programming [17, 37], and discrete nonlinear programming [36].

In Chapter 2, we exploit rigorous properties of monotonic programs and review the *Polyblock* algorithm proposed by Tuy [34]. We propose enhancements in order to improve the efficiency of the algorithm. On some problems, we have observed jamming of the Polyblock Algorithm (i.e., the algorithm requires many iterations to check certain regions or points). By identifying properties of vertices defining polyblocks, we propose a more efficient way to manage vertices. The efficiency of our modification is demonstrated by computational experiments. Furthermore, we implement a branch-and-bound based algorithm. Because of monotonicity, it is possible to make an efficient domain reduction scheme and to construct optimality cuts. Similar ideas were independently developed by Tuy et al. [35] for the more general difference of monotone case without implementation and computational testing.

In order to compare the performance of algorithms proposed in Chapter 2, we consider a class of polynomial programming problems in Chapter 3. Polynomial programming problems are widely used in engineering and science applications. Polynomial programming problems are either monotonic programming problems or difference of monotonic programming problems. We show how to transform a general polynomial programming problem into a monotonic programming problem. We enhance the algorithms developed in Chapter 2 by providing linear convex relaxations. We report extensive computational results on the performance of these proposed enhanced algorithms.

Stochastic programming problems are optimization problems involving uncertainly. Probabilistically constrained linear programming (PCLP) is an instance of a stochastic program that deals with reliability or satisfaction of conditions (e.g., to satisfy demand or service level) under uncertainty. The most well-known applications are in areas of finance, economic planning, inventory control, reservoir management and telecommunication reliability.

In Chapter 4, we consider probabilistically constrained linear programs with general distributions for the uncertain parameters. We develop a branch-and-bound algorithm that searches for a global solution to this problem by successively partitioning the nonconvex feasible region and by using bounds on the objective function to fathom inferior partitions. This basic algorithm is enhanced by domain reduction and cutting plane strategies to reduce the size of the partitions and hence tighten bounds. The proposed *branch-reduce-cut* algorithm exploits the monotonicity properties inherent in the problem, and requires solving linear programming subproblems. We provide convergence proofs for the algorithm. Some illustrative numerical results involving problems with discrete distributions are presented.

CHAPTER II

MONOTONIC PROGRAMMING: STRUCTURE AND ALGORITHMS

2.1 Introduction

An optimization problem that consists of a monotone objective and the feasible set, defined by monotone functions, is called a *monotonic program.* Many mathematical programming functions that arise in economics and engineering applications either have monotonic properties or can be represented as the difference of monotonic functions [32].

Mathematical programs with monotonic properties have been studied in the literature. Li et al. [12] proposed convexification and concavification schemes that transform a monotone function into either a concave or a convex function by using a p^{th} power transformation. It follows that a monotonic programming problem can be transformed into either a convex minimization problem over a convex set or a d.c. programming problem. Consider a twice differentiable monotonic increasing function $h : \mathbb{R}^n \mapsto \mathbb{R}$. For any $x = [x_1, \ldots, x_n]^T \in \mathbb{R}^n_+$ and $p > 0$, $x^{1/p}$ is defined as follows:

$$x^{1/p} = [x_1^{1/p}, \ldots, x_n^{1/p}]^T.$$

The p^{th} power concavification transformation of h is as follows:

$$\phi_h(x) = -[h(x^{1/p})]^{-p}.$$

They show that there exists $p_1 > 0$ such that ϕ_h is a concave function for $p > p_1$ under the condition that there exist two positive numbers $\epsilon_0 > 0$ and $\epsilon_1 > 0$ such that

$$h(x) \geq \epsilon_0, \qquad \forall\, x \in X, \tag{2.1.1}$$

$$\frac{\partial h}{\partial x_j} \geq \epsilon_1, \qquad \forall\, x \in X \text{ and } \forall j \in \{1, \ldots, n\}. \tag{2.1.2}$$

Similarly, we can transform a monotonic decreasing function into a convex function. After these transformations, a monotonic programming problem can be reformulated as a convex or a d.c. programming problem. Li et al. [12] also proposed a method that transforms a non-convex general problem into a monotonic optimization problem. Consider a twice differentiable function h on X satisfying $h(x) \geq \epsilon_0 > 0$ for all $x \in X$. Let l_j and u_j be the lower and upper bounds of a variable x_j. The transformed monotonic function $w_h^+(x)$ of h is as follows:

$$w_h^+(x) = h(x) \exp\left(q\left(\sum_{j=1}^{n} x_j - \rho\right)\right),$$

where q and ρ are parameters with $q > 0$ and $\sum_{j=1}^{n} l_j \leq \rho \leq \sum_{j=1}^{n} u_j$. Then, there exists a finite $q_1 > 0$ such that $w_h^+(x)$ is a strictly increasing function on $X_1(\rho) = \{x \in X : \sum_{j=1}^{n} x_j \geq \rho\}$ for $q \geq q_1$. By using these two reformulation techniques, some non-convex problems can be transformed into convex problems or d.c. problems.

Tuy [32] has studied the properties of monotonic programming problems and developed a general framework for solving bounded monotonic programming problems to global optimality. The algorithm is based upon successively outer approximating the feasible region of a monotonic programming problem using a nested sequence of polyblocks, or unions of hyperrectangles, and exploiting the fact that the minima of a non-decreasing function over a polyblcok is at an extreme point of the polyblock. While the p^{th}-power convexification and concavification schemes [12] require continuity and twice differentiable properties of the function, the Polyblock Algorithm does not require any other properties except monotonicity of functions. The efficiency of the Polyblock Algorithm has been demonstrated in various applications such as multiplicative programming, linear and polynomial fractional programming [17, 37], and discrete nonlinear programming [36].

In this chapter, the characteristics of monotonic programs are exploited and the *Polyblock Algorithm* [32] is revised with enhancements. The effectiveness of our enhancements are demonstrated by computational experiments. We also propose a branch-and-bound algorithm for monotonic programming problems. Because of monotonicity, it is easy to reduce the range of the variables and to determine bounds for a given region.

In the original work, Tuy [32] mentioned that a branch-and-bound scheme using the Polyblock Algorithm for bounding can be devised for monotonic programs but gave no details. In this thesis, we developed and implemented such a scheme. Computational experiments suggested the need for enhancements to improve the performance of the approach. Independently, Tuy et al. [35] describe a branch-and-bound method and propose some enhancements; however, the algorithm was not implemented.

2.2 Characteristics of Monotonic Programs

The general statement of a bounded monotonic programming problem is as follows:

$$\begin{array}{lll} \text{(MO):} & \min & f(x) \\ & \text{s.t.} & h_j(x) \geq 0 \quad \text{for } j = 1, \ldots, m_H \\ & & g_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m_G \\ & & x \in [a, b] \subset \mathbb{R}^n, \end{array}$$

where $[a, b]$ denotes a hyperrectangle with least element a and greatest element b, the function $f : \mathbb{R}^n \mapsto \mathbb{R}$ is monotonic non-decreasing, and the functions $g_i : \mathbb{R}^n \mapsto \mathbb{R}$ and $h_j : \mathbb{R}^n \mapsto \mathbb{R}$ are monotonic non-decreasing for $i = 1, \ldots, m_G$ and $j = 1, \ldots, m_H$. In order to ensure that the feasible set is closed, we assume that function g_i is lower semi-continuous for all i and function h_j is upper semi-continuous for all j.

Consider the sets $G := \{x \in \mathbb{R}^n : g_i(x) \leq 0 \quad \text{for } i = 1, \ldots, m_G\}$ and $H := \{x \in \mathbb{R}^n : h_i(x) \geq 0 \quad \text{for } i = 1, \ldots, m_H\}$. The feasible region of (MO) is then $G \cap H$. The sets G and H have the following properties:

$$\text{for any } x, y \in \mathbb{R}^n : x \in G \text{ and } y \leq x \Rightarrow y \in G, \tag{2.2.1}$$

$$\text{for any } x, y \in \mathbb{R}^n : x \in H \text{ and } y \geq x \Rightarrow y \in H. \tag{2.2.2}$$

A set that satisfies property (2.2.1) is known as a *normal* set, and the set H is called a *reverse normal* set.

Example 2.1. *Consider the following monotonic programming problem in two dimensions:*

$$\min \quad x_1 + x_2 \tag{2.2.3}$$

$$s.t. \quad (x_1 - 3)^3 + 9(x_2 - 3) \geq 0 \tag{2.2.4}$$

$$5x_1 + 6x_2 - 36 \leq 0 \tag{2.2.5}$$

$$(x_1, x_2) \in [0, 6]^2. \tag{2.2.6}$$

The objective function in (2.2.3) and the functions in (2.2.4) through (2.2.6) that define the feasible set are monotone continuous functions. The feasible region of Example 2.1 is illustrated in Figure 1. The dark shaded area in Figure 1 is the feasible region of Example 2.1 and it is a nonconvex and disconnected set.

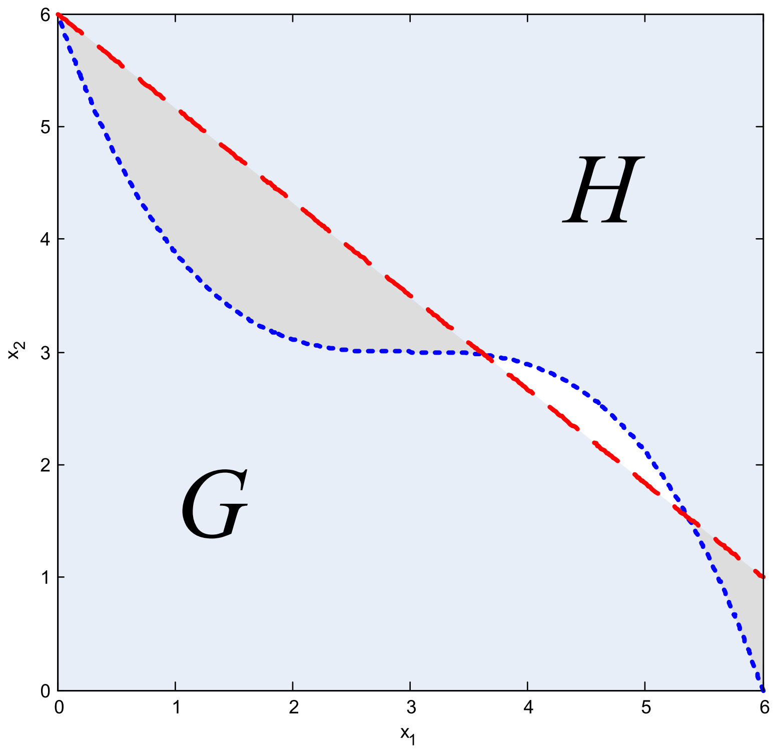


Figure 1: Feasible region of Example 2.1

In Example 2.1, the *normal* set of the problem is defined by the constraint (2.2.5), which is represented by the region below the *dashed* line in Figure 1. The constraint (2.2.4) defines the *reverse* set H given by the region above the *dotted* line in Figure 1.

We summarize the properties of (MO) from the general results in [32, 35]. For completeness of exposition, we also include a proof.

Proposition 2.1. *For a given* (MO) *and* $x^L, x^U \in \mathbb{R}^n$ *such that* $x^L \leq x^U$*,*

(i) if there exists $x \in G \cap H \cap [x^L,\ x^U]$*, then* $x^L \in G$ *and* $x^U \in H$*;*

(ii) if $x^L \in H$*, then either* x^L *is an optimal solution over* $G \cap H \cap [x^L,\ x^U]$ *or* $G \cap H \cap [x^L,\ x^U] = \emptyset$*;*

(iii) if $\widehat{x} \in [x^L,\ x^U]$ *is on the relative boundary of* H*, then there cannot be a solution that is better (i.e., feasible with a smaller objective value) in the set* $Q := \{x : x^L \leq x < \widehat{x}\}$ *and* $Q' := \{x : \widehat{x} \leq x \leq x^U\}$*;*

(iv) if $G \cap H \cap [x^L, x^U] \neq \emptyset$*, then there exists an optimal solution of (MO) that lies on the relative boundary of* H*.*

Proof. To prove part (i), we must show that if $x^L \notin G$ or $x^U \notin H$, then $G \cap H \cap [x^L,\ x^H] = \emptyset$. If $x^L \notin G$, there is no point such that $x \in G$ because of the property of the normal set G in (2.2.1) and $x^L \leq x$. The case $x^U \notin H$ is analogous to the case $x^L \notin G$.

To prove part (ii), consider $x^L \in H \setminus G$. By part (i), there is no feasible solution in $x \in [x^L,\ x^U]$. Consider $x^L \in H \cap G$. By the monotonicity of the objective function, $f(x^L) \leq f(x)$ for $x \in [x^L,\ x^U]$. Thus, x^L is an optimal solution.

To prove part (iii), consider the set Q. Let $\widehat{x}$ be a boundary point of H. Suppose that there is $x \in H \cap Q$. That is, $x < \widehat{x}$. Then, this contradicts that $\widehat{x}$ is a boundary point. Therefore, $G \cap H \cap Q = \emptyset$. For the set Q', by part (ii), either $G \cap H \cap Q' = \emptyset$ or $\widehat{x}$ is an optimal solution over $G \cap H \cap Q'$.

To prove part (iv), consider an optimal solution $y \in ri(H)$. Since the objective function f of (MO) is a monotonic non-decreasing function, i.e., $f(x) \leq f(y)$ if $x \leq y$, for any $y \in ri(H)$ then there exists a point $x \in H$ such that $f(x) \leq f(y)$ and $x \leq y$. This implies that an optimal solution of (MO) lies on the boundary of H. $\square$

By using the properties in Proposition 2.1, Tuy [32, 35] proposed the *Polyblock Algorithm* for (MO).

2.3 Polyblock Algorithm

For $a, b \in \mathbb{R}^n$ and $a < b$, define $[a, b] = \prod_{j=1}^{n}[a_j, b_j]$. Let T be a finite set of points in $[a, b]$. The set $P = \cup_{v \in T}[v, b]$ is called a *polyblock* and the set T is called the *vertex set* of P. A polyblock is clearly a reverse normal set, and the intersection of polyblocks is also a polyblock [32, 35]. A vertex $v \in T$ is *proper* if there is no $v' \in T$ such that $v' \neq v$ and $v' \leq v$, and *improper* otherwise. Improper vertices can be deleted without changing the polyblock, so a polyblock is completely determined by its proper vertices.

The Polyblock Algorithm inductively generates a nested sequence of polyblocks that outer approximate the feasible set:

$$[a, b] := P_0 \supset P_1 \supset P_2 \supset \cdots \supset P_k \supset \cdots \supset G \cap H \cap [a, b]$$

such that

$$\min\{f(x) : x \in P_k\} \nearrow \min\{f(x) : x \in G \cap H\}.$$

It can be shown from Proposition 2.1 (ii) that

$$\min\{f(x) : x \in P_k\} \Leftrightarrow \min\{f(x) : x \in T_k\},$$

where T_k is the vertex set of P_k. The Polyblock Algorithm then proceeds as follows. At each iteration, a vertex $v^k \in T_k \cap G$ is chosen such that $f(v^k)$ is minimal among all vertices in $T_k \cap G$. If no such vertex exists, the algorithm terminates with the best known solution so far being the optimal solution of (MO), or it is resolved that the problem is infeasible. If $v^k \in H$ then it is a global optimal solution. Otherwise, a point $x^k \in \partial(H)$ is derived such that the set $\{x : a \leq x < x^k\}$ contains no feasible solution; hence, this set can be removed from $[v^k, b]$ without losing any feasible solutions. In order to obtain a boundary point $x^k \in \partial(H) \cap [v^k, b]$ for $v^K \in G \setminus H$ and $b \in H$, we solve the following problem (2.3.1):

$$\pi_H(v^k, b) = v^k + \mu^*(v^k - b), \text{where } \mu^* = \min\{\mu \in [0\ 1] : v^k + \mu(v^k - b) \in H\}. \qquad (2.3.1)$$

Since we assume that the reverse normal set H is defined by upper semi-continuous functions, the reverse normal set H is closed. Thus, problem (2.3.1) for obtaining μ^* has an optimal solution. Problem (2.3.1) can be solved by using bisection search. If $x^k \in G$ then it is a feasible solution and can be used to update the current best feasible solution. After removing $\{x : a \leq x < x^k\}$ from P_k, a new polyblock $P_{k+1} \subset P_k$ is constructed which excludes v^k but still contains all global solutions. Under mild conditions, it has been shown [32, 35] that as $k \to \infty$, the sequence x^k converges to a global optimal solution of the problem. The detailed procedure of the Polyblock Algorithm [35] is shown in Algorithm 1.

Algorithm 1 Polyblock Algorithm proposed in [35]

```
**initialization:** Select $\epsilon \geq 0$ (tolerance). Let $T_1$ be the proper vertex set of an initial polyblock $P_1 \supset G \cap H$ (for instance, $T_1 = \{a\}$, with $P_1 = [a, b]$). Assign every $v \in T_1$ a number $\beta(v)$ such that $\min\{f(x) : x \in G \cap H \cap [v, b]\} \geq \beta(v) \geq f(v)$. Let $\bar{x}^1$ be the best feasible solution available and $CBV = f(\bar{x}^1)$ (if no feasible solution is available, $CBV = \infty$). Set $k = 1$.
**step k.1:** From $T_k$ remove all $v \in T_k \setminus G$ and all $v$ such that $\beta(v) \geq CBV - \epsilon$. Denote the set of remaining elements again by $T_k$.
**step k.2:** If $T_k = \emptyset$, terminate: if $CBV = \infty$, the problem is infeasible; if $CBV < \infty$, the current best feasible solution $\bar{x}^k$ is accepted as an $\epsilon$-optimal solution.
**step k.3:** If $T_k \neq \emptyset$, select $v^k \in \arg\min\{\beta(v) | v \in T_k\}$.
**if** convex relaxations are available **then**
    Let $w^k$ be an optimal solution of convex relaxation. Reset $\beta(v^k) \leftarrow \max\{f(w^k), \beta(v^k)\}$.
**end if**
let $x^k = \pi_H(v^k, b)$ to be the intersection of the boundary of $H$ with the line segment joining $b$ with $v^k$, where $\pi_H(v^k, b)$ is defined in (2.3.1).
**step k.4:** Determine the new current best value $CBV$ and the new current best feasible solution $\bar{x}^{k+1}$.
**if** $\beta(v^k) \geq CBV - \epsilon$ **then**
    reset $T_k \leftarrow T_k \setminus \{v^k\}$, and goto **step k.2**.
**else**
    goto **step k.5**.
**end if**
**step k.5:** Let $T_{k,*} = \{v \in T_k | \ v < x^k\} \cup v^k$. Compute
        $T'_k = (T_k \setminus T_{k,*}) \cup \{v^{k,i} | v \in T_{k,*}, v_i < x^k_i, i = 1, \ldots, n\}$        (2.3.2)
where $v^{k,i} = v + (x^k_i - v_i)e^i$. Let $T_{k+1}$ be the set obtained from $T'_k$ by removing improper vertices. For every $v \in T_{k+1} \setminus T_k$ set $\beta(v) = \max\{f(v), \beta(v^k)\}$ ($v^k$ is the father of $v$).
**step k.6:** $k \leftarrow k + 1$ and goto **step k.1**.
```

2.3.1 An illustrative example

Here, the steps of the Polyblock Algorithm for solving Example 2.1 are illustrated. An application of the Polyblock Algorithm with a termination tolerance of $\epsilon = 0.1$ requires 17 iterations with the maximum number of vertices considered in any iteration being 6.

Figure 2 shows iterations 1,2,3 and 17 of the Polyblock Algorithm. The algorithm starts out with $P_1 = [(0,0),(6,6)]$ as the initial (poly)block. The minimal vertex chosen is $v^1 = (0,0)$. For this polyblock, we get $x_b^1 = (3,3)$ and the corresponding objective value is 6. The current best feasible solution is found to be $(3,3)$. By removing $[(0,0),(3,3))$ from P_1, a new polyblock $P_2 = [(3,0),(6,6)] \cup [(0,3),(6,6)]$ is generated. In iteration 2, the vertex $(0,3)$ is chosen to be minimal, and $x_b^2 = (1.2307, 3.6154)$ is the corresponding boundary point of H. Polyblock P_3 is constructed by removing $[(0,0),(1.2307,3.6154))$ from P_2. A new polyblock $P_3 = [(3,0),(6,6)] \cup [(0,3.6154),(6,6)] \cup [(1.2307,3),(6,6)]$ is generated. In P_3 a vertex $(3,0)$ has a minimum lower bound among vertices. The line segment $[(3,0),(6,6)]$ intersects the boundary of the reverse normal set at the point $(4.3602, 2.7204)$. After 17 iterations, the point $(1.2307, 3.6154)$ is determined to be the ϵ-global optimal solution with an objective value of 4.8461 for $\epsilon = 0.1$. The line in *Iteration 17* of Figure 2 depicts the objective function contour crossing the best solution so far within our optimality tolerance. All of the vertices that define a polyblock are above the line carrying the optimal solution. This implies that we cannot find a better solution than the current best solution.

2.3.2 Enhancement

Based on our computational experiments, we found that a significant portion of the computational effort of the Polyblock Algorithm is in finding proper vertices. Consider two vertices v^1 and v^2 in the vertex set of the algorithm such that $v^1 \leq v^2$. The block defined by the vertex v^1 contains the block defined by the vertex v^2. For this case, we say that vertex v^1 *dominates* vertex v^2 and that vertex v^2 is improper because we can remove it without changing the shape of the polyblock. In order to identify improper vertices in Algorithm 1, all pairs of vertices in T'_k should be compared to each other. Thus, the number of comparisons is $\frac{|T'_k| \times (|T'_k| - 1)}{2}$. Through our enhancement, we identify a smaller vertex set that includes all of the improper vertices and those improper vertices can be identified by comparisons of all pairs within the smaller vertex set. That is, any vertex in this smaller set dominates only members of this set and is not dominated by vertices outside the set. Since the number of comparison is reduced, so is the computational effort.

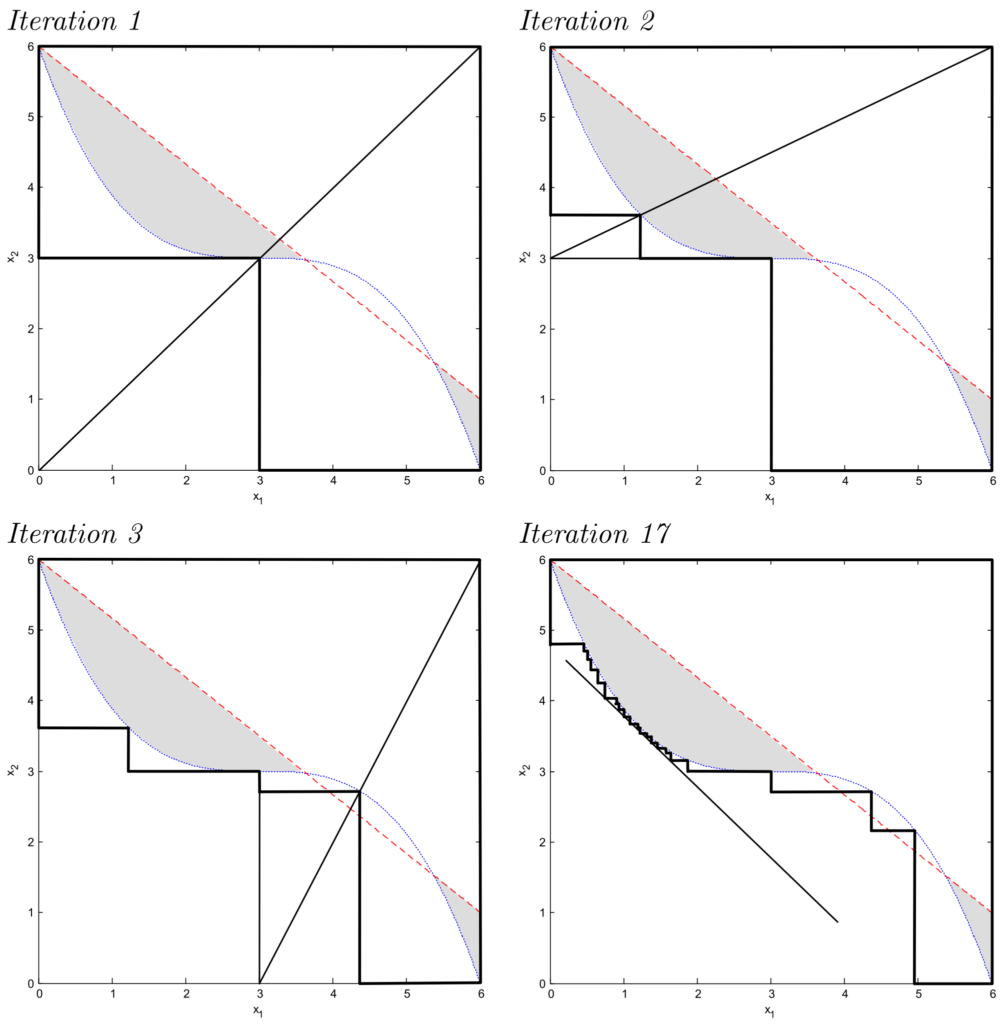


Figure 2: Procedures of the Polyblock Algorithm applied to the example (EX1)

Another computational difficulty associated with the polyblock algoirthm is *jamming* that occurs in the converging subsequence of vertices to a vertex having at least one j^{th} element equal to b_j. That is, when one of the elements in the vertex is close to its upper bound, then one of the successive vertices is almost the same as the previous vertex (i.e., for that element, the distance between the vertex and the corresponding successive vertex is smaller than the gap between the element and its upper bound). To circumvent this behavior, we propose a redefinition of the function $\pi_H(\cdot)$.

2.3.2.1 Eliminating improper vertices

In Algorithm 1, vertices in $T_{k,*} = \{v \in T_k : v < x^k\} \cup \{v^k\}$ will be used in our proposed refinement scheme, where x^k is a boundary point of H. In order to eliminate improper vertices, all of the vertices in T'_k defined by (2.3.2) should be compared with each other. In the worst case, this requires $\frac{|T'_k|(|T'_k|-1)}{2}$ comparisons. In this section, a smaller vertex set will be identified and it will be shown that the comparisons of all pairs in the new vertex set are enough to identify improper vertices. That is, all improper vertices are in the new vertex set and vertices that dominate improper vertices are also in the same vertex set.

An improper vertex ($v^{imp} \in T$) can be defined as following. A vertex v^{imp} is *improper* if there exists a vertex $v' \in T$ such that $v' \leq v^{imp}$. Thus, we can define the improper vertex set ($IMP(T)$) and the proper vertex set ($PS(T)$) as follows

$$\begin{aligned} IMP(T) &= \{v \in T : \exists\ v' \in T, v \geq v'\}, &(2.3.3)\\ PS(T) &= T \setminus IMP(T). &(2.3.4)\end{aligned}$$

In order to specify a vertex set that contains improper vertices, we redefine sets of vertices. Let T^x_k be a vertex set such that $T^x_k = \{v \in T | v \leq x^k\}$ where x^k is a boundary point of H. Let $T^{x=}_k$ be a vertex set such that $T^{x=}_k = T^x_k \setminus T_{k,*}$. Let T^{new}_k be a newly refined vertex set obtained from $T_{k,*}$ such that

$$T^{\text{new}}_k = \{v^{k,i} | v \in T_{k,*}, v_i < x^k_i, i = 1, \ldots, n\}, \tag{2.3.5}$$

where $v^{k,i} = v + (x^k_i - v_i)e^i$ and e^i denotes the ith unit vector.

Proposition 2.2. *The proper vertex set has the following properties:*

(i) $PS(T \cup T') \subseteq T \cup PS(T')$;

(ii) if $T_k = PS(T_k)$, *then* $PS(T_k \setminus T^x_k) = T_k \setminus T^x_k$;

(iii) $\{T_k \setminus T^x_k\} \cap \{T^{x=}_k \cup T^{\text{new}}_k\} = \emptyset$;

(iv) $T_k \setminus T_{k,*} = \{T_k \setminus T^x_k\} \cup T^{x=}_k$.

Proof. Part (i) is trivial;

$$\begin{aligned} T \cup PS(T') &\supseteq PS(T) \cup PS(T') \\ &\supseteq PS(PS(T) \cup PS(T')) \\ &= PS(T \cup T'). \end{aligned}$$

To prove part (ii), $T_k \setminus T_k^x$ is a subset of T_k and T_k is a proper set. This implies that $T_k \setminus T_k^x$ is also a proper set.

To prove part (iii), we know that $v \leq x^k$ for all $v \in T_k^{x=}$ while $v \nleq x^k$ and $v \ngeq x^k$ for all $v \in T_k \setminus T_k^x$ because of the definitions of $T_k^{x=}$ and T_k^x. We must show that $v \leq x^k$ for all $v \in T_k^{\text{new}}$. According to equation (2.3.5), vertices in T_k^{new} are generated from vertices in $T_{k,*}$ by replacing one element of a vertex with one element of x^k. Thus, $v \leq x^k$ for all $v \in T_k^{\text{new}}$. This implies that $\{T_k \setminus T_k^x\} \cap \{T_k^{x=} \cup T_k^{\text{new}}\} = \emptyset$.

To prove part (iv),

$$\begin{aligned} T_k \setminus T_{k,*} &= T_k \setminus (T_k^x \setminus T_k^{x=}) \quad \text{by the definitions of } T_k^{x=} \text{ and } T_{k,*} \subseteq T_k^x \\ &= T_k \setminus (T_k^x \cap (T_k^{x=})^c) \\ &= T_k \cap (T_k^x \cap (T_k^{x=})^c)^c \\ &= (T_k \cap (T_k^x)^c) \cup (T_k \cap T_k^{x=}) \\ &= T_k \setminus T_k^x \cup T_k^{x=}. \end{aligned}$$

□

In Algorithm 1, the proper vertex set is obtained from $T'_k = (T_k \setminus T_{k,*}) \cup T_k^{\text{new}}$ by comparing each pair of two vertices in the set T'_k. We will show that there do not exist $v \in T_k^{x=} \cup T_k^{\text{new}}$ and $v' \in T_k \setminus T_k^x$ such that either $v \leq v'$ or $v \geq v'$. Since the set $T_k \setminus T_k^x$ is a proper set by Proposition 2.2 (ii), it follows that we only need to compare vertices in the set $T_k^{x=} \cup T_k^{\text{new}}$ in order to obtain proper vertices.

Proposition 2.3. $PS(T'_k) = T_k \setminus T_k^x \cup PS(T_k^{x=} \cup T_k^{new})$

Proof. By the procedure of the algorithm, we may assume that T_k is a proper vertex set.

We must show that

$$\begin{array}{llll} (i) & PS(T'_k) & \subseteq & T_k \setminus T_k^x \cup PS(T_k^{x=} \cup T_k^{\text{new}}), \quad \text{and} \\ (ii) & PS(T'_k) & \supseteq & T_k \setminus T_k^x \cup PS(T_k^{x=} \cup T_k^{\text{new}}). \end{array}$$

By Proposition 2.2 (iv), we can rewrite T'_k as follows;

$$T'_k = (T_k \setminus T_{k,*}) \cup T_k^{\text{new}} = (T_k \setminus T_k^x) \cup (T_k^{x=} \cup T_k^{\text{new}}).$$

Part (i) is trivial and is proven by Proposition 2.2 (iv).

To prove part (ii), suppose that there exists v such that $v \notin PS((T_k \setminus T_k^x) \cup (T_k^{x=} \cup T_k^{\text{new}}))$ and $v \in \{(T_k \setminus T_k^x) \cup PS(T_k^{x=} \cup T_k^{\text{new}})\}$. By Proposition 2.2.$(iii)$, v is an element of either the set $T_k \setminus T_k^x$ or $PS(T_k^{x=} \cup T_k^{\text{new}})$. If v is an element of $T_k \setminus T_k^x$, then there exists v' such that $v \geq v' \in (T_k \setminus T_k^x) \cup (T_k^{x=} \cup T_k^{\text{new}})$. But v' cannot be an element of $T_k \setminus T_k^x$ because T_k is a proper vertex set and $T_k \setminus T_k^x$ is also proper. If v' is in $T_k^{x=}$, then T_k is not a proper vertex set because both v and v' would be in T_k. This implies that v' is in T_k^{new}. Since v' is generated from some vertex $z \in T_{k,*} \subset T_k$, we know that $v' \geq z$. Furthermore, $v \geq v' \geq z$ and $v, z \in T_k$. This contradicts the assumption that T_k is a proper set.

Consider the other case that v is an element of $PS(T_k^{x=} \cup T_k^{new})$, then there exists v' such that $v \geq v' \in T_k \setminus T_k^x$ because there is no vertex v' such that $v \geq v' \in T_k^{x=} \cup T_k^{new}$ by the definition of $PS(T_k^{x=} \cup T_k^{new})$. Since $v \in PS(T_k^{x=} \cup T_k^{new})$, then $v \leq x^k$. This implies that $v' \leq v \leq x^k$. By the definition of T_k^x, the vertex v' is an element of T_k^x. This contradicts $v' \in T_k \setminus T_k^x$. □

In this way, we are able to reduce the size of a vertex set that contains improper vertices. It follows that the computational effort in identifying improper vertices decreases. In order to identify improper vertices from m vertices, we need at most $\frac{m(m-1)}{2}$ vector comparisons and each comparison requires at most n scalar comparisons for an n-dimensional problem. According to Proposition 2.3, we remove vertices belonging to the vertex set $T_k \setminus T_k^x$ from our comparisons at iteration k. Thus, we avoid approximately $\frac{|T_k \setminus T_k^x|^2}{2} \times n$ comparisons. We make this procedure more efficient with the following Proposition 2.4.

Proposition 2.4. *If v is an improper vertex and $v \in T'_k$, then $v \in T_k^{\text{new}}$ and $v \notin T_k^{x=} \setminus T_k^{\text{new}}$.*

Proof. By Proposition 2.3, an improper vertex can only be an element of the set $T_k^{x=} \cup T_k^{\text{new}}$. We may assume that T_k is a proper vertex set. Suppose that $v \in T_k^{x=}$ and v is an improper vertex. Then, there exists a vertex v' such that $v \geq v'$ and $v' \in T_k^{x=} \cup T_k^{\text{new}}$. We consider two cases; one is $v' \in T_k^{x=}$ and the other is $v' \in T_k^{\text{new}}$. For the first case, since $v, v' \in T_k$, it contradicts the assumption that T_k is a proper vertex set. In the second case, there exists the parent vertex v'_p of the vertex v' because v' is an element of T_k^{new} and v'_p is also an element of T_k. Thus, $v'_p \leq v' \leq v$ and $v'_p, v \in T_k$. This again contradicts the assumption that T_k is a proper vertex set. □

According to Proposition 2.4, an efficient procedure for finding improper vertices is as follows: first, we identify and remove improper vertices from the vertex set T_k^{new}, then we compare each remaining vertex in T_k^{new} with each vertex in $T_k^{x=}$. By this proposed procedure, we can save $|T_k^{x=}|^2$ vector comparisons.

2.3.2.2 Preventing jamming

First, we illustrate the phenomenon of jamming with an example.

Example 2.2. *We consider the following problem:*

$$\begin{aligned} \min \quad & 2x_1 + 2x_2 + x_3 \\ s.t. \quad & x_1 + x_2 + x_3 \geq 7 \\ & 1 \leq x_j \leq 3 \quad \forall\, j = 1, 2, 3. \end{aligned}$$

We can depict the feasible region of Example 2.2 as Figure 3.

The optimal objective value of Example 2.2 is 11. We consider a sequence of vertices that converges from the vertex (1, 1, 1) to the vertex (3, 1, 1). In the first iteration, we have the boundary point $(\frac{7}{3}, \frac{7}{3}, \frac{7}{3})$ on the line segment [(1, 1, 1), (3, 3, 3)] and the corresponding objective value is $\frac{35}{3}$. By removing $[(1,1,1), (\frac{7}{3}, \frac{7}{3}, \frac{7}{3})]$, we have the three new vertices $(\frac{7}{3}, 1, 1)$, $(1, \frac{7}{3}, 1)$ and $(1, 1, \frac{7}{3})$, and the corresponding objective values are $\frac{23}{3}$, $\frac{23}{3}$ and $\frac{19}{3}$, respectively. Since the current best objective value is $\frac{35}{3}$, we cannot fathom any vertex in the vertex set. After a few iterations, we will refine the region corresponding to the vertex $(\frac{7}{3}, 1, 1)$. The corresponding boundary point is $(\frac{19}{7}, \frac{15}{7}, \frac{15}{7})$. By removing

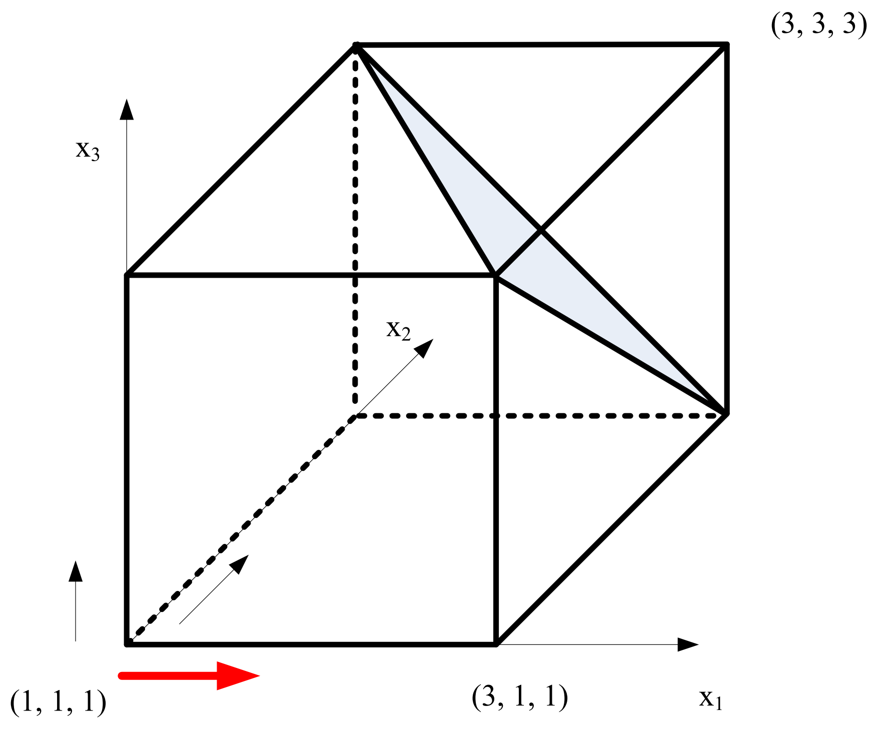


Figure 3: An illustrative example of jamming

$[(\frac{7}{3},1,1),(\frac{19}{7},\frac{15}{7},\frac{15}{7})]$, we have new vertices $(\frac{19}{7},1,1)$, $(\frac{7}{3},\frac{15}{7},1)$ and $(\frac{7}{3},1,\frac{15}{7})$. We consider vertices that lie on the line segment [(1,1,1), (3,1,1)]. Since the optimal value is 11 and the objective value corresponding to the vertex (3, 1, 1) is 9, no vertex on the line segment [(1,1,1), (3,1,1)] is fathomed.

Table 1 summarizes an instance of *jamming* that occurs for a subsequence from the vertex (1, 1, 1) to the vertex (3, 1, 1). In our implementation, we allow for numerical errors. With a numerical error tolerance of 1E-6, twenty iterations are required. In the case of a 1E-16 tolerance, fifty-two iterations are required. For the given problem, there are three subsequences converging to points (3, 1, 1), (1, 3, 1) and (1, 1, 3). Any vertex in those subsequences cannot be fathomed. That is, it requires $60(=20\times3)$ iterations with tolerance 1E-6 or $156(=52\times3)$ iterations with tolerance 1E-16. Thus, if the dimension of a problem is increased, the number of iterations is exponentially increased. Suppose that the problem is n-dimensional. It requires a lot of computational effort to converge from the initial vertex $a=(a_1,\cdots,a_n)$ to the vertex $(a_1,\cdots,a_{n-1},b_n)$ and from the vertex $(a_1,\cdots,a_{n-1},a_n)$ to

Table 1: The subsequence converging to the vertex (3, 1, 1)

Sequence	x_1	x_2	x_3
1	2.3333	1	1
2	2.7143	1	1
3	2.8667	1	1
4	2.9355	1	1
5	2.9683	1	1
6	2.9843	1	1
7	2.9922	1	1
8	2.9961	1	1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
20	3.0000	1	1

the vertex $(a_1, \cdots, b_{n-1}, b_n)$ and so on. For high dimensional problems, these converging subsequences require more memory storage to carry vertices and more computational effort.

We have noticed *jamming* due to the steepness of the angle between a vertex and a point b. We might be able to avoid *jamming* by choosing a reasonable base point b. The reason why a point b is needed is to find a boundary point of H within a rectangle $[v,\ b]$. We propose an alternate scheme for finding a boundary point of H in $[v,\ b]$.

Consider an n-dimensional problem. Let $v = (v_1, v_2, \ldots, v_n)$ be a vertex and let $b = (b_1, b_2, \ldots, b_n)$ be a point. Let d be a non-zero minimum distance between a point b and a vertex v such as

$$d = \min_{\{j : b_j - v_j > 0\}} b_j - v_j. \tag{2.3.6}$$

We define a new point b' as follows:

$$b'_j = \begin{cases} v_j + d & \text{when } v_j < b_j \\ b_j & \text{when } v_j = b_j \end{cases} \tag{2.3.7}$$

for $j = 1, \ldots, n$. If b' is in the reverse normal set H, there exists a boundary point on a line segment $[v,\ b']$. If b' is not in the reverse normal set H, we might find a boundary point on a line segment $[b',\ b]$. In either case, we can find a boundary point x^k of H within a hyperrectangle $[v,\ b]$. Therefore, we can refine a Polyblock by removing the region $[v,\ x^k)$.

With our modified rule, at the first iteration, we start with the vertex $(1, 1, 1)$ and find the boundary point $(\frac{7}{3}, \frac{7}{3}, \frac{7}{3})$. After refining vertex $(1, 1, 1)$, we have the three new vertices $(\frac{7}{3}, 1, 1)$, $(1, \frac{7}{3}, 1)$ and $(1, 1, \frac{7}{3})$. Consider the vertex $(\frac{7}{3}, 1, 1)$. According to equation (2.3.7), the new point b' is $(3, \frac{5}{3}, \frac{5}{3})$ and it is clearly not in the set H. By the modified rule, we should find a boundary point of H on the line segment $[(3, \frac{5}{3}, \frac{5}{3}),\ (3, 3, 3)]$. The corresponding boundary point is $(3, 2, 2)$. Thus, we can refine the vertex $(\frac{7}{3}, 1, 1)$ with three new vertices $(3, 1, 1)$, $(\frac{7}{3}, 2, 1)$ and $(\frac{7}{3}, 1, 2)$. With the original method, we generate twenty vertices having $x_2 = 1$ and $x_3 = 1$, while we only need two vertices having $x_2 = 1$ and $x_3 = 1$ under our modification.

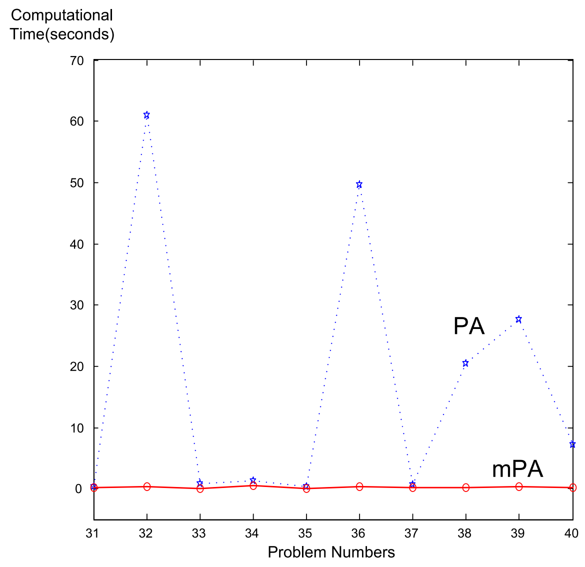
2.3.2.3 Modified Polyblock Algorithm

In order to include the two enhancement ideas discussed in the previous subsections, we need to modify **step k.3** and **step k.5** in Algorithm 1.

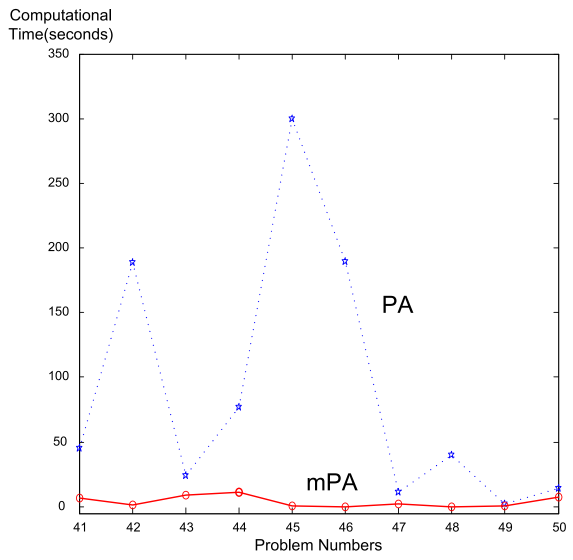
Algorithm 2 Modification of Algorithm 1

```
step k.3: If $T_k \neq \emptyset$, select $v^k \in \arg\min\{\beta(v) | v \in T_k\}$.
if convex relaxations are available then
    Let $w^k$ be an optimal solution of convex relaxation. Reset $\beta(v^k) \leftarrow \max\{f(w^k), \beta(v^k)\}$.
end if
Let $d$ be a nonzero minimum distance between $v^k$ and $b$ in equation (2.3.6). Let $b'$ be a
modified point defined by equation (2.3.7). Let $x^k$ be a boundary point H.
if $b' \in H$ then
    $x^k = \pi_H(v^k,\ b)$
else
    $x^k = \pi_H(b',\ b)$
end if
step k.5: Let $T_{k,*} = \{v \in T_k |\ v < x^k\} \cup v^k$. Compute

    $T_k^{\text{new}} = \{v^{k,i} | v \in T_{k,*}, v_i < x_i^k, i = 1, \ldots, n\}$,

where $v^{k,i} = v + (x_i^k - v_i)e^i$. set $T_{k+1}$ as follows;

    $T_{k+1} = T_k \setminus T_k^x \cup PS(T_k^{x=} \cup T_k^{\text{new}})$

For every $v \in T_k^{\text{new}}$, set $\beta(v) = \max\{f(v), \beta(v^p)\}$ ($v^p$ is the parent of $v$).
```

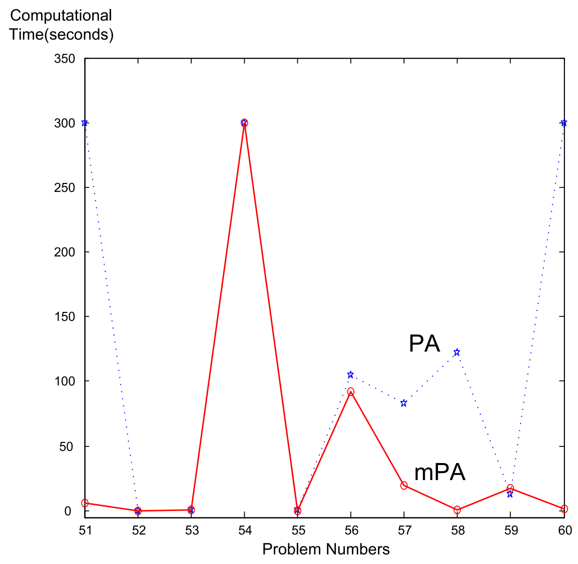
We compared the efficiency of the original and enhanced algorithms for solving separable polynomial programming problems. We randomly generated 4-dimensional test problems with various terminating tolerances (e.g., $\epsilon \in \{0.01, 0.001, 0.0001\}$). In Figure 4, the stars with dotted lines represent the computational time required by the Polyblock Algorithm (labeled 'PA') and the circles with dashed lines represent the computational time required



(a) ε = 0.01



(b) ε = 0.001



(c) ε = 0.0001

Figure 4: CPU seconds of Polyblock Algorithm (PA, Algorithm 1) and modified Algorithm (mPA, Algorithm 2).

by the modified Polyblock Algorithm (labeled 'mPA'). The detailed data are reported in the tables in Appendix A.1. Figure 4 shows that the modified Polyblock Algorithm is better than the Polyblock Algorithm in terms of computational time.

2.3.3 Convergence analysis

Proposition 2.5. *The modified Polyblock Algorithm in Algorithm 2 converges to an optimal solution.*

Proof. Let v^k be a vertex defining a polyblock $[v^k,\ b]$. Let x^k be a boundary point of H that is determined by the rule in **step k.3** of Algorithm 2. We know that $v^k \in G$ and $x^k \in H$. We must show that $\{x^k - v^k\} \to 0$ as $k \to \infty$. Suppose there exists $\eta > 0$ such that $\|x^{k_\nu} - v^{k_\nu}\| \geq n \times \eta > 0$ for some infinite subsequence v^{k_ν} and for all $\nu > M$. Suppose that $v^{k_{\nu_i}}$ denotes the i^{th} vertex which is defined by

$$v^{k_{\nu_i}} = v^{k_\nu} + (x_i^{k_\nu} - v_i^{k_\nu})e^i,$$

for $x_i^{k_\nu} - v_i^{k_\nu} > 0$ and $i = 1, \ldots, n$. This implies that for $x_i^{k_\nu} - v_i^{k_\nu} > 0$,

$$\|v^{k_{\nu_i}} - v^{k_\nu}\| = |x_i^{k_\nu} - v_i^{k_\nu}|^2. \tag{2.3.8}$$

Since the subsequence v^{k_ν} is infinite, for every v^{k_ν} in the subsequence, there exists at least one element i such that $x_i^{k_\nu} - v_i^{k_\nu} > 0$. Also, $\|x^{k_\nu} - v^{k_\nu}\|$ can be written in terms of $\max_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} |x_i^{k_\nu} - v_i^{k_\nu}|$; that is,

$$\begin{aligned} \|x^{k_\nu} - v^{k_\nu}\| &= \sum_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} |x_i^{k_\nu} - v_i^{k_\nu}|^2 \\ &\leq n \times \max_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} |x_i^{k_\nu} - v_i^{k_\nu}|^2. \end{aligned}$$

This implies that we can find some subsequence such that

$$\begin{aligned} n \times \eta &\leq n \times \max_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} |x_i^{k_\nu} - v_i^{k_\nu}|^2 \\ &= n \times \max_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} \|v^{k_{\nu_i}} - v^{k_\nu}\|. \end{aligned}$$

Therefore, there exists a subsequence such that $\eta \leq \max_{\{i:x_i^{k_\nu} - v_i^{k_\nu} > 0\}} \|v^{k_{\nu_i}} - v^{k_\nu}\|$. This contradicts the boundness of the sequence $\{v^{k_{\nu_i}}\} \subset [a, b]$. Thus, $\{v^k - x^k\} \to 0$. We may assume that $v^k \to \overline{x}$ and $x^k \to \overline{x}$. Since $v^k \in G$ and $x^k \in H$, then $\overline{x} \in G \cap H$. Furthermore, $f(v^k)$ is always less than or equal to $f(x)$ for all $x \in G \cap H \cap [v^k,\ b]$. As $k \to \infty$, the sequence $\{v^k\}$ converges to $\overline{x}$. By the selection rule for the vertex in **step k.3** of Algorithm 2, we choose a vertex that has a minimal lower bound. This implies that $f(v^k) \leq f(x)$ for all $x \in G \cap H$. That is, $f(\overline{x}) \leq f(x)$ for all $x \in G \cap H$. This implies that $\overline{x}$ is an optimal solution of (MO). $\square$

2.4 Branch-and-Bound Algorithm

We have shown that monotonic programming problems are nonconvex problems. The branch-and-bound algorithm is a traditional approach for finding global optimal solutions of nonconvex problems. Initially proposed to handle non-convexities arising from discreteness [10], the branch-and-bound scheme has subsequently been expanded to the continuous case [7, 29] and has since evolved into a general purpose global optimization technique for a wide class of non-convex problems [8, 9, 30]. The branch-and-bound algorithm implicitly enumerates all possible solutions by partitioning the region into two or more subregions recursively. For a given partition (sub-region), bounds on the optimal objective value and the feasibility criteria are used to decide whether the given region will be further refined or removed from further consideration. This procedure guarantees convergence to a global solution, but may be very time-consuming.

In the following description, $\mathcal{R}$ denotes a hyperrectangular partition in $\mathbb{R}^n$, i.e., $\mathcal{R} = [a, b]$ with $a, b \in \mathbb{R}^n$ and $a \leq b$; $LB(\mathcal{R})$ denotes the lower bound on the optimal value over partition $\mathcal{R}$; UB denotes an upper bound on the global optimal objective value; $\mathcal{L}$ is a list of unfathomed hyperrectangles; $\beta_{\mathcal{R}}$ denotes an optimal solution of a convex relaxation over hyperrectangle $\mathcal{R}$; $\mathcal{S}$ denotes a set of feasible solutions; k denotes the number of iterations; x^* denotes the best candidate solution; and ϵ denotes a pre-specified optimality tolerance. An algorithmic description of the proposed branch-reduce-cut scheme is presented in Algorithm 3. The key steps are highlighted in italics, and are described below.

The algorithm starts by considering the initial partition $[a,\ b]$ given by the bounds on the variables. We ensure a and b to be in $G \setminus H$ and H, respectively. If this condition is not satisfied, either the problem is infeasible (when a is not in the set G) or a is an optimal solution of the problem (when a is in $G \cap H$).

2.4.1 Selection and branching

The first main step is to select a partition from the list $\mathcal{L}$ of unfathomed partitions. If the list is empty, either we have an ϵ-optimal solution or we resolve that the problem is infeasible. With a non-empty list, we select a partition having a least lower bound according to the

Algorithm 3 A Branch-Reduce-Cut algorithm for (MO)

```
initialization:
  ensure: $a \in G \setminus H$ and $b \in H$
  set $\mathcal{R} = [a, b]$, set $\epsilon > 0$, $\mathcal{L} = \{\mathcal{R}\}$, $LB(\mathcal{R}) = f(a)$, $UB = +\infty$, $x^* = \emptyset$, $k \leftarrow 1$.

selection:
 1: if $\mathcal{L} = \emptyset$ then
 2:   Terminate.
 3: else
 4:   select $\mathcal{R} = [x^L \; x^U] \in \mathcal{L}$ such that $LB(\mathcal{R}) = \min_{\mathcal{R}' \in \mathcal{L}}\{LB(\mathcal{R}')\}$ and $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\mathcal{R}\}$
 5: end if

partition:
 1: branch $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$

domain reduction:
 1: for $i = 1, 2$ do
 2:   reduce the domain $\mathcal{R}_i$ using the set $G$ and $H$
 3:   suppose $\mathcal{R}_i = [x^{L_i} \; x^{U_i}]$
 4:   if $x^{L_i} \in G \setminus H$ and $x^{U_i} \in H$ then
 5:     do bounding
 6:   else
 7:     if $x^{L_i} \in G \cap H$ then
 8:       $\mathcal{S} \leftarrow x^{L_i}$
 9:     end if
10:     fathom $\mathcal{R}_i$
11:   end if
12: end for

update $\mathcal{L}$:
 1: for each $\mathcal{R} \in \mathcal{L}$ do
 2:   if $LB(\mathcal{R}) > UB - \epsilon$ then
 3:     fathom $\mathcal{R}$, i.e., set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\mathcal{R}\}$
 4:   end if
 5: end for
 6: $k \leftarrow k + 1$
 7: goto selection
```

Algorithm 3 A Branch-Reduce-Cut algorithm for (MO)(Continued)

```
***bounding*** $LB(\mathcal{R})$:
1: **if** a convex relaxation is available over $\mathcal{R}_i$ **then**
2:     Let $\beta_{\mathcal{R}_i}$ be an optimal solution of the convex relaxation
3:     **if** $\beta_{\mathcal{R}_i} \in G \cap H$ **then**
4:         $\mathcal{S} \leftarrow \beta_{\mathcal{R}_i}$
5:         *fathom* $\mathcal{R}_i$
6:     **else**
7:         $LB(\mathcal{R}_i) \leftarrow f(\beta_{\mathcal{R}_i})$
8:     **end if**
9: **else**
10:    $LB(\mathcal{R}_i) \leftarrow f(\beta_{x^{L_i}})$
11: **end if**

***bounding*** $UB$:
1: select a finite set $\mathcal{S} \subset \mathcal{R}_i$ (described in section 2.4.3)
2: **for** $x \in \mathcal{S}$ **do**
3:     $\mathcal{S} \leftarrow \mathcal{S} \setminus \{x\}$
4:     **if** $x \in G \cap H$ and $f(x) < UB$ **then**
5:         $x^* \leftarrow x$ and $UB \leftarrow f(x)$
6:         $G \leftarrow G \cap f(x) \leq UB - \epsilon$
7:     **end if**
8: **end for**
```

rule in line 4 of ***selection*** in Algorithm 3. This guarantees that the bounding process is *bound improving* [9].

After selecting a hyperrectangle $\mathcal{R}$, we bisect the longest edge of $\mathcal{R}$. That is, given $\mathcal{R} = [x^L \ x^U]$ with the longest edge index $\hat{\jmath} \in \arg\max_{j=1,\ldots,n}\{x_i^U - x_i^L\}$, $\mathcal{R}$ is partitioned into

$$\begin{aligned}\mathcal{R}_1 &= [x^L,\ x^U - \left(\frac{x_{\hat{\jmath}}^U - x_{\hat{\jmath}}^L}{2}\right) e^{\hat{\jmath}}],\ \text{and}\\ \mathcal{R}_2 &= [x^L + \left(\frac{x_{\hat{\jmath}}^U - x_{\hat{\jmath}}^L}{2}\right) e^{\hat{\jmath}},\ x^U],\end{aligned}$$

where $e^j \in \mathbb{R}^n$ is the j^{th} unit vector. The longest-edge bisection rule guarantees that the branching process is *exhaustive* [9].

2.4.2 Domain reduction

Given a partition $\mathcal{R} = [x^L,\ x^U]$ such that $x^L \in G \setminus H$ and $x^U \in H$, the idea of domain reduction [9, 30] is to find a hyperrectangle $\mathcal{R}' \subset \mathcal{R}$ such that $\mathcal{R}' \supseteq \mathcal{R} \cap G \cap H$. This reduction helps to obtain tighter lower bounds and reduces the size of the search region. Figure 5 shows an example; the left figure in Figure 5 depicts the initial rectangle before a branching step of the algorithm. After a branching step, appropriate domain reductions reduce the search region to two relatively smaller hyper-rectangles (bold boxes in the right figure) without losing feasible solutions. It follows that the bound obtained from refined hyperrectangles might be closer (tighter) to an optimal objective value.

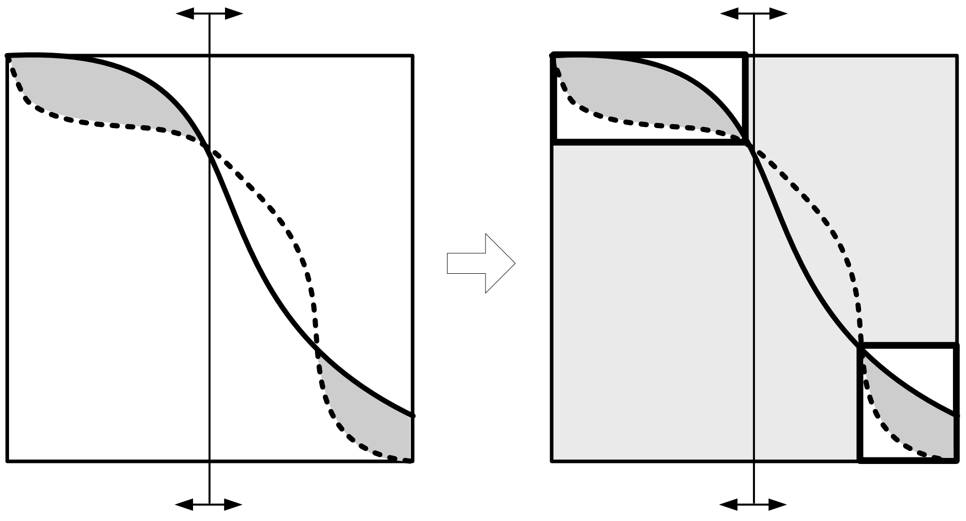


Figure 5: An example of domain reduction

The domain reduction on $\mathcal{R} = [x^L,\ x^U]$, with $x^L \in G \setminus H$ and $x^U \in H$, can be carried out as follows. Let

$$\lambda_j^* = \max\{\lambda \in [0,1] : x^L + \lambda(x_j^U - x_j^L)e^j \in G\} \quad \text{for } j = 1, \ldots, n, \tag{2.4.1}$$

$$\mu_j^* = \max\{\mu \in [0,1] : x^U - \mu(x_j^U - x_j^L)e^j \in H\} \quad \text{for } j = 1, \ldots, n. \tag{2.4.2}$$

Since G and H are closed sets, the above problems have optimal solutions. In general, each optimal solution of (2.4.1) and (2.4.2) can be obtained by using bisection search. We reduce the size of a given partition $\mathcal{R}$ to $\mathcal{R}' = [x^{L'},\ x^{U'}]$, where

$$x^{L'} = x^U - \sum_{j=1}^{n} \mu_j^*(x_j^U - x_j^L)e^j, \text{ and} \tag{2.4.3}$$

$$x^{U'} = x^L + \sum_{j=1}^{n} \lambda_j^*(x_j^U - x_j^L)e^j. \tag{2.4.4}$$

Proposition 2.6. $[x^{L'},\ x^{U'}] \supseteq [x^L,\ x^U] \cap G \cap H$.

Proof. Suppose there exists $x \in [x^L,\ x^U] \cap G \cap H$ and $x \in [x^{L'},\ x^{U'}]$. Then, there exists an index $\hat{\jmath} \in \{1, \ldots, n\}$ such that either $x^L_{\hat{\jmath}} \le x_{\hat{\jmath}} < x^{L'}_{\hat{\jmath}}$ or $x^{U'}_{\hat{\jmath}} < x_{\hat{\jmath}} \le x^U_{\hat{\jmath}}$. Consider the first case (the second case is analogous). Let $y = x + \sum_{j=1, j \ne \hat{\jmath}}^{n} (x^U_j - x_j)e^j$; i.e., $y_j = x^U_j$ for all $j = 1, \ldots, n, j \ne \hat{\jmath}$ and $y_{\hat{\jmath}} = x_{\hat{\jmath}}$. Let $\widehat{\mu}_{\hat{\jmath}} = (x^U_{\hat{\jmath}} - y_{\hat{\jmath}})/(x^U_{\hat{\jmath}} - x^L_{\hat{\jmath}})$. Clearly $\widehat{\mu} \in [0, 1]$, and we can write $y = x^U - \widehat{\mu}_{\hat{\jmath}}(x^U_{\hat{\jmath}} - x^L_{\hat{\jmath}})e^{\hat{\jmath}}$. Note that, by construction, $y \ge x$, and since $x \in H$, we have that $y \in H$ (since H is a reverse normal set). Thus $\widehat{\mu}_{\hat{\jmath}}$ is a feasible solution to problem (2.4.2) corresponding to $\hat{\jmath}$. Then

$$x^U_{\hat{\jmath}} - \widehat{\mu}_{\hat{\jmath}}(x^U_{\hat{\jmath}} - x^L_{\hat{\jmath}}) = y_{\hat{\jmath}} = x_{\hat{\jmath}} < x^{L'}_{\hat{\jmath}} = x^U_{\hat{\jmath}} - \mu^*_{\hat{\jmath}}(x^U_{\hat{\jmath}} - x^L_{\hat{\jmath}}).$$

The above implies that $\widehat{\mu}_{\hat{\jmath}} > \mu^*_{\hat{\jmath}}$ and we have a contradiction to the fact that $\mu^*_{\hat{\jmath}}$ is an optimal solution to problem (2.4.2) corresponding to $\hat{\jmath}$. $\square$

Proposition 2.6 shows that the domain reduction procedure is valid. That is, no feasible solution is lost. The domain reduction procedure can be improved by exploiting a property of (MO). By Proposition 2.1 (*iii*), once we find a feasible point x^k, the region $\{x | x^k < x \le b\}$ can be removed from further consideration. Based on this idea, we refine λ^*_j as follows:

$$\lambda^*_j = \min\{\lambda^G_j, \lambda^H_j\}, \tag{2.4.5}$$

where

$$\begin{aligned} \lambda^G_j &= \max\{\lambda \in [0,1] : x^L + \lambda(x^U_j - x^L_j)e^j \in G\} \quad \text{for } j = 1, \ldots, n, \text{ and} & (2.4.6)\\ \lambda^H_j &= \min\{\lambda \in [0,1] : x^L + \lambda(x^U_j - x^L_j)e^j \in H\}, & (2.4.7) \end{aligned}$$

for $\{j \in \{1, \ldots, n\} : x^L + (x^U_j - x^L_j)e^j \in H\}$ and $\lambda^H_j = 1$ for $\{j \in \{1, \ldots, n\} : x^L + (x^U_j - x^L_j)e^j \notin H\}$. Recall that set $\mathcal{S}$ contains candidate feasible solutions. Since we may find a feasible solution by solving (2.4.7), we should add it into the set $\mathcal{S}$.

Proposition 2.7. *By using a solution* λ^*_j *of* (2.4.5), *for any* $x \in [x^L,\ x^U] \cap G \cap H$, *there exists* $\bar{x}^* \in [x^{L'},\ x^{U'}] \cap G \cap H$ *such that* $f(\bar{x}^*) \le f(x)$.

Proof. We need consider only the case that $x \in [x^L,\ x^U] \cap G \cap H$ and $x \notin [x^{L'},\ x^{U'}] \cap G \cap H$; the other case is trivial. By Proposition 2.6, we have shown that $[x^{L'},\ x^{U'}]$ is a valid reduced domain for λ_j^*'s obtained from (2.4.6) and for μ_j^*.

Assume that there exists $x \in [x^L,\ x^U] \cap G \cap H$ such that $f(x) \leq f(\bar{x})$ for all $\bar{x} \in [x^{L'},\ x^{U'}] \cap G \cap H$. Then, there exists an index $\widehat{\jmath} \in \{1, \dots, n\}$ such that $x_{\widehat{\jmath}}^{U'} < x_{\widehat{\jmath}} \leq x_{\widehat{\jmath}}^U$. By Proposition 2.6, $x_{\widehat{\jmath}}^{U'}$ is obtained from (2.4.7). That is, $\lambda_{\widehat{\jmath}}^H \leq \lambda_{\widehat{\jmath}}^G$. Construct

$$\bar{x}^* = x^L + \lambda_{\widehat{\jmath}}^H (x_{\widehat{\jmath}}^U - x_{\widehat{\jmath}}^L) e^{\widehat{\jmath}}. \tag{2.4.8}$$

It is clear that $x_j^L \leq x_j$ for all $j \in \{1, \dots, n\}$ and $x_{\widehat{\jmath}}^{U'} < x_{\widehat{\jmath}}$. It implies $\bar{x}^* \leq x$. Because of the definition of problem (2.4.7), $\bar{x}^*$ is in the reverse normal set H. Furthermore, $\lambda_{\widehat{\jmath}}^H \leq \lambda_{\widehat{\jmath}}^G$ implies that

$$x^L + \lambda_{\widehat{\jmath}}^H (x_{\widehat{\jmath}}^U - x_{\widehat{\jmath}}^L) e^{\widehat{\jmath}} \leq x^L + \lambda_{\widehat{\jmath}}^G (x_{\widehat{\jmath}}^U - x_{\widehat{\jmath}}^L) e^{\widehat{\jmath}} \in G. \tag{2.4.9}$$

Thus, $\bar{x}^* \in G$ (by the monotonicity of the set G). We found the point $\bar{x}^* \in G \cap H \cap [x^{L'}, x^{U'}]$ having $f(\bar{x}^*) \leq f(x)$. This contradicts our assumption. $\square$

According to equation (2.4.3) and equation (2.4.4), we tighten lower bounds with the upper bound values and tighten upper bounds with the lower bound values. Once we update bounds, we may re-tighten bounds with new bounds values. It follows that the domain reduction for a given hyperrectangle can be performed several times. Figure 6 shows one iteration of domain reductions. After one iteration, further reduction does not change the size of the hyperrectangle in this example. It is time-consuming to get the tightest bound. Hence, we can specify the number of domain reductions steps *a priori*. The termination conditions for the procedure of domain reductions is either that there is no improvement for bounds or that the number of iterations reaches the pre-defined limit.

2.4.3 Bounding and optimality cuts

In branch-and-bound schemes, a tighter bound plays a key role in reducing computational effort. For a given hyperrectangle $\mathcal{R} = [x^L,\ x^U]$, a lower bound $(LB(\mathcal{R}))$ is obtained from the objective function value of x^L. For a high dimensional problem, a lower bound $(LB(\mathcal{R}) = f(x^L))$ is usually weak; especially, in early stages of the process, the gap between

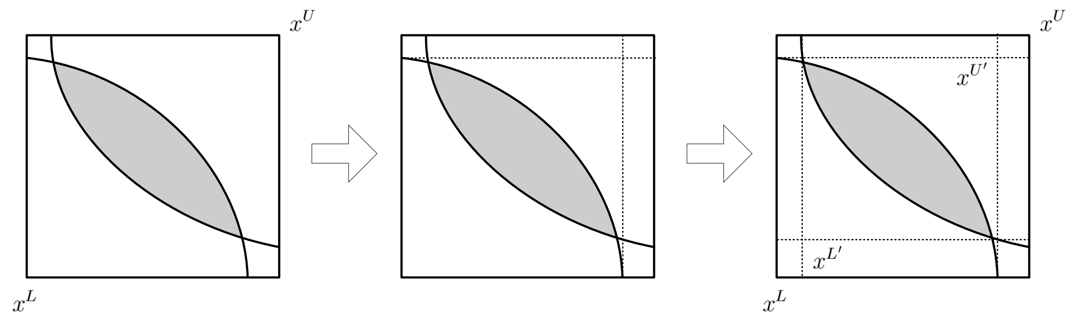


Figure 6: An illustrative example of domain reductions

a lower bound ($LB(\mathcal{R})$) and an upper bound (UB) of the problem is wide. One way to improve bounds is to use convex relaxations. To be effective for branch-and-bound procedures, the relaxation needs to be solved easily and its optimal objective value needs to be a tighter bound than the Polyblock Algorithm bound for a given partition; that is, an optimal solution of the convex relaxed subproblem should be obtained without much effort and the relaxations need to exclude as much of the infeasible regions as possible.

Whenever a feasible solution is found, the corresponding objective value is checked against the upper bound (UB). In ***bounding*** UB of Algorithm 3, feasible solutions in the set $\mathcal{S}$ are checked to update the upper bound UB. Whenever a better solution is found in the previous steps, it is added to the set $\mathcal{S}$. To search for more feasible solutions within an unfathomed partition $\mathcal{R} = [x^L,\ x^U]$, the algorithm checks the set of n vertices of $\mathcal{R}$ adjacent to x^U. Adjacent vertices b^j are defined as follows:

$$b^j = x^U - (x_j^U - x_j^L)e^j \quad \text{for } j = 1, \ldots, n. \tag{2.4.10}$$

Note that adjacent vertices of x^L are considered in the steps of domain reduction.

Whenever the upper bound UB is updated, an optimality cut can be generated as follows:

$$f(x) \leq UB - \epsilon \quad \text{for } x \in [x^L,\ x^U]. \tag{2.4.11}$$

Due to the monotonicity of objective function f, the set defined by equation (2.4.11) is a normal set. In order to ensure that the set is closed, we assume that f is lower semi-continuous. In line 6 of ***bounding*** UB, an optimality cut is added into the normal set. We add a cut so that we have a tighter bound in domain reductions. Recall that in domain reductions for x^U in $\mathcal{R} = [x^L,\ x^U]$, we use the normal set and a point x^L. It is obvious that the smaller the normal set, the tighter the bound obtained.

2.4.4 Fathoming

A given partition $\mathcal{R} = [x^L,\ x^U]$ can be fathomed, or discarded from further consideration. In Algorithm 3, there are three fathoming steps - (a) line 10 of ***domain reduction***, (b) line 5 of ***bounding*** $LB(\mathcal{R})$, and (c) line 3 of ***update*** $\mathcal{L}$. We fathom a partition $\mathcal{R} = [x^L,\ x^U]$ if one of the following conditions are satisfied.

Infeasibility: If $x^L \notin G$ or $x^U \notin H$ for the given partition $\mathcal{R} = [x^L,\ x^U]$, then it can be fathomed since $\mathcal{R}$ contains no feasible solutions. Such a partition will be fathomed in step (a).

Feasibility: For a given partition $\mathcal{R} = [x^L,\ x^U]$, if we found a feasible solution $\bar{x} \in G \cap H$ such that $f(\bar{x}) = LB(\mathcal{R})$, the given partition will be removed from further consideration because it does not contain a better solution. Such partitions can be identified in the following two cases: one is when x^L is in the feasible set $G \cap H$, and the other is when an optimal solution of a convex relaxation over partition $\mathcal{R}$ is in the feasible region $G \cap H$. Partitions belonging to the first case are fathomed in step (a), while partitions in the other case are fathomed in step (b).

Inferiority: When $LB(R) > UB - \epsilon$, we fathom such partition $\mathcal{R}$ since it contains no better feasible solution.

2.4.5 Convergence analysis

Consider a nested sequence of successively refined partitions $\{\mathcal{R}_{j^k}\}$ such that $\mathcal{R}_{j^k} \subset \mathcal{R}_j$. We consider two cases; one is that Algorithm 3 terminates with $\epsilon = 0$ and the other is that Algorithm 3 does not terminate with $\epsilon = 0$. To prove the convergence of Algorithm 3, we shall need the following concept.

Definition 2.1. [9] *A bounding operation is called* consistent *if, at every step, any unfathomed partition element can be further refined, and if any infinitely decreasing sequence*

$\{\mathcal{R}_{j^k}\}$ of successively refined partition elements satisfies

$$\lim_{k\to\infty} UB - LB(\mathcal{R}_{j^k}) = 0. \tag{2.4.12}$$

Lemma 2.1. *In an infinite branch-and-bound procedure, suppose that the bounding operation is consistent and the selection operation is bound improving. Then the procedure converges to an optimal solution.*

Proof. See Theorem IV.3. in [9]. □

Lemma 2.2. *The bounding operation of Algorithm 3 is* consistent*.*

Proof. Consider a subset $\mathcal{R}_{j^k} = [x^L_{j^k}\ x^U_{j^k}]$ that is unfathomed. We know that $x^L_{j^k} \in G$ and $x^U_{j^k} \in H$. This implies that the branching step can further refine it. By the exhaustiveness of the branching rule, the following limit holds

$$\lim_{k\to\infty} (x^U_{j^k} - x^L_{j^k}) = 0. \tag{2.4.13}$$

Since the set G and H are closed, for any convergent subsequence, we must have $\lim_{k\to\infty} x^L_{j^k} = \lim_{k\to\infty} x^U_{j^k} = x^* \in G \cap H$. This implies that x^* is feasible solution of a bounded (MO) and hence x^* is an upper bound for $\mathcal{R}_{j^k}$. By Definition 2.1, the bounding operation of Algorithm 3 is consistent. □

Theorem 2.1. *If Algorithm 3 with $\epsilon = 0$ terminates, then it terminates either with a global optimal solution or resolves that the problem is infeasible.*

Proof. The selection operation of Algorithm 3 is bounding improving (cf. see section 2.4.1) and the bounding operation of Algorithm 3 is consistent by Lemma 2.2. It follows that the algorithm converges by Lemma 2.1.

We consider the first case that the algorithm terminates with a feasible solution x^*. This implies that the given problem (MO) is nonempty, therefore (MO) has an optimal solution. Suppose that x' is an optimal solution such that $f(x') < f(x^*)$. We consider a set of partitions $\mathcal{R}_i = [a'_i\ b'_i]$ that contains x'. Since x' is a feasible solution ($x' \in G \cap H$)

and $a'_i \leq x' \leq b'_i$, we know that a'_i is in the normal set G and b'_i is in the reverse normal set H by the definition of the normal and reverse normal set. This implies that the partitions containing x' cannot be fathomed due to infeasibility. Suppose a partition $\mathcal{R}_i$ is fathomed due to feasibility; i.e., a'_i is a feasible solution such that $a'_i \in G \cap H$. This implies that $f(a'_i) \leq f(x') < f(x^*)$. By the rule of choosing an upper bound, $f(a'_i)$ should be selected as an upper bound. This contradicts the assumption that x^* is a solution of the problem. Finally, since $f(a'_i) \leq f(x') < f(x^*)$, the algorithm cannot fathom the partition $\mathcal{R}_i$ due to inferiority. Thus, $\mathcal{R}_i$ can be further refined. This implies that the algorithm should not have terminated. Hence $f(x') = f(x^*)$.

Consider the second case that the algorithm terminated with no feasible solution. This implies that the upper bound (UB) remains $+\infty$ upon termination. Suppose we have a feasible solution x'. Any partition $\mathcal{R}_i$ containing x' cannot be fathomed by inferiority since $f(a'_i) \leq f(x') < \infty$. Suppose we fathom a partition $\mathcal{R}_i = [a'_i \ b'_i]$ due to feasibility. It implies that a'_i belongs to G and H. This contradicts $UB = \infty$. So, we cannot fathom a partition $\mathcal{R}_i$ due to feasibility. Thus, $\mathcal{R}_i$ can be further refined, and the algorithm should not have terminated. Hence, (MO) cannot have any feasible solutions. □

Theorem 2.2. *If Algorithm 3 with $\epsilon = 0$ does not terminate, then the (MO) has an optimal solution x^* and there is a sequence of partitions $\mathcal{R}_{k_q} = [a^{k_q},\ b^{k_q}]$ such that*

$$\lim_{q\to\infty} a^{k_q} = x^* = \lim_{q\to\infty} b^{k_q} \tag{2.4.14}$$

Proof. By Lemma 2.2, there exists x^* that satisfies the equation 2.4.14. We must show that x^* is an optimal solution for a given bounded (MO). Since the partition is not fathomed, a^{k_q} is in G and b^{k_q} is in H. This implies that $x^* \in G \cap H$.

The least lower bound selection rule guarantees that $f(a^{k_q}) \leq f(x)$ for all q and all $x \in G \cap H$. Thus, $\lim_{q\to\infty} f(a^{k_q}) = \lim_{q\to\infty} f(b^{k_q}) = f(x^*) \leq f(x)$ for all $x \in G \cap H$. Therefore, x^* is an optimal solution of (MO). □

2.5 Simplicial branching

In Section 2.4, we discussed the rectangular branch-and-bound algorithm. In the rectangular branch-and-bound algorithm, we branch on the longest edge in order to keep sub-partitions in the shape of a rectangle. In this section, we discuss the conical branch-and-bound algorithm. While we define regions under our consideration with rectangles in rectangular branching schemes, we define the feasible region with cones in conical branching schemes. The conical branching schemes for monotonic programming problems have also been considered by Tuy [34, 35]. Conical branching schemes have shown good performance in optimization problems involving concave functions in the objective and constraints (cf. [9]).

In the conical algorithm, we start the algorithm with an initial cone in $\mathbb{R}^n_+$ that contains all feasible points of a given problem. The initial cone is partitioned into sub-cones (partitions). We provide a valid bound for each cone (partition). According to the bound information, we determine whether a cone is considered in further steps or not. We keep refining cones until we do not have any cone left for our consideration. The major steps of the conical algorithm are identical to the branch-and-bound algorithm. The only differences for the conical algorithm are the way of partitioning the feasible region — conical subdivision, the branching schemes, and the bounding schemes.

The conical subdivision can be represented by subdivision of the simplex $S_0^b := [s^1, \ldots, s^n]$, where $s^i \in \mathbb{R}^n_+$ with an origin point b. Each simplex $S_i^b := [s^1, \ldots, s^n]$ defines a cone K_i with n extreme rays such as $u^i = b - \alpha_i s^i$ for $\alpha_i \geq 0$ and $i = 1, \ldots, n$. Without loss of generality, we may set the initial simplex $S_0^b := [s^1, \ldots, s^n]$, where $s^i = e^i$ and $e^i = [0 \cdots 1 \cdots 0]^T$ is the i^{th} unit vector. That is, the initial cone K_0 contains all points in $[a\ b]$.

For each simplex S_i, we are interested in the feasible region within a corresponding cone K_i such as $x \in K_i \cap G \cap H$. Let u^i be a boundary point of H such as

$$u^i = b - \lambda_i^* s^i, \tag{2.5.1}$$

where $\lambda_i = \max\{\lambda \in \mathbb{R}_+ : b - \lambda s^i \in H\}$. For a bounded (MO), we define a hyperrectangle

$\mathcal{R} = [x^L \ x^U]$ for each cone K_i such that

$$x_j^L = a_j, \quad \forall j = 1, \ldots, n, \tag{2.5.2}$$

$$x_j^U = \max_{1 \le i \le n} u_j^i, \quad \forall j = 1, \ldots, n. \tag{2.5.3}$$

Proposition 2.8. *If x^* is an optimal solution of $\min\{f(x)|x \in G \cap H \cap K_i\}$, there exists x' such that $x' \in G \cap H \cap [x^L \ x^U]$ and $f(x^*) = f(x')$.*

Proof. See, [33, 35]. □

We have established a hyper-rectangle $\mathcal{R}_i = [x^L \ x^U]$ for a cone K_i. By the reduction proposed in section 2.4.2, we might modify the lower bound x^L and upper bound x^U. After domain reduction, the objective function value corresponding to x^L is a valid lower bound of the feasible region over a cone K_i.

In order to ensure that the process is bound improving [9], we select a partition (cone) which achieves a least lower bound.

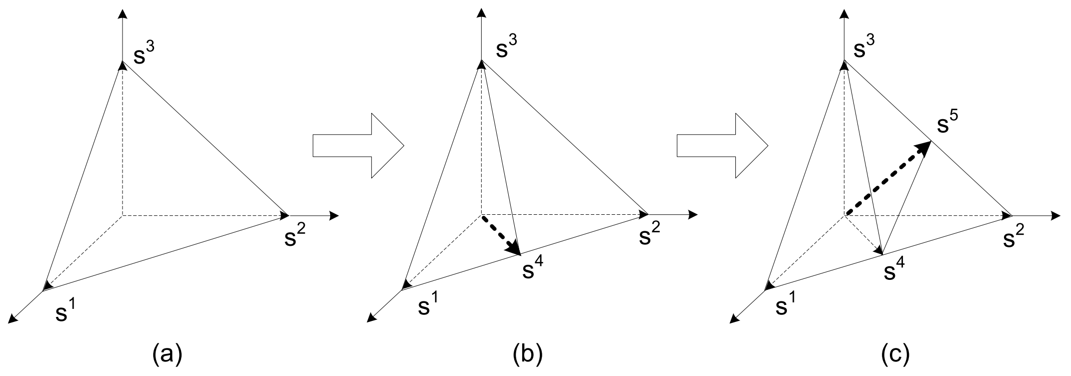


Figure 7: An illustrative example of simplex branching

Figure 7 shows an example of conical branchings. The first figure in Figure 7 (a) depicts the initial simplex which is defined by the three unit vectors $s^1 = [1\ 0\ 0]^T$, $s^2 = [0\ 1\ 0]^T$ and $s^3 = [0\ 0\ 1]^T$. For our example, $S_0^o = [s^1 \ s^2 \ s^3]$. We should choose an edge for branching. Suppose that we choose the edge between vector s^1 and s^2. The bisection point is s^4. After branching, we have two simplices such as $S_1^o := [s^1 \ s^3 \ s^4]$ and $S_2^o := [s^2 \ s^3 \ s^4]$ (see Figure 7 (b)). The last figure (c) in Figure 7 shows one further step.

In the branching scheme, we bisect the longest edge of a simplex in the following manner. Suppose we choose a cone K_i with corresponding S_i. First, we find a longest edge $(\widehat{\imath}, \widehat{\jmath})$ between $s^{\widehat{\imath}}$ and $s^{\widehat{\jmath}}$ such that

$$(\widehat{\imath}, \widehat{\jmath}) \in \arg \max_{i=1,\ldots,n, j=i+1,\ldots,n} \{\|s^i - s^j\|\}. \tag{2.5.4}$$

We create a vector s' such that $s' = \frac{s^{\imath} - s^{\jmath}}{2}$. The simplex S_i is partitioned into

$$S_i^1 = \{S_i \setminus s^{\widehat{\imath}}\} \cup s', \quad \text{and} \tag{2.5.5}$$

$$S_i^2 = \{S_i \setminus s^{\widehat{\jmath}}\} \cup s'. \tag{2.5.6}$$

The longest-edge bisection rule guarantees that the branching process is exhaustive [9].

Since we create a valid hyperrectangle for each cone, we may use the fathoming rules proposed in section 2.4.4 and other improvements such as domain reduction schemes and optimality cuts without modification of either the algorithm or the procedures. The computational results will be provided in section 3.2.3.

CHAPTER III

COMPUTATIONAL RESULTS FOR SEPARABLE POLYNOMIAL PROGRAMMING PROBLEMS

3.1 Separable polynomial programming

The main purpose of this chapter is to compare the performance of the Polyblock Algorithm, the modified Polyblock Algorithm, and branch-and-bound for solving separable polynomial problems. Separable polynomial programming is a class of polynomial programming involving polynomials that do not have cross-product terms between variables. A separable polynomial programming problem can be easily transformed into a class of monotonic programming problems.

We consider a general separable polynomial programming problem of the form;

$$\begin{aligned} \text{(SPP):} \quad \min \quad & \sum_{j=1}^{n} f_j(x_j) && (3.1.1)\\ \text{s.t.} \quad & \sum_{j=1}^{n} h_{ij}(x_j) \geq 0 \quad \text{for } i = 1, \ldots, m && (3.1.2)\\ & x \in [x^L,\ x^U], && (3.1.3) \end{aligned}$$

where $x \in \mathbb{R}^n$ is a vector of decision variables bounded by $x^L, x^U \in \mathbb{R}^n$; $f_j : \mathbb{R} \mapsto \mathbb{R}$ and $h_{ij} : \mathbb{R} \mapsto \mathbb{R}$ are univariate polynomial functions; and m, n denote the number of constraints and the dimension of decision variables, respectively. Univariate polynomial functions can be redefined as follows:

$$f_j(x_j) = (\alpha_1^j)x_j + (\alpha_2^j)x_j^2 + \cdots + (\alpha_{r_j^0}^j)x_j^{r_j}, \quad \text{for } j = 1, \ldots, n \qquad (3.1.4)$$

where $r_j^0 \in \mathbb{Z}^+$ is the degree of the polynomial f_j, and $\alpha_k^j \in \mathbb{R}$ is a constant for $k = 1, \ldots, r_j^0$. Since the polynomial f_j appears only in the objective function, we assume that the constant term of the polynomial is zero. Similarly, the univariate polynomials that appear in the constrains are of the form;

$$h_{ij}(x_j) = (\beta_0^{ij}) + (\beta_1^{ij})x_j + (\beta_2^{ij})x_j^2 + \ldots + (\beta_{r_j^i}^{ij})x_j^{r_j^i}, \quad \text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n \tag{3.1.5}$$

where $r_j^i \in \mathbb{Z}^+$ is the degree of the polynomial h_{ij}, and $\beta_k^{ij} \in \mathbb{R}$ is a constant for $k = 1, \ldots, r_j^i$. In general, the problem (SPP) is not a monotonic programming problem.

3.1.1 Problem transformation

Without loss of generality, we can assume that $x_j^L \geq 0$ for all $j = 1, \ldots, n$. Indeed, if $x_j^L < 0$, we can replace x_j with $x_j^+ = x_j - x_j^L$ and the new variable ^+_j is in $[0,\ x_j^U - x_j^L]$.

Under the assumption of $x_j \geq 0$, a problem (SPP) is of the form either a monotonic programming problem or a difference of monotonic programming problem. Since the x_j's are positive variables, a term cx_j^r in either (3.1.4) or (3.1.5) has a monotone property where c is a real number and r is a positive integer, i.e., when $c > 0$, the term cx_j^r is monotone increasing and for $c < 0$, it is monotone decreasing. Let K_{ij}^+ be a set of indices such that $K_{ij}^+ := \{k |\ \beta_k^{ij} > 0, k = 1, \ldots, r_j^i\}$ and let K_{ij}^- be a set of indices such that $K_{ij}^- := \{k |\ \beta_k^{ij} < 0, k = 1, \ldots, r_j^i\}$ for a given i and j. We define two functions h_{ij}^+ and h_{ij}^- such as

$$\begin{aligned} h_{ij}^+(x_j) &= \sum_{k \in K_{ij}^+} (\beta_k^{ij})x_j^k, \quad \text{and} \\ h_{ij}^-(x_j) &= \sum_{k \in K_{ij}^-} (\beta_k^{ij})x_j^k. \end{aligned}$$

A polynomial in the constraints is divided into two polynomials h_{ij}^+ having positive coefficients and h_{ij}^- having negative coefficients for each term; i.e.,

$$\sum_{j=1}^n h_{ij}(x_j) = \sum_{j=1}^n h_{ij}^+(x_j) + \sum_{j=1}^n h_{ij}^-(x_j). \tag{3.1.6}$$

The right-hand side of the equality (3.1.6) is of the form difference of monotonic functions. Adding an additional variable t_i, for a given i, the equality (3.1.6) can be written as follows:

$$\sum_{j=1}^n h_{ij}(x_j) = \sum_{j=1}^n h_{ij}^+(x_j) + t_i, \tag{3.1.7}$$

$$0 = t_i - \sum_{j=1}^n h_{ij}^-(x_j). \tag{3.1.8}$$

The right-hand side of equality (3.1.7) is monotone non-decreasing. Since h_{ij}^{-} is monotone non-increasing, $-h_{ij}^{-}$ is monotone non-decreasing. Thus, the right-hand side of equality (3.1.8) is also monotone non-decreasing. Therefore, we are able to transform a constraint having separable polynomials into three monotone polynomial constraints; i.e., the inequality (3.1.2) can be replaced with the following three inequalities;

$$\sum_{j=1}^{n} h_{ij}^{+}(x_j) + t_i \geq 0, \tag{3.1.9}$$

$$t_i - \sum_{j=1}^{n} h_{ij}^{-}(x_j) \geq 0, \tag{3.1.10}$$

$$t_i - \sum_{j=1}^{n} h_{ij}^{-}(x_j) \leq 0, \tag{3.1.11}$$

for $i = 1, \ldots, m$. The first two inequalities (3.1.9) and (3.1.10) define a reverse normal set while the last inequality (3.1.11) defines a normal set. Next, we consider the objective function of a problem (SPP). This term can be replaced with a linear monotone objective by introducing an additional variable t_o. Consider the following problem:

$$\min_{x} \quad t_o \tag{3.1.12}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} f_j(x_j) \leq t_o, \tag{3.1.13}$$

$$x \in G \cap H, \tag{3.1.14}$$

where $G \cap H$ denotes the feasible set defined by (3.1.2) and (3.1.3). The given problem is equal to a problem (SPP) and the inequality (3.1.13) consists of difference of monotone functions. After transforming the inequality (3.1.13) into monotonic constraints, we have a monotonic programming problem.

3.1.2 Convex relaxations

Consider Example 2.1 presented in section 2.2. The example (EX1) is a polynomial programming problem and shows that the feasible set defined by the polynomials is non-convex, and even disconnected. For a given bounded problem, tight lower bounds are required in order to improve the efficiency of various algorithms.

Two relaxation schemes - reformulation-linearization technique(RLT) [26, 27, 28] and semidefinite programming(SDP) [11] have been proposed for constructing convex relaxations of nonconvex polynomial programming problems.

Consider a separable polynomial program with monotone polynomials;

$$\begin{aligned} \text{(SPMO):} \quad \min \quad & c^T x && (3.1.15) \\ \text{s.t.} \quad & \sum_{j=1}^{n} h_{ij}(x_j) \geq 0, \quad \text{for } i = 1, \ldots, m_H, && (3.1.16) \\ & \sum_{j=1}^{n} g_{ij}(x_j) \leq 0, \quad \text{for } i = 1, \ldots, m_G, && (3.1.17) \\ & x \in [x^L,\ x^U], && (3.1.18) \end{aligned}$$

where $x \in \mathbb{R}^n$ denotes a vector of decision variables; polynomial functions h_{ij} and g_{ij} are monotone non-decreasing; m_H and m_G denote the number of constraints defining a reverse normal set and a normal set, respectively; $c \in \mathbb{R}$ is a constant; and $x^L, x^U \in \mathbb{R}^n_+$.

We propose a linear relaxation that is an outer approximation of a univariate polynomial. The convex relaxations will then be linear programs, and it will be solved without much effort. In section 3.1.1, we have transformed problem (SPP) to problem (SPMO) having a linear objective. Therefore, we only consider convex relaxations of the constraints.

3.1.2.1 Linear relaxation of a reverse normal set

Consider a linear function $h_{ij}^{L_p} : \mathbb{R} \mapsto \mathbb{R}$ such that

$$h_{ij}^{L_p}(x_j) \geq h_{ij}(x_j), \quad \forall\ x_j \in [x_j^L,\ x_j^U]. \tag{3.1.19}$$

We can construct a linear relaxation of the constraint (3.1.16) with these linear functions. The proof of Proposition 3.1 is trivial and is omitted.

Proposition 3.1. *For a given i, if linear functions $h_{ij}^{L_p} : \mathbb{R} \mapsto \mathbb{R}$ satisfy the inequality* (3.1.19) *for all $j = 1, \ldots, n$, then the linear inequality,*

$$\sum_{j=1}^{n} h_{ij}^{L_p}(x_j) \geq 0, \tag{3.1.20}$$

is a valid outer approximation of a constraint (3.1.16).

The basic idea of constructing a linear relaxation $h_{ij}^{L_p}$ is as follows: Recall that h_{ij} can be divided into two parts h_{ij}^{+} and h_{ij}^{-} according to the definitions of K_{ij}^{+} and K_{ij}^{-} (see, section 3.1.1). Gradients of functions h_{ij}^{+} and h_{ij}^{-} are also strictly monotone. Thus, the functions h_{ij}^{+} and h_{ij}^{-} are convex and concave, respectively. Owing to condition (3.1.19), for h_{ij}^{+}, we can construct a linear relaxation having the same function values as h_{ij}^{+} at the lower and upper bounds of variable x_j, while two tangent lines crossing at the lower and upper bound of x_j is used as a linear relaxation of h_{ij}^{-}. Figure 8 shows relaxations of h_{ij}^{+} and h_{ij}^{-}.

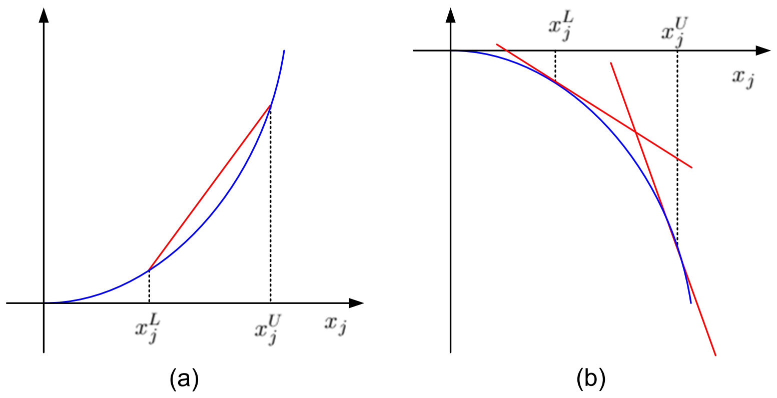


Figure 8: Linear relaxation for univariate polynomial h_{ij} : (a) relaxation for h_{ij}^{+} and (b) relaxations for h_{ij}^{-}

More precisely, the linear relaxations of h_{ij}^{+} and h_{ij}^{-} are computed as follows: let u_{ij}^{+} and l_{ij}^{+} be the corresponding function values of $h_{ij}^{+}(x_j^U)$ and $h_{ij}^{+}(x_j^L)$, respectively, and let u_{ij}^{-} and l_{ij}^{-} be the corresponding function values of $h_{ij}^{-}(x_j^U)$ and $h_{ij}^{-}(x_j^L)$, respectively; let ∇h_{ij}^{L-} be the gradient at $x_j = x_j^L$ such that $\nabla h_{ij}^{L-} = \nabla h_{ij}^{-}(x_j^L)$; and let ∇h_{ij}^{U-} be the gradient at $x_j = x_j^U$ such that $\nabla h_{ij}^{U-} = \nabla h_{ij}^{-}(x_j^U)$.

- Let $h_{ij}^{L_1^+}$ be a linear relaxation of h_{ij}^{+} such that

$$h_{ij}^{L_1^+}(x_j) = \frac{u_{ij}^{+} - l_{ij}^{+}}{x_j^U - x_j^L} x_j + u_{ij}^{+} - \frac{u_{ij}^{+} - l_{ij}^{+}}{x_j^U - x_j^L} x_j^U. \tag{3.1.21}$$

- Let $h_{ij}^{L_1^-}$ and $h_{ij}^{L_2^-}$ be linear relaxations of h_{ij}^{-} such that

$$h_{ij}^{L_1^-}(x_j) = \nabla h_{ij}^{L-} x_j + l_{ij}^{-} - \nabla h_{ij}^{L-} x_j^L, \quad \text{and} \tag{3.1.22}$$

$$h_{ij}^{L_2^-}(x_j) = \nabla h_{ij}^{U-} x_j + u_{ij}^{-} - \nabla h_{ij}^{U-} x_j^U. \tag{3.1.23}$$

Thus, we have two possible linear relaxations of h_{ij} by combining functions either (3.1.21) and (3.1.22), or (3.1.21) and (3.1.23) such as

$$h_{ij}^{L_1} = h_{ij}^{L_1^+}(x_j) + h_{ij}^{L_1^-}(x_j), \quad \text{and} \tag{3.1.24}$$

$$h_{ij}^{L_2} = h_{ij}^{L_1^+}(x_j) + h_{ij}^{L_2^-}(x_j). \tag{3.1.25}$$

3.1.2.2 Linear relaxation for a normal set

For a normal set, we construct a linear function $g_{ij}^{L_p} : \mathbb{R} \mapsto \mathbb{R}$ such that

$$g_{ij}^{L_p}(x_j) \leq g_{ij}(x_j), \quad \forall\, x_j \in [x_j^L,\ x_j^U]. \tag{3.1.26}$$

Proposition 3.2. *For a given i, if linear functions $g_{ij}^{L_p} : \mathbb{R} \mapsto \mathbb{R}$ satisfy the inequality* (3.1.26) *for all $j = 1, \ldots, n$, then the linear inequality,*

$$\sum_{j=1}^{n} g_{ij}^{L_p}(x_j) \leq 0, \tag{3.1.27}$$

is a valid outer approximation of a constraint (3.1.17)*.*

Following the foregoing, we construct the linear relaxation of g_{ij}^+ and g_{ij}^-. The only difference compared to previous is that, for g_{ij}^+, two tangent lines corresponding to x_j^L and x_j^U are used, and, for g_{ij}^-, we construct a line crossing two points $(x_j^L,\ g_{ij}^-(x_j^L))$ and $(x_j^U,\ g_{ij}^-(x_j^U))$. Let u_{ij}^+ and l_{ij}^+ be the corresponding function values of $g_{ij}^+(x_j^U)$ and $g_{ij}^+(x_j^L)$, respectively, and let u_{ij}^- and l_{ij}^- be the corresponding function values of $g_{ij}^-(x_j^U)$ and $g_{ij}^-(x_j^L)$, respectively. Let ∇g_{ij}^{L+} be the gradient at $x_j = x_j^L$ such that $\nabla g_{ij}^{L+} = \nabla g_{ij}^+(x_j^L)$ and ∇g_{ij}^{U+} be the gradient at $x_j = x_j^U$ such that $\nabla g_{ij}^{U+} = \nabla g_{ij}^+(x_j^U)$.

- Let $g_{ij}^{L_1^+}$ and $g_{ij}^{L_2^+}$ be linear relaxations of g_{ij}^+ such that

$$g_{ij}^{L_1^+}(x_j) = \nabla g_{ij}^{L+} x_j + l_{ij}^+ - \nabla g_{ij}^{L+} x_j^L, \quad \text{and} \tag{3.1.28}$$

$$g_{ij}^{L_2^+}(x_j) = \nabla g_{ij}^{U+} x_j + u_{ij}^+ - \nabla g_{ij}^{U+} x_j^U. \tag{3.1.29}$$

- Let $g_{ij}^{L_1^-}$ be a linear relaxation of h_{ij}^- such that

$$g_{ij}^{L_1^-}(x_j) = \frac{u_{ij}^- - l_{ij}^-}{x_j^U - x_j^L} x_j + u_{ij}^- - \frac{u_{ij}^- - l_{ij}^-}{x_j^U - x_j^L} x_j^U. \tag{3.1.30}$$

Thus, two possible linear relaxations of g_{ij} are

$$g_{ij}^{L_1} = g_{ij}^{L_1^+}(x_j) + g_{ij}^{L_1^-}(x_j), \quad \text{and} \tag{3.1.31}$$

$$g_{ij}^{L_2} = g_{ij}^{L_2^+}(x_j) + g_{ij}^{L_1^-}(x_j). \tag{3.1.32}$$

3.2 Computational Experiments

3.2.1 Test problems

For computational experiments, we consider test problems of the following form:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t} \quad & \sum_j^n a_{ij}(x_j + b_{ij})^{k_{ij}} \geq q_i, \quad \text{for } i = 1, \ldots, m, \\ & x \in [0,\ x^U], \end{aligned}$$

where $c \in \mathbb{R}^n$, $a_{ij}, b_{ij}, q_i \in \mathbb{R}$, and $k_{ij} \in \mathbb{Z}_+$ are randomly generated parameters. The test problems are generated as follows:

1. Each component of c and each a_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ are independently sampled from a uniform distribution $U[0,\ 20]$.

2. Each b_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ is randomly generated with a uniform distribution $U[-30,\ 30]$.

3. Each k_{ij} for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ is randomly chosen from the integer set $\{0,\ 1,\ 3,\ 5,\ 7,\ 9\}$.

4. To avoid infeasible instances, we modify the RHS q_i so that a test problem contains one predefined feasible solution (i.e., a point $\bar{x}_j = 10$ for all j is feasible for any generated problems). For example, if $\sum_j^n a_{ij}(\bar{x}_j + b_{ij})^{k_{ij}}$ is greater than or equal to zero, we set q_i as 0. Otherwise, q_i is set as $\sum_j^n a_{ij}(\bar{x}_j + b_{ij})^{k_{ij}} + q_i'$, where q_i' is randomly chosen from a uniform distribution $U[-10, 0]$.

5. For initial bounds of the x_j variables, x_j^U is set at 20 for $j = 1, \ldots, n$.

In order to construct a linear relaxation of a generated problem according to the method described in 3.1.2, we should expand each polynomial algebraically. Rather than following the previous method, we construct its own linear relaxation. Let $h_{ij}(x_j)$ be the polynomial corresponding to variable x_j in the i^{th} constraint such that $h_{ij}(x_j) = a_{ij}(x_j + b_{ij})^{k_{ij}}$. By the generation rule of each parameters, h_{ij} is a non-decreasing polynomial, but it has only one inflection point at $x_j = -b_{ij}$. (The gradient of h_{ij} is $\nabla h_{ij}(x_j) = a_{ij}k_{ij}(x_j + b_{ij})^{k_{ij}-1}$. For $a_{ij}k_{ij} > 0$, the gradient $\nabla h_{ij}(x_j)$ can be zero at $x_j = -b_{ij}$ because $k_{ij} - 1$ is an even integer.) For the given $x_j \in [x_j^L,\ x_j^U]$, we consider three cases: (a) $x_j^L \geq -b_{ij}$, (b) $x_j^U \leq -b_{ij}$, and (c) $x_j^L \leq -b_{ij} \leq x_j^U$. Figure 9 depicts four possible linear relaxations.

Let u_{ij} and l_{ij} be the function values corresponding to x_j^U and x_j^L, respectively. Let ∇h_{ij}^U and ∇h_{ij}^L be the gradients at x_j^U and x_j^L, respectively. Consider the following cases.

(a) $x_j^L \geq -b_{ij}$.

In this case, only one linear relaxation is possible.(see, Figure 9 (a).) The linear relaxation $h_{ij}^{L_1}$ is

$$h_{ij}^{L_1}(x_j) = \frac{u_{ij} - l_{ij}}{x_j^U - x_j^L} x_j + u_{ij} - \frac{u_{ij} - l_{ij}}{x_j^U - x_j^L} x_j^U.$$

(b) $x_j^U \leq -b_{ij}$.

Two tangent lines $h_{ij}^{L_2}$ and $h_{ij}^{L_3}$ are constructed for linear relaxations

$$\begin{aligned} h_{ij}^{L_1}(x_j) &= \nabla h_{ij}^L x_j + l_{ij} - \nabla h_{ij}^L x_j^L \quad \text{and} \\ h_{ij}^{L_2}(x_j) &= \nabla h_{ij}^U x_j + u_{ij} - \nabla h_{ij}^U x_j^U. \end{aligned}$$

(c) $x_j^L \leq -b_{ij} \leq x_j^U$.

First, the tangent line $h_{ij}^{L_t}$ at $x_j = x_j^L$ is constructed as follows:

$$h_{ij}^{L_t}(x_j) = \nabla h_{ij}^L x_j + l_{ij} - \nabla h_{ij}^L x_j^L.$$

If $h_{ij}^{L_t}(x_j^U) \geq u_{ij}$, then it is a valid relaxation (see, Figure 9 (c)). We set $h_{ij}^{L_1}(x_j^U)$ equal to $h_{ij}^{L_t}(x_j^U)$. The line containing the two points $(x_j^L, 0)$ and (x_j^U, u_{ij}) is used as the second relaxation:

$$h_{ij}^{L_2}(x_j) = \frac{u_{ij}}{x_j^U - x_j^L} x_j + u_{ij} - \frac{u_{ij}}{x_j^U - x_j^L} x_j^U.$$

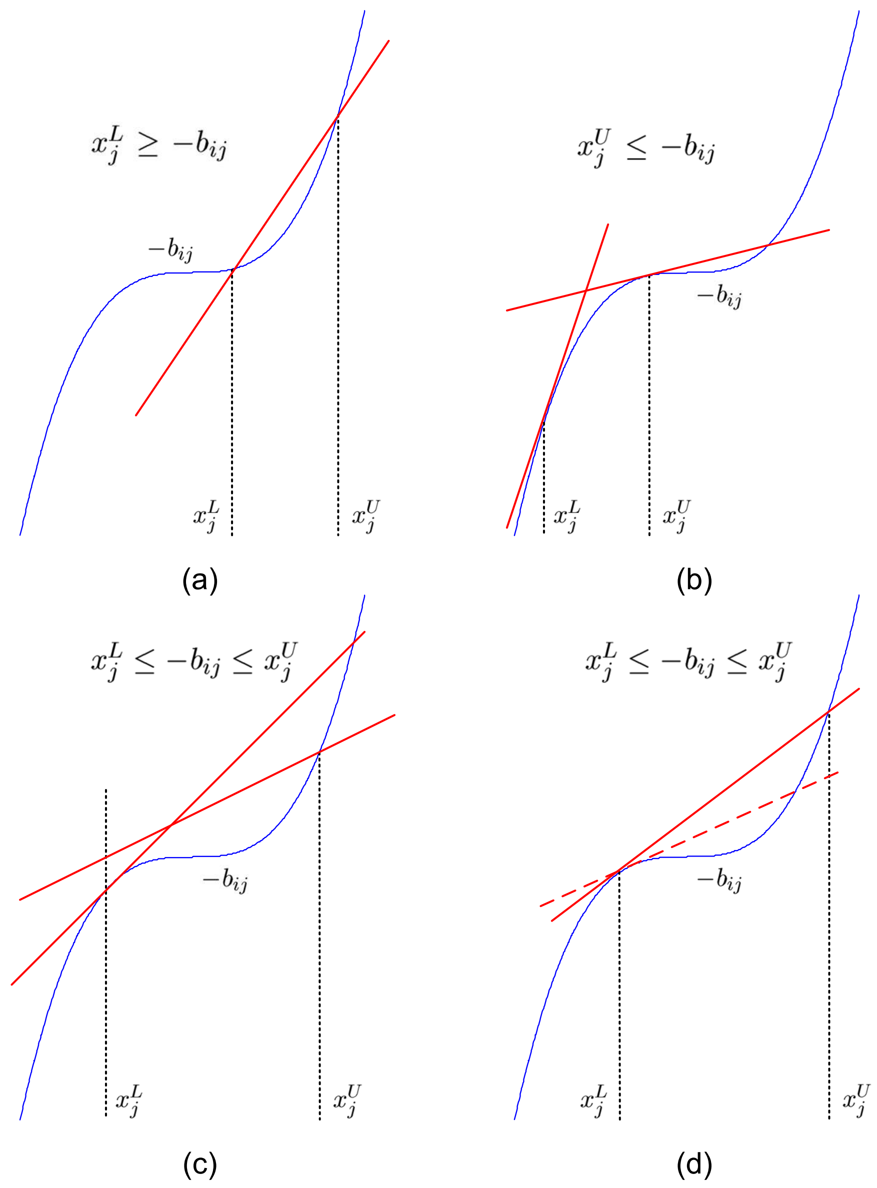


Figure 9: Linear relaxations of h_{ij} of a generated problem: (a) $x_j^L \geq -b_{ij}$, (b) $x_j^U \leq -b_{ij}$, and (c)&(d) $x_j^L \leq -b_{ij} \leq x_j^U$.

Consider the second case that $h_{ij}^{L_t}(x_j^U) < u_{ij}$ (see, Figure 9 (d)). Then, $h_{ij}^{L_t}(x_j)$ is not a valid relaxation. We construct a new relaxation $h_{ij}^{L_1}(x_j)$ as follows:

$$h_{ij}^{L_1}(x_j) = \frac{u_{ij} - l_{ij}}{x_j^U - x_j^L} x_j + u_{ij} - \frac{u_{ij} - l_{ij}}{x_j^U - x_j^L} x_j^U .$$

We generate at most two linear relaxations of a polynomial for a variable in a constraint. Suppose we have n variables. Then, All combinations of each linear relaxation of a variable are linear relaxations of a constraint. Thus, we might have at most 2^n linear constraints for one original constraint.

3.2.2 Bounding by variable fixing

For separable polynomial programming problems, it is also possible to establish linear relaxations for a given partition. For some problems, we might not be able to establish reasonable convex relaxations or we might have too weak relaxations. We are able to calculate a lower bound for a given partition by subdividing the variables. For a given partition $[x^L,\ x^U]$, we can consider the following problem

$$\begin{aligned} \text{(MO/H):} \quad \min_{x} \quad & f(x) && (3.2.1)\\ \text{s.t.} \quad & h_i(x) \geq 0 \quad \text{for } i = 1, \ldots, m_H, && (3.2.2)\\ & x \in [x^L,\ x^U]. && (3.2.3) \end{aligned}$$

Clearly, the problem (MO/H) is a valid relaxation of the problem (MO) because we relax the normal set. We solve this problem for a certain number of iterations by one of the algorithms proposed. If we have an optimal solution for the given partition, we can remove the partition from further consideration after updating the feasible solution. If we fail to get an optimal solution within a predetermined number of iterations or a time limit, we can use the current best lower bound as the lower bound for the given partition. However, branch-and-bound based algorithms do not show good performance on higher dimensional problems. For separable problems, we can partition the decision vector into two disjoint sets of variables. Some of the variables remain as free variables and we fix all other variables. We fix most of the variables in the reverse normal constraints at their upper bounds and remove them from the objective function. For example, we may take the first three variables (x_1, x_2, x_3) as our free variables. In the reverse normal constraints, we set the fixed variables to their upper bounds (i.e., $x_j = x_j^U$ for $j = 4, \ldots, n$). We minimize an objective function that includes only variables x_1, x_2 and x_3. The subproblem can be written as follows:

$$\begin{aligned} (\text{SPP}_K): \quad & \min_{\{x_i : i \in K\}} && \sum_{j \in K} f_j(x_j) && (3.2.4) \\ & \text{s.t.} && \sum_{j \in K} h_{ij}(x_j) + \sum_{j \notin K} h_{ij}(x_j^U) \geq 0 \quad \text{for } i = 1, \ldots, m, && (3.2.5) \\ & && x \in [x^L,\ x^U], && (3.2.6) \end{aligned}$$

where K is a set of indices corresponding to the free variables we consider.

Proposition 3.3. *If K_j^* is an optimal value of (SPP$_{K_j}$) and $K_i \cap K_j = \emptyset$ for $i \neq j$, then $\sum_j K_j^*$ is a valid lower bound of (SPP).*

Proof. Suppose that there exists an optimal solution x^* such that $f(x^*) < \sum_j K_j^*$. It follows that there exists $\sum_{k \in K_j} f_k(x_k^*) < \sum_{k \in K_j} f_k(\widehat{x}_k)$ for some j, where $\widehat{x}$ is an optimal solution of (SPP$_{K_j}$). We consider the reverse normal constraints.

$$\begin{aligned} 0 &\leq \sum_{k \in K_j} h_{ik}(x_k^*) + \sum_{k \notin K_j} h_{ik}(x_k^*) && (3.2.7) \\ &\leq \sum_{k \in K_j} h_{ik}(x_k^*) + \sum_{k \notin K_j} h_{ik}(x_k^U) \quad \text{by the monotonicity of } h_i && (3.2.8) \end{aligned}$$

That is, $x^* \in K_j$ is also a feasible point in (SPP$_{K_j}$). It contradicts that $\widehat{x}$ is an optimal solution of K_j. Therefore, the proposed bound is valid. □

In separable problems, we may easily apply this method. However, the tightness of the bound obtained will depend on how we partition the variables. As we increase the number of variables in a set K, we have tighter bounds at the expense of higher computational times for solving subproblems. We compare the performance of this method in section 3.2.3.

3.2.3 Computational Results

In this section, we report on the computational performance of the Polyblock Algorithm, the modified Polyblock Algorithm, and the branch-and-bound scheme (with and without linear convex relaxations proposed previously) on randomly generated separable polynomial programming problems.

The proposed algorithms were implemented in C++ and the linear relaxation was solved using concert technology with CPLEX 8.0. All computations were on an UltraSparc-III-Cu UNIX workstation 2x900 MHz CPUs with 2 GB RAM.

3.2.3.1 Polyblock versus branch-and-bound with or without convex relaxation

We considered 9 different experimental conditions by varying the number of problem variables equal to 3, 4, or 5, and the termination tolerance $\epsilon = 0.01, 0.001, 0.0001$. We generated 10 random instances for each experimental condition[1]. Thus, 90 problems were solved. In order to compare the performance of algorithms, we solve each instance by 3 algorithms — Polyblock Algorithm (Algorithm 1), modified Polyblock Algorithm (Algorithm 2) and branch-and-bound algorithm (Algorithm 3) with or without convex relaxation (i.e., the linear relaxation proposed in 3.1.2). A time limit of 300 CPU seconds is imposed on the runs. The CPU times and optimal values are reported in Appendix A.1.

Table 2: Average CPU seconds.
('PA', 'mPA' and 'BNB' denote Polyblock Algorithm (Algorithm 1), modified Polyblock Algorithm (Algorithm 2) and branch-and-bound Algorithm (Algorithm 3).)

	3 variables case			4 variables case			5 variables case		
Alg.	0.01	0.001	0.0001	0.01	0.001	0.0001	0.01	0.001	0.0001
PA	0.05	0.57	0.53	16.96	89.08	122.52	255.12	302.52	270.46
mPA	0.01	0.04	0.21	0.21	3.85	43.83	213.75	271.06	256.26
BNB	0.01	0.02	0.09	0.05	0.29	4.18	0.35	153.50	92.28

(a) Without convex relaxations.

	3 variables case			4 variables case			5 variables case		
Alg.	0.01	0.001	0.0001	0.01	0.001	0.0001	0.01	0.001	0.0001
PA	0.08	0.26	0.31	2.44	12.89	9.23	173.71	269.60	180.67
mPA	0.05	0.07	0.17	0.43	1.59	1.45	102.94	169.60	103.79
BNB	0.02	0.04	0.09	0.05	0.17	0.18	0.19	0.68	0.56

(b) With convex relaxations.

Table 2 reports the average CPU seconds over 10 instances for each problem condition

[1]For each condition, we generated 5 instances with 5 constraints and 5 instances with 7 constraints. We noticed that the number of constraints did not have a key effect on the computational time. Thus, we omitted that classification.

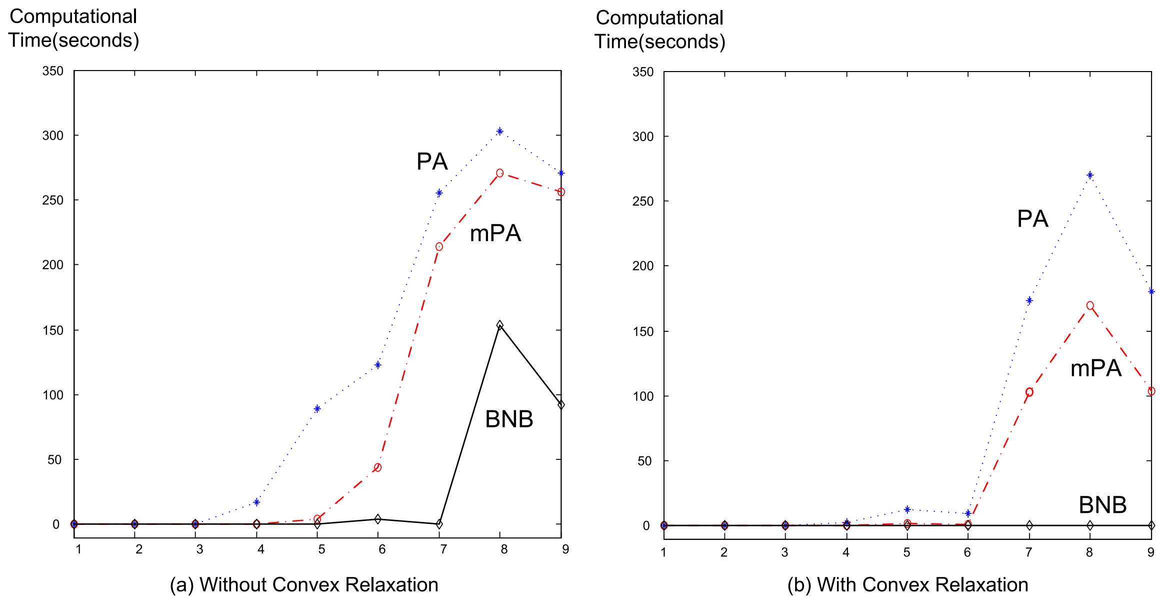


Figure 10: Average CPU seconds in various cases

and each algorithm. Notice that as the number of variables are increased and the termination tolerance is decreased, more computational time is required. Figure 10 clearly shows that the branch-and-bound algorithm (Algorithm 3) shows better performance than the others in terms of CPU seconds, and that the convex relaxations help to reduce the computational time when problems are relatively harder. For the Polyblock Algorithm and the modified Polyblock Algorithm, there exist overlapping areas among blocks. That is, there exist several blocks that contain an ϵ-optimal solution. Even though we found an optimal solution in the middle of the procedure, we should refine those blocks many times in order to prove that one is an ϵ-optimal solution. In the branch-and-bound algorithm, there might exist overlapping areas. However, those are relative small region. In higher dimensional problems, these overlapping areas will grow exponentially. As the dimension of a problem is increased, we may observe that the computational time required to solve a problem with the Polyblock Algorithm or the modified Polyblock Algorithm grows faster than the time required by the branch-and-bound algorithm.

Figure 11 compares CPU seconds of the branch-and-bound algorithm with and without convex relaxations for lower dimensional instances and higher dimensional instances. We use

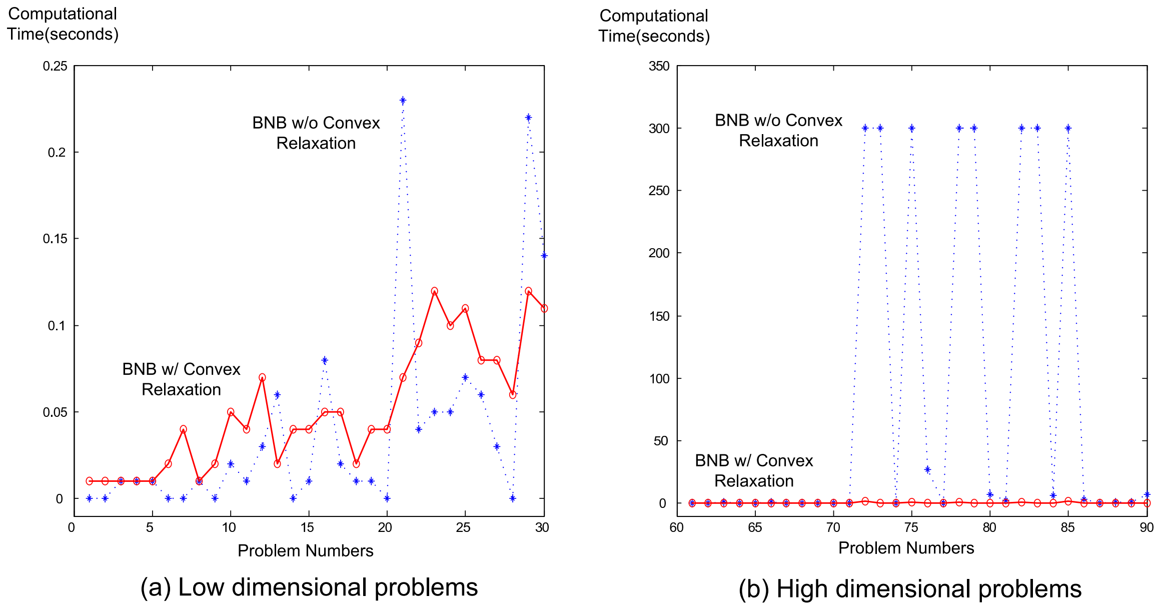


Figure 11: CPU seconds with or without convex relaxations for (a) low dimensional instances and (b) high dimensional instances.

convex relaxations to get a tighter bound for the given block in the Polyblock and modified Polyblock Algorithms and the given partition in the branch-and-bound algorithm. Tighter bounds help us to identify some blocks or partitions that do not contain a better solution. It follows that we can fathom those blocks or partitions. However, in order to obtain a tighter bound, we should solve an additional optimization problem. That is, it requires additional computational time while it decreases overall computational time by fathoming some blocks or partitions in early stages. We observe that for easy problems (e.g., lower dimensional problems (see, Figure 11 (a))), convex relaxations do not decrease computational time dramatically, while they do decrease the computational time for relatively hard problems (e.g. higher dimensional problems (see, Figure 11 (b))).

In terms of speed of algorithm, we cannot say that one is better than the other. The performance of each algorithm varies with the problems. However, according to average computational results, the branch-and-bound algorithm with convex relaxations shows better performance than the others for separable polynomial programming problems.

3.2.3.2 Conical algorithm versus branch-and-bound algorithm

We described a conical algorithm in section 2.5. We generated 60 problems under 6 problem conditions. The conditions are determined by the combinations of the number of variables $\{5, 7\}$ and termination tolerance $\{0.01, 0.001, 0.0001\}$. Each problem was solved by the conical algorithm and the branch-and-bound algorithm. In Appendix A.1.2, we report CPU seconds, required iterations and the maximum number of partitions considered.

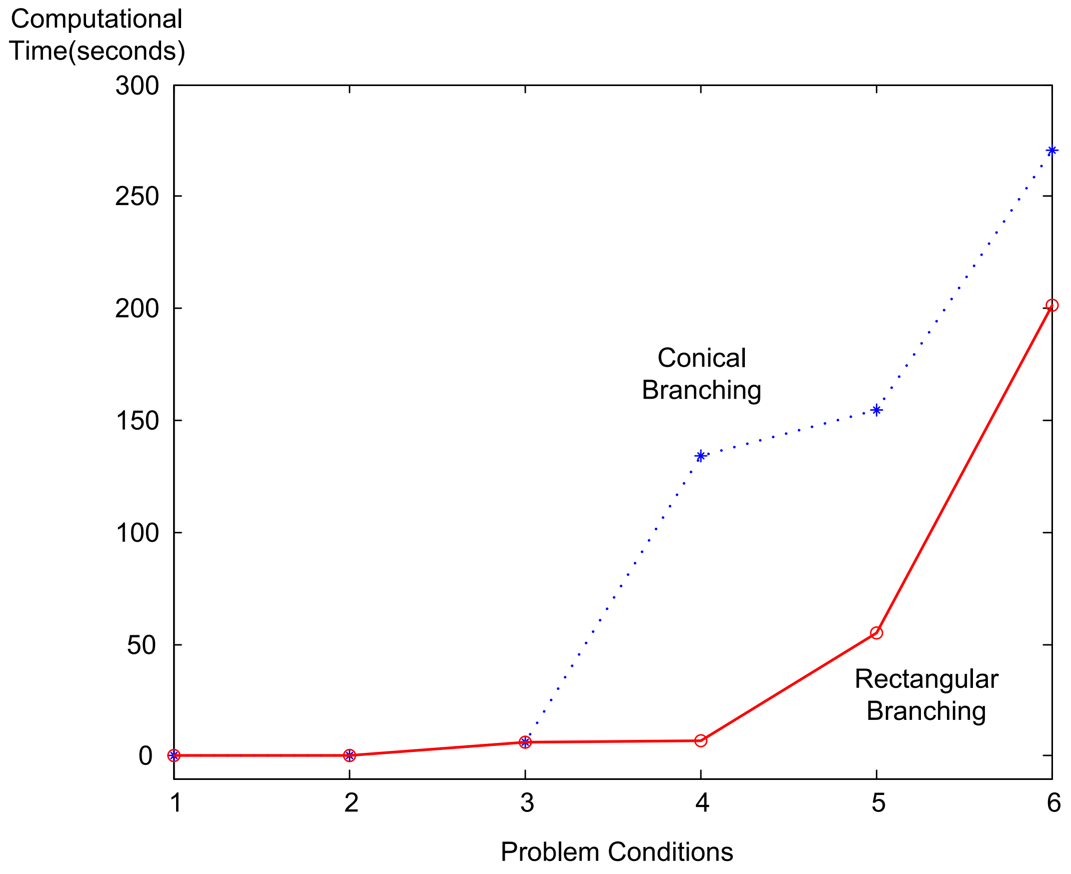


Figure 12: Average CPU seconds comparison

In Figure 12, we plot the average CPU time required over 10 instances of each condition. According to Figure 12 the rectangular branch-and-bound algorithm shows better performance than the conical algorithm on separable polynomial programming problems. Table 3 summarizes the comparative performance. The results show that the rectangular algorithm is faster than the conical algorithm for 27 instances while the conical algorithm is faster for 23 instances. However, the conical algorithm shows better performance for small dimensional problems. In our implementation, we set a CPU second limit of 300 seconds. In 18 instances, the conical algorithm failed to terminate at an optimal solution while the rectangular algorithm failed for 6 instances.

Table 3: Computational result summary

	Conical algorithm	Rectangular algorithm
Fails	18	6
Faster than the others	23	27

According to our computational results, the conical algorithm shows better performance than the rectangular algorithm on small dimensional problems while the branch-and-bound algorithm does better on relatively high dimensional problems.

3.2.3.3 Bounding by variable fixing

In this section, we report the results of comparing the average CPU seconds required by the branch-and-bound algorithm alone, the branch-and-bound with linear relaxation, and the branch-and-bound with bounds provided by the Polyblock Algorithm[2] for 90 instances. The detailed results for these experiments is reported in Appendix A.1.3. We have seen that the Polyblock Algorithm outperforms the others on small dimension problems (i.e., $n \leq 3$). Thus, we separated variables into several sets K_j such that the cardinality of K_j is less than or equal to 3.

Figure 13 shows the average computational time for each condition. In these instances, the branch-and-bound algorithm with linear relaxation outperforms the others. As we provide a tighter bound, we can eliminate some partitions earlier than regular branch-and-bound does. It follows that we carry only a small number of partitions under consideration. It helps us to reduce computational time for sorting the list of partitions and thus save memory space. When we do not have reasonable relaxations for separable problems or decomposable problems, we might use the subdivision bounding scheme proposed in section

[2]We set a time limit of 10 seconds. That is, if we fail to get an optimal solution for a given subregion, we use a least lower bound of the list of blocks in the Polyblock Algorithm as a lower bound of a given subregion.

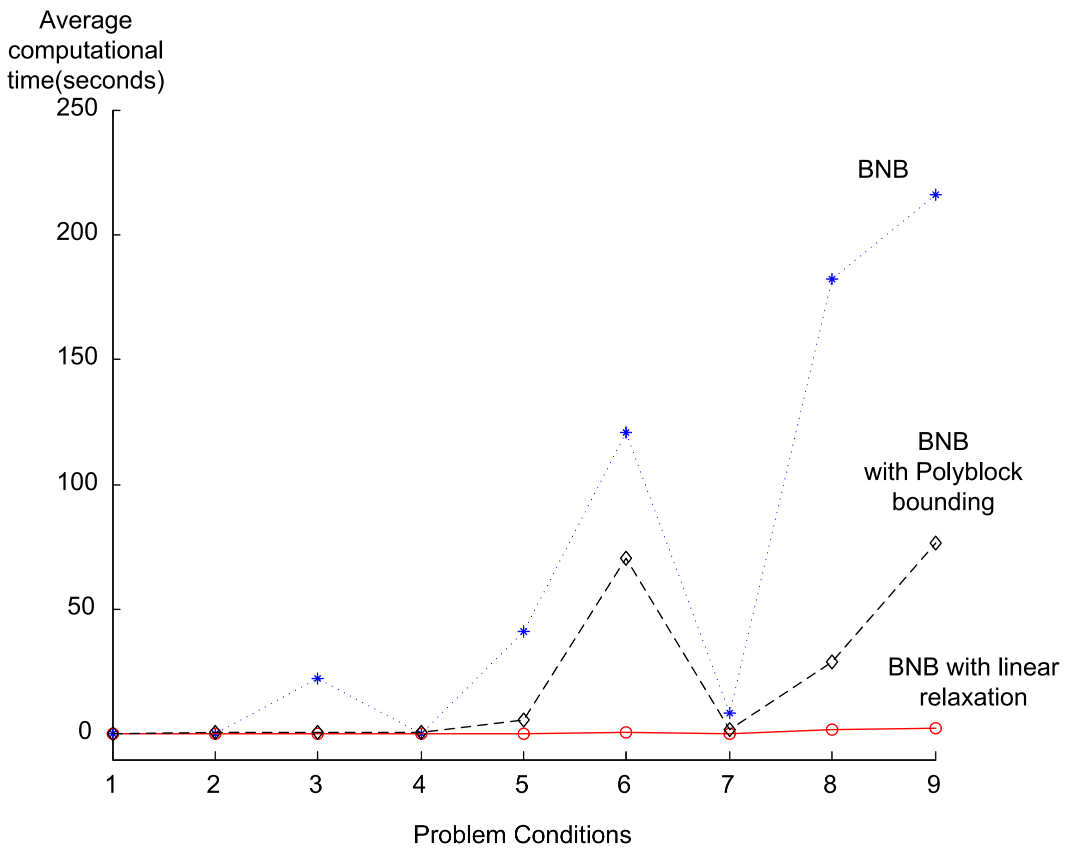


Figure 13: Average CPU seconds comparison of branch-and-bound, branch-and-bound with linear relaxation, branch-and-bound with Polyblock bounding

3.2.2.

CHAPTER IV

PROBABILISTICALLY CONSTRAINED LINEAR PROGRAMS

4.1 Introduction

Various applications in reliability and risk management (cf.[20]) give rise to probabilistically-constrained linear programs (PCLP) of the following form

$$\min_{x} \quad c^T x \tag{4.1.1}$$

$$\text{s.t.} \quad Ax = b \tag{4.1.2}$$

$$\mathbb{P}\{Tx \geq \xi(\omega)\} \geq \alpha \tag{4.1.3}$$

$$x \geq 0. \tag{4.1.4}$$

In the above model, $x \in \mathbb{R}^n$ is a vector of decision variables; the parameters $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{p \times n}$, $b \in \mathbb{R}^p$ and $T \in \mathbb{R}^{m \times n}$ represent deterministic problem data; ω is a random vector from the probability space $(\Omega, \Sigma, \mathcal{P})$ and $\xi : \Omega \mapsto \mathbb{R}^m$ represent stochastic right-hand-side parameters; $\mathbb{P}\{S\}$ denotes the probability of the event $S \in \Sigma$ under the probability measure $\mathcal{P}$; and $\alpha \in (0, 1)$ is a scalar. The probabilistic constraint (4.1.3) requires that the inequalities $Tx \geq \xi(\omega)$, involving random data, hold with a probability of at least α.

Beginning with the seminal work of Charnes and Cooper [4], many different solution techniques have been proposed for different versions of PCLP. Most of the existing approaches rely on specific assumptions on the distribution of the stochastic parameters that render convex the feasible region defined by the probabilistic constraints (4.1.3). For example, if the stochastic parameters have a continuous log-concave distribution, then PCLP is guaranteed to have a convex feasible set [18], and hence may be solvable with standard convex programming techniques (see [20] for a comprehensive review). For general distributions, in particular for discrete distributions, the feasible region defined by (4.1.3) is non-convex. This non-convexity is illustrated with the following simple example from [22] (a similar example is presented in [25]).

Example 1: Consider the following PCLP:

$$\begin{aligned} \min_{x_1,x_2} \quad & c_1x_1 + c_2x_2 \\ \text{s.t.} \quad & \mathbb{P}\left\{ \begin{array}{c} x_1 + x_2 \geq \xi_1 \\ x_1 + 3x_2 \geq \xi_2 \end{array} \right\} \geq 0.5 \\ & x_1,\ x_2 \geq 0, \end{aligned}$$

where ξ_1 and ξ_2 are random variables with the joint distribution $\mathbb{P}\{\xi_1 = 2, \xi_2 = 4\} = 0.5$ and $\mathbb{P}\{\xi_1 = 3, \xi_2 = 0\} = 0.5$. The feasible region is given by union *of the polyhedra*

$$\begin{aligned} P_1 = \quad & \{(x_1, x_2) \in \mathbb{R}^2_+ :\ x_1 + x_2 \geq 2, x_1 + 3x_2 \geq 4\} \quad \text{and} \\ P_2 = \quad & \{(x_1, x_2) \in \mathbb{R}^2_+ :\ x_1 + x_2 \geq 3, x_1 + 3x_2 \geq 0\}, \end{aligned}$$

and is clearly non-convex.

For discrete distributions, a PCLP can be immediately reformulated as a mixed-integer linear program (milp) (see [20]). Ruszczyński [21] has developed specialized cutting planes for this milp reformulation, and embedded these within a branch-and-cut algorithm. Unless the random variables are independent, the milp reformulation involves a huge number of binary variables – one for every possible joint realization of the random parameters – and may be computationally impractical in general. An improved milp formulation for the PCLP can be constructed if the set of p-efficient points (PEPs) can be identified. A point $z \in \mathbb{R}^m$ is said to be a PEP for the random variable $\xi(\omega)$ if $\mathbb{P}\{z \geq \xi(\omega)\} \geq \alpha$ and there is no y such that $y \leq z$, $y \neq z$ and $\mathbb{P}\{y \geq \xi(\omega)\} \geq \alpha$[19]. For discrete distributions with certain concavity properties, Dentcheva et al. [5, 6] have developed efficient convex programming relaxations for PEP-based milp formulations for PCLPs. Such relaxations have been used within exact branch-and-bound algorithms for some classes of probabilistically constrained milps [1, 2]. Sen [24] has suggested convex relaxations for PCLPs with general discrete distributions using disjunctive programming techniques.

In this chapter, we develop an algorithm for obtaining global optimal solutions to PCLPs.

Unlike prior work, we do not make any concavity or continuity assumptions on the underlying distributions. The proposed algorithm is a branch-and-bound scheme that searches for a global solution by successively partitioning the non-convex feasible region and by using bounds on the objective function to fathom inferior partitions. The basic scheme is enhanced by domain reduction and cutting plane strategies to reduce the size of the partitions and hence tighten bounds. We refer to this strategy as branch-reduce-cut. The proposed method exploits the monotonicity properties inherent in the problem, and requires solving linear programming subproblems. Convergence analysis of the algorithm in case of discrete and continuous distributions is provided. We also present numerical results for PCLPs involving discrete distributions and demonstrate that the proposed algorithm is significantly superior to a straight-forward milp approach.

The remainder of this chpater is organized as follows. In Section 4.2, we reformulate PCLP to reveal the inherent monotonicity in the problem. The proposed branch-reduce-cut algorithm is developed in Section 4.3, and its convergence analysis is presented in Section 4.4. Finally, in Section 4.5 we present some numerical results with the proposed algorithm on randomly generated PCLP instances involving discrete distributions.

4.2 Problem reformulation and structural properties

Consider the following problem:

$$\begin{aligned} \min_{y} \quad & f(y) && (4.2.1) \\ \text{s.t.} \quad & y \in G \cap H, && (4.2.2) \end{aligned}$$

where $y \in \mathbb{R}^m$ is a vector of decision variables; $f : \mathbb{R}^m \mapsto \mathbb{R} \cup \{-\infty, +\infty\}$ is the linear programming value function

$$f(y) = \min_{x}\{c^T x : \ Ax = b, Tx \geq y, x \geq 0\} \tag{4.2.3}$$

(we let $f(y) = -\infty$ and $f(y) = +\infty$ when the linear program (4.2.3) is unbounded and infeasible, respectively); the set G is the set of y's for which the linear program (4.2.3) is feasible, i.e.,

$$G = \{y \in \mathbb{R}^m : \ f(y) < +\infty\}; \tag{4.2.4}$$

and the set H is defined as

$$H = \{y \in \mathbb{R}^m : \; F(y) \geq \alpha\}, \tag{4.2.5}$$

where $F : \mathbb{R}^m \mapsto [0, 1]$ is the cumulative density function of the random vector $\xi(\omega)$, i.e., $F(y) = \mathbb{P}\{y \geq \xi(\omega)\}$.

The following result establishes the equivalence between PCLP (4.1.1)-(4.1.4) and the problem (4.2.1)-(4.2.2). The proof is straight-forward and is omitted.

Proposition 4.1.

(i) If x^* is an optimal solution of the PCLP (4.1.1)-(4.1.4), then $y^* = Tx^*$ is an optimal solution of (4.2.1)-(4.2.2), and both problems have the same optimal objective function value.

(ii) If y^* is an optimal solution of (4.2.1)-(4.2.2), then $x^* \in \operatorname{argmin}\{c^T x : \; Ax = b, Tx \geq y^*, x \geq 0\}$ is an optimal solution of the PCLP (4.1.1)-(4.1.4), and both problems have the same optimal objective function value.

Using Proposition 4.1, we consider solving the reformulated PCLP (4.2.1)-(4.2.2) throughout the remainder of this paper. Henceforth, the acronym PCLP will refer to the reformulation (4.2.1)-(4.2.2). The following assumptions are required to ensure that PCLP is well-defined.

Assumption 4.1. *The set of dual solutions to the linear program (4.2.3) is non-empty, i.e.,*

$$\{(\pi, \rho) \in \mathbb{R}^{p+m} : \; \pi A + \rho T \leq c, \rho \geq 0\} \neq \emptyset.$$

By weak duality, the above assumption guarantees that $f(y) > -\infty$ for all y.

Assumption 4.2. *The constraint (4.2.2) in PCLP can be replaced by*

$$y \in G \cap H \cap [y^L, y^U],$$

for some $y^L \leq y^U$.

The above assumption is required to make the feasible region bounded.

Proposition 4.2 (see, e.g., [3]). *Under Assumption 4.1,*

(i) the set G is non-empty, closed and convex, and

(ii) the function f is continuous, piece-wise linear, convex, and non-decreasing over G.

Proposition 4.3. *Under Assumptions 4.1 and 4.2, if the feasible region of the PCLP (4.2.1)-(4.2.2) is non-empty, then there exists an optimal solution.*

Proof. The set H is closed due to the upper semi-continuity of the cumulative density function F. Thus the feasible region of PCLP is compact, and the objective function is continuous. The result then follows from the Weirstrass Theorem. $\square$

Note that the set G satisfies the following property

$$x \leq y \text{ and } y \in G \Rightarrow x \in G.$$

Such a set is called a *normal* set [32]. Owing to the non-decreasing property of F, the set H satisfies

$$x \geq y \text{ and } y \in H \Rightarrow x \in H.$$

Such a set is called a *reverse normal* set [32]. Thus the problem (4.2.1)-(4.2.2) involves minimizing a non-decreasing function over the intersection of a normal and a reverse normal set. Such problems belong to the class of non-convex monotonic optimization problems recently studied by Tuy [32], Li et al. [12] and Toh [31]. Following are some important properties adapted to our setting (the proofs follow immediately from the general results in [32]).

Proposition 4.4. *Under Assumptions 4.1 and 4.2,*

(i) if the problem is feasible then $y^L \in G$ and $y^U \in H$;

(ii) if $y^L \in H$, then either y^L is an optimal solution or the problem is infeasible;

(iii) if $y^L \notin H$ and the problem is feasible, then there exists an optimal solution on the relative boundary of H;

(iv) if $\widehat{y} \in [y^L, y^U]$ is on the relative boundary of H, then there cannot be a solution that is better (i.e., feasible with a smaller objective value) in the sets $Q_A := \{y : \ y^L \leq y < \widehat{y}\}$ and $Q_B := \{y : \ \widehat{y} \leq y \leq y^U\}$. Consequently, the sets Q_A and Q_B can be removed from further consideration.

In the case of a (finite) discrete distribution of the random vector $\xi(\omega)$, PCLP has some additional properties. Suppose $\xi(\omega)$ has K possible realizations $\{\xi^1, \dots, \xi^K\}$ with probabilities $\{p^1, \dots, p^K\}$. Let

$$\Xi_j = \cup_{k=1}^{K} \{\xi_j^k\} \text{ for } j = 1, \dots, m \ \text{ and } \ \mathcal{C} = \prod_{j=1}^{m} \Xi_j. \tag{4.2.6}$$

Note that the set $\mathcal{C}$ is finite.

Lemma 4.1. *If $\xi(\omega)$ has a discrete distribution, then for any $y \in \mathbb{R}^m$ such that $y \in H$, there exists $\widehat{y}$ such that*

$$\widehat{y} \leq y, \ \widehat{y} \in \mathcal{C} \text{ and } \widehat{y} \in H.$$

Proof. Let $\mathcal{K}(y) = \{k \in \{1, \dots, K\} : \ \xi^k \leq y\}$, then $y \in H \Leftrightarrow \sum_{k \in \mathcal{K}(y)} p^k \geq \alpha$. Let $\widehat{y}_j = \max_{k \in \mathcal{K}(y)} \{\xi_j^k\}$ for $j = 1, \dots, m$. Then $\widehat{y} \leq y$, $\widehat{y} \in \mathcal{C}$, and $\widehat{y} \geq \xi^k$ for all $k \in \mathcal{K}(y)$, thus $\widehat{y} \in H$. □

Proposition 4.5. *If $\xi(\omega)$ has a discrete distribution and PCLP has an optimal solution, then there exists an optimal solution $y^* \in \mathcal{C}$.*

Proof. Let $y' \in H \cap G$ be any optimal solution of PCLP, then by Lemma 4.1, there exists $y^* \in \mathcal{C}$ such that $y^* \leq y'$ and $y^* \in H$. Since G is a normal set, we have $y^* \in G$, thus y^* is feasible. By the monotonicity of the objective function, $f(y^*) \leq f(y')$, thus y^* is also an optimal solution. □

By virtue of Proposition 4.5, in case of discrete distributions, we can restrict our search of the optimal solution to the finite set $\mathcal{C}$ intersected with $G \cap H$.

In the following section, we develop a branch-and-bound algorithm for solving PCLP by exploiting the properties outlined above. In addition to revealing the monotonicity properties, the reformulation (4.2.1)-(4.2.2) has the added advantage (over the original formulation

(4.1.1)-(4.1.4)) that the problem dimension is m rather than n, and often $m < n$. We conclude this section with an example to illustrate the properties of the reformulation.

Example 2: Consider the following PCLP:

$$\begin{array}{ll}\min\limits_{x_1,x_2} & -x_1 - 2x_2 \\ s.t. & \mathbb{P}\left\{\begin{array}{c} -x_1 - x_2 \geq \xi_1 \\ x_1 + \frac{1}{2}x_2 \geq \xi_2 \end{array}\right\} \geq 0.5 \\ & x_1, x_2 \geq 0, \end{array}$$

where ξ_1 and ξ_2 are random variables with a discrete distribution consisting of the 10 realizations given in Table 4.

Table 4: Distribution of (ξ^1, ξ^2)

k	1	2	3	4	5	6	7	8	9	10
ξ_1^k	-7	-6	-6	- 5.5	-5	-3	-3	-2	0	1
ξ_2^k	1.5	1	2	3	1	1	5.5	3	1	2
p^k	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

It can be verified that the three p-efficient points of the distribution are $(-5, 3)$, $(-3, 2)$ and $(0, 1.5)$. The feasible region of PCLP is then give by the union of the polyhedra

$$\begin{aligned} P_1 &= \{(x_1, x_2) \in \mathbb{R}^2_+ : -x_1 - x_2 \geq -5,\ x_1 + \frac{1}{2}x_2 \geq 3\}, \\ P_2 &= \{(x_1, x_2) \in \mathbb{R}^2_+ : -x_1 - x_2 \geq -3,\ x_1 + \frac{1}{2}x_2 \geq 2\} \quad \textit{and} \\ P_3 &= \{(x_1, x_2) \in \mathbb{R}^2_+ : -x_1 - x_2 \geq 0,\ x_1 + \frac{1}{2}x_2 \geq 1.5\}, \end{aligned}$$

and is illustrated in Figure 14 (a).

The reformulation of the problem is

$$\begin{array}{ll}\min\limits_{y_1,y_2} & f(y_1, y_2) \\ s.t. & (y_1, y_2) \in G \cap H \cap [y^L, y^U], \end{array}$$

where $f(y_1, y_2) = \min_{x_1,x_2}\{-x_1 - 2x_2 : \ -x_1 - x_2 \geq y_1, \ x_1 + \frac{1}{2}x_2 \geq y_2, \ x_1 \geq 0, \ x_2 \geq 0\}$, $G = \{(y_1, y_2) : \ f(y_1, y_2) < +\infty\}$, $H = \{(y_1, y_2) : \ F(y_1, y_2) \geq \alpha\}$, $y^L = (-7, 1.5)^T$, $y^U = (1, 5.5)^T$. *The feasible region in the reformulated problem is shown in Figure 14 (b). The points of intersection of the dotted grid constitute the finite set* $\mathcal{C}$.

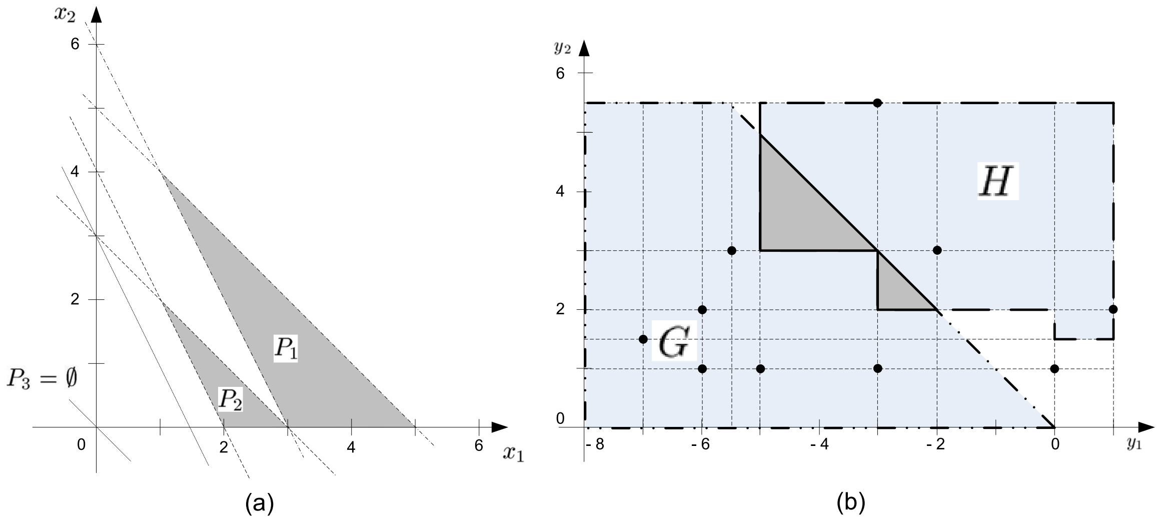


Figure 14: (a) The feasible region in the x-space, (b) The feasible region in the y-space.

4.3 A Branch-Reduce-Cut algorithm

Owing to the non-convexity of the set H (recall that the normal set G and the function f are convex), PCLP is a non-convex optimization problem. Initially proposed to deal with non-convexities arising from discreteness [10], the branch-and-bound scheme has since evolved into a general purpose global optimization technique for a wide class of non-convex problems [8, 9, 30]. The scheme proceeds by recursively partitioning (branching) the feasible region in search of a global optimal solution. For a given partition, bounds on the optimal objective value, based on efficiently solvable relaxations of the problem, are used to decide whether to examine the partition further or to fathom it, i.e., discard it from further consideration (bounding). For example, in the case of milps, linear programming relaxation based bounding and integrality-based branching rules are used (cf. [14]). For general non-convex non-linear problems, developing tight and tractable (convex) relaxations pose a crucial challenge [30]. Furthermore, since the feasible region is typically (semi)continuous, special care has to be taken to decide how a particular partition is to be further refined in order to ensure convergence.

A variety of branch-and-bound schemes has been proposed for various classes of non-convex optimization problems. These methods rely on exploiting the analytical form of the objective and constraints of the problem in order to develop convex relaxations. Several sophisticated global optimization software that automatically construct such relaxations and use these within enhanced branch-and-bound search are also available [13, 23]. Unfortunately, none of these existing global optimization methods can be used for PCLP since neither the objective function f, nor the set G, and in some cases, the set H defining the constraints is available in closed analytic form.

In this section, we exploit the monotonicity properties of PCLP outlined in the previous section to develop a specialized branch-and-bound algorithm. The algorithm recursively partitions the hyper-rectangle $[y^L, y^U]$, that contains the feasible region, into smaller and smaller hyper-rectangles. The monotonicity of the objective function f guarantees that for any hyper-rectangular partition $[a, b]$, $f(a)$ is a valid lower bound and can be used for fathoming. The monotonicity property of the set H is used for branching and to find feasible solutionss. This basic branch-and-bound scheme is enhanced by cutting plane strategies to successively approximate the set G and the set of optimal solutions contained in G. The cutting planes along with the monotonicity of the set H is used within a domain reduction step to reduce the size of the current hyper-rectangle. We refer to this algorithm as branch-reduce-cut and provide its detailed description next.

In the following description, $\mathcal{R}$ denotes a hyper-rectangular partition in $\mathbb{R}^m$, i.e. $\mathcal{R} = [a, b]$ with $a, b \in \mathbb{R}^m$ and $a \leq b$; $v(\mathcal{R}) = ||b - a||_\infty$ is a measure of the size of $\mathcal{R}$; $LB(\mathcal{R})$ denotes the lower bound on the optimal value over $\mathcal{R}$; UB denotes a global upper bound on the optimal value; $\mathcal{L}$ is a list of un-fathomed hyper-rectangles; y^* denotes the best candidate solution; $\overline{G}$ denotes a normal set approximation of the set of optimal solutions contained in G (the algorithm will ensure that this approximation is a normal set); and ϵ denotes a pre-specified tolerance. Recall that, given a vector $y \in \mathbb{R}^m$, the function $f(y)$ is evaluated by solving the linear program (4.2.3), and if $f(y) = +\infty$ then $y \notin G$. We assume that we

Initialization:

```
1: if $f(y^L) = +\infty$ or $y^U \notin H$ then
2:    STOP the problem is infeasible
3: else $\{f(y^L) < +\infty$ and $y^U \in H\}$
4:    if $y^L \in H$ then
5:       STOP $y^L$ is an optimal solution
6:    else
7:       set $\mathcal{R} = [y^L, y^U]$, $\mathcal{L} = \{\mathcal{R}\}$, $LB(\mathcal{R}) = f(y^L)$, $UB = +\infty$, $y^* = \emptyset$, and $\overline{G} = \mathbb{R}^m$
8:    end if
9: end if
```

Main loop:

```
1: while $\mathcal{L} \neq \emptyset$ do
2:    select $\mathcal{R} \in \mathcal{L}$ such that $LB(\mathcal{R}) = \min_{\mathcal{R}' \in \mathcal{L}}\{LB(\mathcal{R}')\}$
3:    branch $\mathcal{R}$ into $\mathcal{R}_1$ and $\mathcal{R}_2$
4:    for $i = 1, 2$ do
5:       reduce the domain of $\mathcal{R}_i$ using $H$ and $\overline{G}$
6:       suppose $\mathcal{R}_i = [a, b]$
7:       if $f(a) = +\infty$ or $b \notin H$ then
8:          in case of $f(a) = +\infty$, add a feasibility cut to refine $\overline{G}$
9:          fathom $\mathcal{R}_i$ (it contains no feasible solutions)
10:      else $\{f(a) < +\infty$ and $b \in H\}$
11:         set $LB(\mathcal{R}_i) = f(a)$
12:         if $a \in H$ then
13:            $a$ is a feasible solution, add an optimality cut to refine $\overline{G}$
14:            if $f(a) < UB$ then
15:               $UB \leftarrow f(a)$ and $y^* \leftarrow a$
16:            end if
17:            fathom $\mathcal{R}_i$ ($a$ is the best feasible solution in $\mathcal{R}_i$)
18:         else $\{a \notin H\}$
19:            set $\mathcal{L} \leftarrow \mathcal{L} \cup \{\mathcal{R}_i\}$
20:            search for feasible solutions: select a finite set $\mathcal{S} \subset \mathcal{R}_i$
21:            for each $y \in \mathcal{S}$ do
22:               if $f(y) = +\infty$ then
23:                  add a feasibility cut to refine $\overline{G}$
24:               else if $f(y) < +\infty$ and $y \in H$ then
25:                  $y$ is a feasible solution, add an optimality cut to refine $\overline{G}$
26:                  if $f(y) < UB$ then
27:                     $UB \leftarrow f(y)$ and $y^* \leftarrow y$
28:                  end if
29:               end if
30:            end for
31:         end if
32:      end if
33:   end for
34:   for each $\mathcal{R} \in \mathcal{L}$ do
35:      if $LB(\mathcal{R}) > UB$ or $v(\mathcal{R}) \leq \epsilon$ then
36:         fathom $\mathcal{R}$, i.e, set $\mathcal{L} \leftarrow \mathcal{L} \setminus \{\mathcal{R}\}$
37:      end if
38:   end for
39: end while
```

Figure 15: A Branch-Reduce-Cut algorithm for PCLP

have an oracle to check, given $y \in \mathbb{R}^m$, whether $y \in H$, i.e., we can evaluate the cumulative density function F of the random vector $\xi(\omega)$ and check if $F(y) \geq \alpha$. An algorithmic description of the proposed branch-cut-reduce scheme is presented in Figure 15. The key steps are highlighted in italics, and are described below.

The algorithm starts by considering the initial parition $[y^L, y^U]$ given by the bounds on the variables and checks the necessary feasibility condition in Proposition 4.4(i) and the sufficient optimality condition in Proposition 4.4(ii). If neither of these two conditions are satisfied, the algorithm starts its main loop.

4.3.1 Selection and branching

The first main step is to select a partition from the list $\mathcal{L}$ of unfathomed partitions. This choice is based upon the least-lower bound rule (line 2 of the main loop in Figure 15). This guarantees that the bounding process is *bound improving* [9].

After the selection step, the selected hyper-rectangle is partitioned into two hyper-rectangles. In the case of a continuous distribution, the branching rule is to bisect the longest edge. That is, given $\mathcal{R} = [a, b]$ of size $v(\mathcal{R}) > 0$ with the longest edge index $\widehat{\jmath} \in \operatorname{argmax}_{j=1,\ldots,m}\{b_j - a_j\}$, $\mathcal{R}$ is partitioned into $\mathcal{R}_1 = [a, b - e^{\widehat{\jmath}} v(\mathcal{R})/2]$ and $\mathcal{R}_2 = [a + e^{\widehat{\jmath}} v(\mathcal{R})/2, b]$, where $e^j \in \mathbb{R}^m$ is the j-th unit vector. The longest-edge bisection rule guarantees that the branching process is *exhaustive* [9].

Since, in the discrete case we can restrict our search to the set $\mathcal{C}$ (Proposition 4.5), our branching rule is as follows. Given a hyper-rectangle $\mathcal{R} = [a, b]$ of size $v(\mathcal{R}) > 0$ suppose the longest edge has an index $\widehat{\jmath} \in \operatorname{argmax}_{j=1,\ldots,m}\{b_j - a_j\}$. Let $\Xi_{\widehat{\jmath}} = \{\sigma_1, \sigma_2, \ldots, \sigma_K\}$ with $\sigma_k < \sigma_{k+1}$ for all k. Choose $\{\sigma_p, \sigma_q\} \in \Xi_{\widehat{\jmath}}$ such that $\sigma_p < (a_{\widehat{\jmath}} + b_{\widehat{\jmath}})/2 \leq \sigma_q$, and let $b_1 = b - (b_{\widehat{\jmath}} + \sigma_p)e^{\widehat{\jmath}}$ and $a_2 = a + (a_{\widehat{\jmath}} + \sigma_q)e^{\widehat{\jmath}}$. Then $\mathcal{R}$ is partitioned into $\mathcal{R}_1 = [a, b_1]$ and $\mathcal{R}_2 = [a_2, b]$. Note that, if $\mathcal{R} = [a, b]$ is such that

$$a, b \in \mathcal{C} \tag{4.3.1}$$

then σ_p and σ_q always exist and the above branching rule is well-defined. Moreover, the resulting partitions $\mathcal{R}_1$ and $\mathcal{R}_2$ also satisfy condition (4.3.1). We shall show that the domain reduction step on any partition preserves condition (4.3.1). Thus, if the initial partition

$[y^L, y^U]$ satisfies condition (4.3.1) (this can be ensured by domain reduction), then all subsequent partitions produced by the algorithm will satisfy (4.3.1).

4.3.2 Domain reduction

Suppose we have a current normal set approximation $\overline{G}$ of the set of optimal solutions, then the region of interest is $\overline{G} \cap H$. Given a partition $\mathcal{R} = [a, b]$ such that $a \in \overline{G}$ and $b \in H$, the idea of domain reduction is to find a hyper-rectangle $\mathcal{R}' \subset \mathcal{R}$ such that $\mathcal{R}' \supseteq \mathcal{R} \cap \overline{G} \cap H$. This reduction helps to obtain improved bounds and reduces the size of the search tree.

The domain reduction on $\mathcal{R} = [a, b]$, with $a \in \overline{G}$ and $b \in H$, can be carried out as follows. Let

$$\lambda_j^* = \max\{\lambda \in [0,1] : \ a + \lambda(b_j - a_j)e^j \in \overline{G}\} \qquad \text{for } j = 1, \dots, m \tag{4.3.2}$$

$$\mu_j^* = \max\{\mu \in [0,1] : \ b - \mu(b_j - a_j)e^j \in H\} \qquad \text{for } j = 1, \dots, m. \tag{4.3.3}$$

The above problems both have optimal solutions, since the sets $\overline{G}$ and H are closed. Construct

$$a' = b - \sum_{j=1}^{m} \mu_j^*(b_j - a_j)e^j \tag{4.3.4}$$

$$b' = a + \sum_{j=1}^{m} \lambda_j^*(b_j - a_j)e^j. \tag{4.3.5}$$

Proposition 4.6. *$[a', b'] \supseteq [a, b] \cap \overline{G} \cap H$.*

Proof. Consider $y \in [a, b] \cap \overline{G} \cap H$, such that $y \notin [a', b']$. Then there exists an index $\widehat{\jmath} \in \{1, \dots, m\}$, such that either $a_{\widehat{\jmath}} \le y_{\widehat{\jmath}} < a'_{\widehat{\jmath}}$ or $b'_{\widehat{\jmath}} < y_{\widehat{\jmath}} \le b_{\widehat{\jmath}}$. Consider the first case (the second case is analogous). Let $z = y + \sum_{j=1, j \neq \widehat{\jmath}}^{m} (b_j - y_j)e^j$, i.e., $z_j = b_j$ for all $j = 1, \dots, m, j \neq \widehat{\jmath}$ and $z_{\widehat{\jmath}} = y_{\widehat{\jmath}}$. Let $\widehat{\mu}_{\widehat{\jmath}} = (b_{\widehat{\jmath}} - z_{\widehat{\jmath}})/(b_{\widehat{\jmath}} - a_{\widehat{\jmath}})$. Clearly $\widehat{\mu} \in [0, 1]$, and we can write $z = b - \widehat{\mu}_{\widehat{\jmath}}(b_{\widehat{\jmath}} - a_{\widehat{\jmath}})e^{\widehat{\jmath}}$. Note that, by construction, $z \ge y$, and since $y \in H$, we have that $z \in H$ (since H is a reverse normal set). Thus $\widehat{\mu}_{\widehat{\jmath}}$ is a feasible solution to the problem (4.3.3) corresponding to $\widehat{\jmath}$. Then

$$b_{\widehat{\jmath}} - \widehat{\mu}_{\widehat{\jmath}}(b_{\widehat{\jmath}} - a_{\widehat{\jmath}}) = z_{\widehat{\jmath}} = y_{\widehat{\jmath}} < a'_{\widehat{\jmath}} = b_{\widehat{\jmath}} - \mu_{\widehat{\jmath}}^*(b_{\widehat{\jmath}} - a_{\widehat{\jmath}}).$$

The above implies that $\widehat{\mu}_{\widehat{\jmath}} > \mu_{\widehat{\jmath}}^*$ and we have a contradiction to the fact that $\mu_{\widehat{\jmath}}^*$ is an optimal solution to the problem (4.3.3) corresponding to $\widehat{\jmath}$. □

Proposition 4.6 establishes that the domain reduction process is valid, i.e., no feasible solutions are lost. In case of continuous distributions, the one-dimensional optimization problems (4.3.2) and (4.3.3) can be solved by bisection. For discrete distributions, the problems simplify to

$$\lambda_j^* = \max\{\lambda \in [0,1] : \ a + \lambda(b_j - a_j)e^j \in \overline{G} \cap \mathcal{C}\} \qquad \text{for } j = 1, \dots, m$$

$$\mu_j^* = \max\{\mu \in [0,1] : \ b - \mu(b_j - a_j)e^j \in H \cap \mathcal{C}\} \qquad \text{for } j = 1, \dots, m,$$

and can be solved by sorting the elements in Ξ_j for all $j = 1, \dots, m$. It is immediately seen that, in this case, the resulting partition $[a', b']$ will satisfy condition (4.3.1).

4.3.3 Feasibility and optimality cuts

Whenever a solution $\widehat{y}$ is found such that $f(\widehat{y}) = +\infty$, i.e., the linear program (4.2.3) is infeasible, a feasibility cut is added to the description of the set $\overline{G}$ (see lines 8 and 22 of the main loop in Figure 15). The cut is generated as follows. Recall that dual polyhedron for the linear program (4.2.3) is

$$\{(\pi, \rho) \in \mathbb{R}^{p+m} : \ \pi A + \rho T \leq c, \ \rho \geq 0\}.$$

If $f(\widehat{y}) = +\infty$, then the dual to (4.2.3) is unbounded (since dual feasibility is assumed), i.e., there exists an extreme ray $(\widehat{\pi}, \widehat{\rho})$ of the above dual polyhedron such that

$$\widehat{\pi} b + \widehat{\rho}\widehat{y} > 0.$$

Thus any $y \in G$ should satisfy the *feasibility cut*

$$\widehat{\rho} y \leq -\widehat{\pi} b. \tag{4.3.6}$$

Whenever a solution $\widehat{y} \in H$ such that $f(\widehat{y}) < +\infty$, i.e., $\widehat{y}$ is feasible, an optimality cut is added to the description of $\overline{G}$ (see lines 12 and 24 of the main loop in Figure 15). The optimality cut is generated as follows. Let $(\widehat{\pi}, \widehat{\rho})$ be an optimal dual solution for the linear program (4.2.3) defining $f(\widehat{y})$. Then any y such that $f(y) \leq f(\widehat{y})$ should satisfy the *optimality cut*

$$\widehat{\rho} y \leq \widehat{\rho}\widehat{y}. \tag{4.3.7}$$

To see this, first note that $(\widehat{\pi}, \widehat{\rho})$ is a feasible dual solution for the linear program (4.2.3) defining $f(y)$, thus $f(y) \geq \widehat{\pi}b + \widehat{\rho}y$. Since $f(\widehat{y}) = \widehat{\pi}b + \widehat{\rho}\widehat{y}$, the requirement $f(y) \leq f(\widehat{y})$ then implies the optimality cut (4.3.7).

At any point in the algorithm, the set $\overline{G}$ is defined by a set of feasibility and optimality cuts of the forms (4.3.6) and (4.3.7), respectively. Note that the cut-coefficients are always non-negative. Thus the set $\overline{G}$ is always maintained to be a normal set.

Example 2 (contd.): Here, we illustrate the cutting plane and domain reduction steps using Example 2. Figure 16(a) shows a feasibility cut. Consider the initial partition defined by $[(-5, 1.5), (1, 5.5)]$ *(shown as the dashed rectangle). While searching for feasible solutions (see the subsection 4.3.4), the algorithm considers the* $(1, 5.5)$ *and its two adjacent vertices* $(-5, 1.5)$ *and* $(1, 1.5)$*. All of these points are infeasible. The sloped dotted line in Figure 16(a) is the feasibility cut corresponding to the point* $(1, 1.5)$*. Upon adding this cut, the domain reduction step is able to reduce the partition* $[(-5, 1.5), (1, 5.5)]$ *to* $[(-5, 2), (-2, 5)]$ *(shown as the solid rectangle).*

Figure 16(b) shows two optimality cuts. Consider now the partition $[(-5, 2), (2, 5)]$*. The vertices* $(-2, 2)$ *and* $(-5, 5)$ *both have finite optimal objective values, and consequently the two optimality cuts shown as sloped dotted lines in Figure 16(b) are generated. The optimality cut corresponding to* $(-5, 5)$ *helps to reduce the partition* $[(-5, 2), (-2, 5)]$ *to* $[(-5, 2), (-3, 5)]$.

4.3.4 Upper bounding and searching for feasible solutions

Whenever a feasible solution is found, the corresponding objective value is checked against the upper bound UB, i.e., the objective value of the best solution y^* found thus far, and UB and y^* are updated appropriately. Also, as mentioned earlier, an optimality cut is added to the description of $\overline{G}$.

For a given partition $\mathcal{R}$, the algorithm first checks if a is feasible, i.e., $f(a) < +\infty$ and $a \in H$. If it is, then this partition is fathomed, since it cannot contain any better feasible solution.

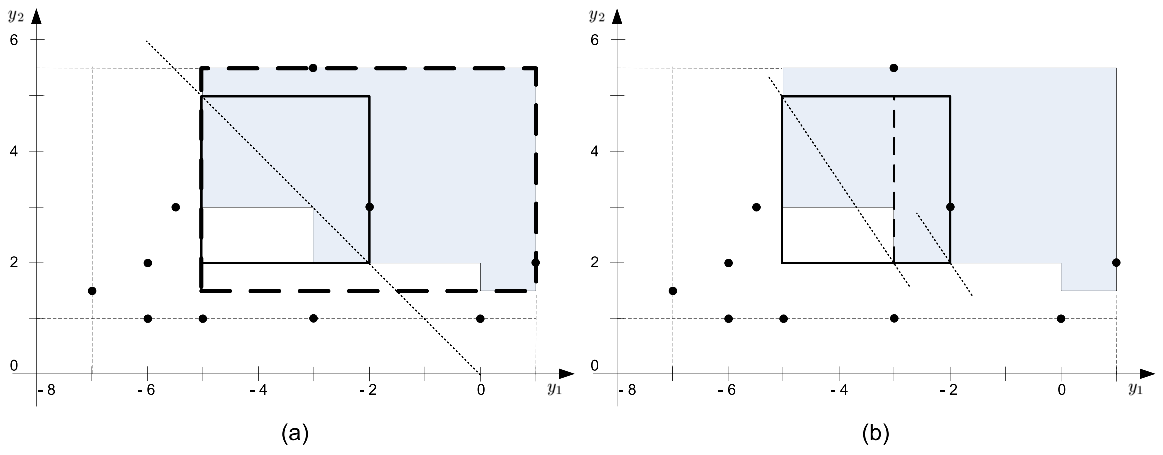


Figure 16: (a) Feasibility cut corresponding to $y = (1, 1.5)^T$ and (b) Optimality cuts corresponding to $y = (-5, 5)^T$ and $y = (-2, 2)^T$

To search for feasible solutions within an unfathomed partition $\mathcal{R} = [a, b]$, the algorithm (see lines 20-30 of the main loop in Figure 15) checks b and the set of m vertices of $\mathcal{R}$ adjacent to b (this is the candidate set $\mathcal{S}$). The adjacent vertices are checked since these might have smaller objective value than b. We might also search for solution on the relative boundary of H on the line segment joining a to b, however, this might be computationally expensive.

Suppose we encounter a solution $\widehat{y}$ such that $f(\widehat{y}) < +\infty$ but $\widehat{y} \notin H$. We may then attempt to "lift" this solution to be feasible as follows. Let $\widehat{x}$ be an optimal solution to the linear program (4.2.3) defining $f(\widehat{y})$. Then, $T\widehat{x} \geq \widehat{y}$. If some of these inequalities are strict, we can consider the solution $y' = \min\{T\widehat{x}, b\}$ (the min is taken component-wise). Clearly, $f(y') = f(\widehat{y})$, and since $y' \geq \widehat{y}$, it might be that $y' \in H$, in which case, y' is a feasible solution.

4.3.5 Fathoming

A given partition $\mathcal{R} = [a, b]$ can be fathomed, or discarded from further consideration, if one of the following conditions are satisfied.

Infeasibility: If the partition fails to satisfy the necessary feasibility condition $a \in G$ and $b \in H$, then it can fathomed since contains no feasible solutions.

Feasibility: If a or a solution obtained by "lifting" a is feasible, then the partition can be fathomed since it contains no better solutions.

Inferiority: If $LB(\mathcal{R}) > UB$, the partition does not contain any solution whose objective function value is smaller than that of the best solution already found. Hence the partition can be fathomed.

Tolerance: If the $v(\mathcal{R}) \leq \epsilon$ and has not been fathomed according to the rules above, then the partition is not considered further. In this case, we have $f(a) < +\infty$, $a \notin H$ and $b \in H$. In this case, we choose a to be an "approximately feasible" solution and compare it with the best candidate solution found so far. Note that, if $\epsilon = 0$ (as in the case of discrete distributions) this fathoming rule is never encountered.

4.4 Convergence analysis

4.4.1 Discrete distribution

We shall show that, in case of a discrete distribution of the random vector ξ, the proposed branch-reduce-cut algorithm (with the tolerance $\epsilon = 0$), either finds a global optimal solution to PCLP or resolves that the problem is infeasible. We shall need the following concept.

Definition 4.1. [9] *The bounding operation in a branch-and-bound algorithm is called* finitely consistent *if, (i) at every step, any unfathomed partition element can be further refined, and if (ii) any nested sequence of successively refined partition elements is finite.*

Lemma 4.2. *The bounding operation in the Branch-reduce-cut algorithm is finitely consistent.*

Proof. Recall that the algorithm always generates partitions satisfying condition (4.3.1). Consider, any unfathomed partition $[a, b]$ satisfying (4.3.1). As mentioned earlier, since $\{a, b\} \in \mathcal{C}$, the branching rule is well-defined, and so $[a, b]$ can be further refined. Also, the branching rule guarantees that any nested sequence of unfathomed partitions will reduce to a point after a finite number of steps, whence such a partition will have to be fathomed. Thus any nested sequence of successively refined partition elements is finite. □

Theorem 4.1. *Given a PCLP with discrete distributions for the random parameters, the Branch-reduce-cut algorithm with $\epsilon = 0$ terminates after finitely many steps either with a global optimal solution or by resolving that the problem is infeasible.*

Proof. From Theorem IV.1 in [9], a branch-and-bound algorithm where the bounding operation is finitely consistent terminates in a finite number of steps. Thus by Lemma 4.2, the proposed Branch-reduce-cut algorithm terminates after finitely many steps.

Consider, first, the case that the algorithm found a feasible solution y^*. In this case, the feasible region is non-empty, therefore PCLP has an optimal solution. Suppose $y' \in \mathcal{C}$ is an optimal solution to PCLP. We shall show that $f(y^*) = f(y')$, so that y^* is an optimal solution of PCLP. Suppose, for contradiction, that $f(y') < f(y^*)$. Let $\mathcal{R}_i = [a^i, b^i]$ for $i = 1, \ldots, I$ be the set of partitions corresponding to the leaf nodes of the branch-and-bound search tree upon termination. By the exhaustiveness of the branching rule and the validity of the domain reduction rules, there exists i' such that $y' \in \mathcal{R}_{i'} = [a^{i'}, b^{i'}]$. Then $a^{i'} \in G$ and $b^{i'} \in H$, therefore $\mathcal{R}_{i'}$ cannot be fathomed due to infeasibility. Moreover, by $f(a^{i'}) \leq f(y') \leq f(b^{i'})$, we must have $a^{i'} \notin H$, since otherwise $a^{i'}$ would be an optimal solution, and the algorithm would have discovered it and set $y^* = a^{i'}$. So $\mathcal{R}_{i'}$ cannot be fathomed by feasibility. Finally, since $f(a^{i'}) \leq f(y') < f(y^*) = UB$, therefore $\mathcal{R}_{i'}$ cannot be fathomed by inferiority. Thus, $\mathcal{R}_{i'}$ can be further refined, and the algorithm should not have terminated. Hence $f(y') = f(y^*)$.

Consider now the second case, that the algorithm did not find any feasible solution. We then claim that PCLP is infeasible. Suppose, for contradiction, that y' is a feasible solution to PCLP. As before, let $\mathcal{R}_i = [a^i, b^i]$ for $i = 1, \ldots, I$ be the set of partitions corresponding to the leaf nodes of the branch-and-bound search tree upon termination, and let i' be such that $y' \in \mathcal{R}_{i'} = [a^{i'}, b^{i'}]$. Then $a^{i'} \in G$ and $b^{i'} \in H$, therefore $\mathcal{R}_{i'}$ cannot be fathomed due to infeasibility. Moreover, $a^{i'} \notin H$, since otherwise $a^{i'}$ would be a feasible solution, and the algorithm would have discovered it. So $\mathcal{R}_{i'}$ cannot be fathomed by feasibility. Finally, since no feasible solution has been found, $UB = +\infty$, and $f(a^{i'}) < UB$, therefore $\mathcal{R}_{i'}$ cannot be fathomed by inferiority. Thus, $\mathcal{R}_{i'}$ can be further refined, and the algorithm should not have terminated. Hence, PCLP cannot have any feasible solution. □

4.4.2 Continuous distribution

In the continuous distribution case, if $\epsilon = 0$, then the algorithm may not terminate finitely and can only be guaranteed to converge in the limit. To see this, suppose that the feasible set is a singleton, then an infinite number of branching operations may be needed to attain a partition $[a, b]$ where the single feasible point coincides with a.

Theorem 4.2. *Given a PCLP with continuous distributions for the random parameters, if the Branch-reduce-cut algorithm with $\epsilon = 0$ terminates, then it terminates with a global optimal solution or by resolving that the problem is infeasible.*

Proof. Analogous to the proof of Theorem 4.1. □

Consider now the case when the algorithm does not terminate. Let k denote the index for the iterations, UB^k denote the upper bound at iteration k, and $[a^k, b^k]$ be the partition considered in iteration k.

Theorem 4.3. *Given a PCLP with continuous distributions for the random parameters, if the Branch-reduce-cut algorithm with $\epsilon = 0$ does not terminate, then PCLP has an optimal solution y^*, and*

$$\lim_{q\to\infty} a^{k_q} = y^*.$$

Proof. Note that $a^k \in G$ and $b^k \in H$ for all k. Also by the exhaustiveness of the branching rule, we have $\lim_{k\to\infty}(b^k - a^k) = 0$. Since the sets G and H are closed, for any convergent subsequence $\lim_{q\to\infty} b^{k_q} = \lim_{q\to\infty} a^{k_q} = a^* \in G \cap H \cap [y^L, y^U]$. Thus PCLP has a feasible solution a^*, and hence an optimal solution y^*.

The least lower bound selection rule guarantees that $f(a^{k_q}) \leq f(y^*)$ for all q. Thus $\lim_{q\to\infty} f(a^{k_q}) = f(a^*) \leq f(y^*)$, where the first equality follows from the continuity of f over G. Therefore, a^* is also an optimal solution. □

To ensure finite termination, a positive tolerance ($\epsilon > 0$) is required. The tolerance-based fathoming rule then guarantees that the algorithm terminates. In this case, we might end up with a δ-feasible solution y^*, i.e., $\alpha - F(y^*) \leq \delta$ for some $\delta > 0$. For a PCLP with an absolutely continuous cumulative density $F : \mathbb{R}^m \mapsto [0,1]$ for the random vector $\xi(\omega)$ with $f_i : \mathbb{R}^m \mapsto \mathbb{R}_+$ for $i = 1, \ldots, m$ as the corresponding marginal probability density functions, ϵ and δ are related as $\delta \leq \epsilon L$, where $L = \max_{i=1,\ldots,m} \max_{y \in [y^L, y^U]} \{f_i(y)\}$.

4.5 Computational results

In this section we report on some computational experience in using the proposed Branch-cut-reduce algorithm for randomly generated instances of PCLP with discrete distributions, i.e., $\xi(\omega)$ has K possible realizations $\{\xi^1, \ldots, \xi^K\}$ with probabilities $\{p^1, \ldots, p^K\}$. Such a problem can be immediately reformulated into the following milp (see, e.g., [20])

$$\begin{aligned}
\min_{x,y,\lambda} \quad & c^T x \\
\text{s.t.} \quad & Ax = b \\
& Tx \geq y \\
& \sum_{k=1}^{K} p^k \lambda_k \geq \alpha \\
& \xi_j^k \lambda_i \leq y_j \quad \text{for } j = 1, \ldots, m,\ k = 1, \ldots, K \\
& \lambda_k \in \{0,1\} \quad \text{for } k = 1, \ldots, K \\
& x \geq 0, y \geq 0.
\end{aligned}$$

The above milp formulation can be improved by adding the constraints

$$\lambda_{k_1} \geq \lambda_{k_2} \quad \text{if } \xi^{k_1} \leq \xi^{k_2} \text{ for } k_1 = 1, \ldots, K \text{ and } k_2 = k_1 + 1, \ldots, K.$$

We compare the performance of the proposed algorithm with that of solving the above milp formulation using CPLEX 8.0.

The proposed algorithm was implemented in C++ with CPLEX 8.0 as the linear programming solver. All computations were on a UltraSparc-III-Cu UNIX workstation 2x900MHz CPUs and 2GB RAM.

For generating our test problems, we replaced the constraint $Ax = b$ by simple bounds. The number n of x-variables was fixed at $n = 50$, and the number m of y-variables was varied in the set $\{3, 6, 9\}$. The number of scenarios K was varied in the set $\{100, 300, 500\}$.

The desired probability level α was set to 0.9. For each combination of m and K, five test problems were randomly generated as follows:

1. K realizations of the ξ vector were randomly sampled from a uniform distribution over $[0, 100]^m$. Each realization was assigned a probability of $1/K$.

2. Each component of c and T were independently randomly sampled from a uniform distribution over $[0, 20]$.

Table 5 compares, for each test problem, the CPU seconds required to solve the milp reformulation using CPLEX 8.0 (under columns labelled "MIP"), the CPU seconds required by a branch-and-reduce algorithm, i.e., the proposed scheme without the enhancements offered by cutting planes (under the columns labelled "BR"), and the CPU seconds required by the proposed branch-reduce-cut algorithm (under the columns labelled "BRC"). In the majority of the test problems, the proposed branch-cut-reduce algorithm performed better that the other two approaches. In particular, in only two out of 45 test problems, the milp approach showed better performance.

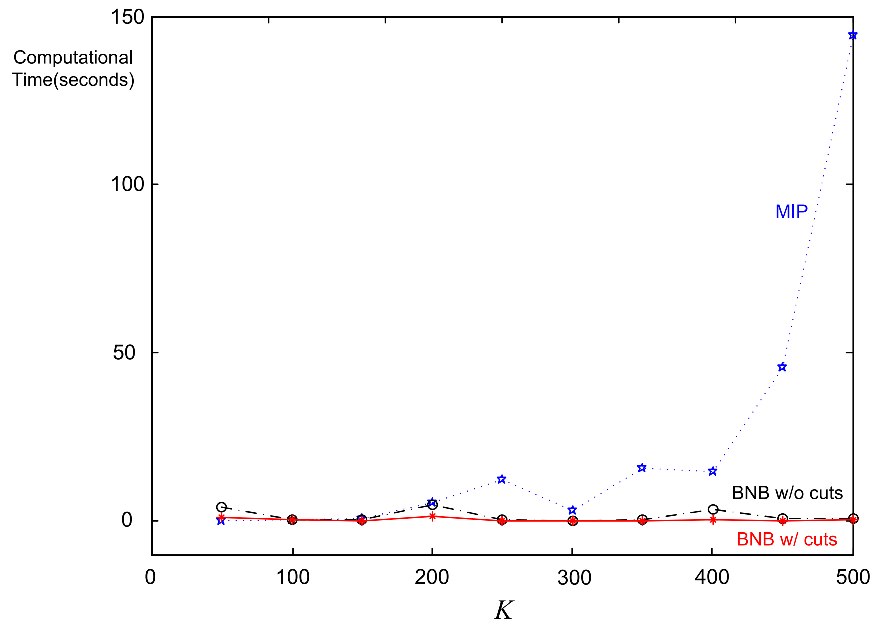


Figure 17: CPU seconds versus K ($m = 5$, $n = 50$).

In Figure 17, we compare the growth of the CPU time required by the milp approach and

Table 5: Comparison of CPU seconds

	$K = 100$			$K = 300$			$K = 500$		
Test No.	MIP	BR	BRC	MIP	BR	BRC	MIP	BR	BRC
1	0.04	0.00	0.00	2.78	0.00	0.00	240.84	0.19	0.04
2	0.04	9.00	1.59	7.67	0.28	0.08	16.26	0.00	0.00
3	0.03	0.00	0.00	42.77	0.25	0.00	175.48	0.59	0.12
4	0.09	0.06	0.00	2.81	0.00	0.00	16.36	0.00	0.00
5	0.04	0.00	0.00	5.35	0.00	0.00	18.71	0.17	0.05
AVG	0.05	2.27	0.40	14.65	0.13	0.02	56.70	0.19	0.04

$m = 3$

	$K = 100$			$K = 300$			$K = 500$		
Test No.	MIP	BR	BRC	MIP	BR	BRC	MIP	BR	BRC
1	0.03	0.00	0.00	10.46	0.32	0.00	7.91	0.00	0.00
2	0.14	0.15	0.00	23.62	0.77	0.13	8.20	0.00	0.00
3	0.05	0.17	0.00	51.43	3.16	0.39	59.11	6.50	4.01
4	0.05	0.37	0.00	2.12	0.62	0.22	4.94	0.00	0.00
5	0.24	0.09	0.00	1.70	0.00	0.00	202.57	3.47	1.51
AVG	0.10	0.16	0.00	17.87	0.97	0.15	56.55	1.99	1.10

$m = 6$

	$K = 100$			$K = 300$			$K = 500$		
Test No.	MIP	BR	BRC	MIP	BR	BRC	MIP	BR	BRC
1	2.21	1.04	0.00	1.91	0.00	0.00	445.89	9.61	5.09
2	0.65	0.46	0.00	82.90	3.18	0.63	307.91	3.94	0.16
3	5.34	29.13	8.61	3.94	0.32	0.00	2261.06	37.55	13.64
4	5.39	0.72	0.08	5.00	0.53	0.00	89.97	1.45	0.38
5	3.25	3.05	0.62	2.73	1.38	0.64	5216.37	21.54	16.75
AVG	3.37	6.88	1.86	19.30	1.08	0.25	1664.24	14.82	7.20

$m = 9$

that required by the proposed algorithm, with and without the cutting plane enhancements, as the number K of realizations in the distribution of the uncertain parameters grows for PCLPs with $m = 5$ and $n = 50$. Each data point in the graph corresponds to an average over 5 randomly generated instances. It is evident that the proposed algorithm offers significant advantages over the milp approach as the the number of realizations increase.

CHAPTER V

CONCLUSION

This thesis addresses the development and application of global optimization algorithms for monotonic optimization problems. In this chapter, we give a brief summary of the various approaches discussed and their computational performance, and give directions for future research.

In Chapter 2, we discussed the basic properties of monotonic programming and reviewed an established outer approximation method called the Polyblock Algorithm. Based on our computational experiments with this procedure, we observed that a considerable portion of computation time was spent because of the inefficiency of procedures for removing improper vertices. Moreover, the method is susceptible to jamming on some problems. We proposed an efficient way of removing improper vertices by identifying a smaller set of vertices that needed to be compared. In the other enhancement, we looked at the jamming phenomenon which occurs when the chord connecting a vertex and a point b approaches 90° (Recall that a point b is the coordinate-wise biggest point of a bounded problem). Because of this steepness, we could generate a new vertex that is close to a refined vertex (see, Table 1). Rather than using a static point b, we proposed a different method to find a boundary point by constructing a dynamic point b'. The efficiency of this modification was demonstrated by our computational results. The convergence of the modified algorithm was proved. We reviewed two types of branch-and-bound algorithms — one is based on a rectangular branching scheme and the other is based on a simplicial branching scheme. For the rectangular branch-and-bound algorithm, we constructed a domain reduction scheme that is enhanced by adding optimality cuts. By exploiting the properties of the feasible set defined by monotone functions, we categorized various fathoming cases — infeasibility, feasibility and inferiority. For the simplicial branch-and-bound algorithm, we discussed the construction of upper and lower bounds for a given partition.

In Chapter 3, we considered polynomial programming problems in order to compare the performance of various algorithms. We showed that a general polynomial programming problem can be transformed into a class of monotonic programming problems. In an algorithm based on a branch-and-bound scheme, a tighter bound for each partition plays a key role in reducing computational effort. We provided two ways to obtain a tighter bound — one is obtained from solving a linear relaxation problem and the other is obtained from solving a subproblem after fixing some of the variables. We tested the performance of variants of algorithms on instances of separable polynomial programming problems. Both bounding schemes help in reducing the computational effort. However, the variant that employs a linear relaxation scheme outperformed one that uses bounding by variable fixing. According to the computational experiments, the branch-and-bound algorithm shows a better performance than either the Polyblock Algorithm or the modified Polyblock Algorithm. Interestingly, the rectangular branching scheme outperformed the conical branching scheme on our test problems.

In Chapter 4, we considered probabilistically constrained linear programs with general distributions for the uncertain parameters, and we proposed an efficient branch-and-bound algorithm. This basic branch-and-bound algorithm was enhanced by domain reduction and cutting plane strategies. The domain reduction scheme is inherent from the branch-and-bound algorithm for a general monotonic programming problem. Two types of cuts, Optimality Cuts and Feasibility Cuts, are constructed by using the weak duality theorem and the existence of extreme rays for infeasible cases. A general approach for a discrete random parameter case is to use mixed integer programming (MIP). In a MIP formulation, we need one binary variable for each realization. That is, as the number of realizations is increased, the number of binary variables is increased. Therefore, it is not easy to solve the MIP formulation for a large number of realizations. This was confirmed by our computational results. We provided convergence proofs for the algorithm.

This research can be extended to two variants of probabilistically constrained linear programs. First, we considered problems having random parameters only on the right-hand side of a linear program. We may extend this research to problems having random parameters on the left-hand side of a linear program. If the supports of the random parameters are positive and the coefficients of the random parameters are positive, then the extended problem is also a monotonic program. The other possible variant of the problem is a problem having integer decision variables. In this case, we need to solve mixed integer programs in order to evaluate bounds for a given box, which are given by optimal objective values of the MIPs. This procedure is clearly expensive. Moreover, we do not have dual multipliers, so we cannot construct feasibility and optimality cuts. Alternatively, we can branch on the discrete decision variables and solve continuous subproblems at each node which are of the form PCLP. This approach would need to be compared with the general mixed integer programming formulation of the discrete decision variable model. Based on our reported comparison between our method and the MIP formulation of PCLP, we believe that our approach for the discrete decision variables extension will be very competitive.

APPENDIX A

COMPUTATIONAL RESULTS

A.1 Comparison of CPU time

In this appendix, we report computational results of various experiments comparing algorithms on test problems with various conditions. In the following description, n denotes the number of variables and ϵ denotes the termination tolerance.

A.1.1 Polyblock, modified Polyblock, and Branch-and-bound

In this section, we report computational results from applying the Polyblock Algorithm (Algorithm 1), the modified Polyblock Algorithm (Algorithm 2) and the branch-and-bound algorithm (Algorithm 3) to randomly generated separable polynomial programming problems with various conditions. Various experimental conditions were determined by the combination of a number of variables in the set $\{3, 4, 5\}$, a number of constraints in the set $\{5, 7\}$[1] and a termination tolerance in the set $\{0.01, 0.001, 0.0001\}$.

Each randomly generated problem was solved by three algorithms — Polyblock Algorithm (labeled 'PA'), modified Polyblock Algorithm (labeled 'mPA') and branch-and-bound algorithm (labeled 'BNB') — and the computational times (required CPU time) and an ϵ-optimal objective value are reported under columns labelled "CTs" and "Opt". The test results are shown in Table 6 through Table 14.

[1]In each results table, the first five problems have 5 constraints and the last five problems have 7 constraints.

Table 6: (PA vs. mPA vs. BNB) $m = 3$ and $\epsilon = 0.01$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
1	0.03	0.864	0.05	0.861	0.01	0.861	0.01	0.864	0.01	0.861	0.00	0.861
2	0.01	0.693	0.01	0.692	0.01	0.691	0.00	0.693	0.01	0.692	0.00	0.692
3	0.04	0.506	0.03	0.507	0.01	0.504	0.03	0.507	0.00	0.507	0.01	0.507
4	0.04	0.716	0.05	0.715	0.01	0.716	0.01	0.720	0.01	0.720	0.01	0.720
5	0.02	0.276	0.03	0.275	0.01	0.275	0.09	0.276	0.01	0.276	0.01	0.276
6	0.02	1.473	0.02	1.472	0.02	1.468	0.01	1.474	0.00	1.472	0.00	1.473
7	0.05	1.091	0.02	1.092	0.04	1.092	0.00	1.093	0.00	1.094	0.00	1.092
8	0.27	0.568	0.05	0.569	0.01	0.568	0.20	0.569	0.01	0.569	0.01	0.569
9	0.06	0.670	0.05	0.674	0.02	0.670	0.01	0.674	0.01	0.674	0.00	0.675
10	0.28	0.568	0.22	0.568	0.05	0.569	0.10	0.569	0.04	0.569	0.02	0.569

Table 7: (PA vs. mPA vs. BNB) $m = 3$ and $\epsilon = 0.001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
11	0.01	0.8094	0.02	0.8091	0.04	0.8088	0.01	0.8093	0.00	0.8096	0.01	0.8093
12	0.14	0.5069	0.08	0.5068	0.07	0.5073	0.06	0.5074	0.05	0.5074	0.03	0.5074
13	0.30	0.2764	0.10	0.2763	0.02	0.2763	0.38	0.2765	0.10	0.2765	0.06	0.2765
14	0.08	0.5921	0.11	0.5923	0.04	0.5924	0.02	0.5928	0.01	0.5928	0.00	0.5928
15	0.05	0.4289	0.07	0.4287	0.04	0.4288	0.02	0.4292	0.03	0.4292	0.01	0.4292
16	1.81	0.5687	0.10	0.5688	0.05	0.5685	5.12	0.5688	0.17	0.5688	0.08	0.5688
17	0.05	0.9484	0.05	0.9478	0.05	0.9482	0.03	0.9486	0.03	0.9486	0.02	0.9486
18	0.07	0.4737	0.14	0.4735	0.02	0.4737	0.02	0.4740	0.02	0.4741	0.01	0.4739
19	0.02	0.4670	0.03	0.4671	0.04	0.4670	0.01	0.4670	0.02	0.4673	0.01	0.4672
20	0.02	0.5225	0.01	0.5216	0.04	0.5216	0.00	0.5225	0.00	0.5224	0.00	0.5220

Table 8: (PA vs. mPA vs. BNB) $m = 3$ and $\epsilon = 0.0001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
21	0.31	0.27649	0.15	0.27650	0.07	0.27646	0.99	0.27654	0.67	0.27654	0.23	0.27654
22	0.94	0.86858	0.15	0.86855	0.09	0.86851	0.57	0.86858	0.05	0.86855	0.04	0.86857
23	0.18	1.50181	0.16	1.50179	0.12	1.50177	0.17	1.50186	0.09	1.50186	0.05	1.50186
24	0.20	0.46284	0.24	0.46285	0.10	0.46293	0.11	0.46291	0.11	0.46290	0.05	0.46294
25	0.56	0.89142	0.11	0.89142	0.11	0.89142	2.24	0.89151	0.20	0.89151	0.07	0.89150
26	0.17	0.47414	0.23	0.47412	0.08	0.47409	0.08	0.47416	0.10	0.47417	0.06	0.47416
27	0.06	0.46749	0.12	0.46748	0.08	0.46745	0.02	0.46747	0.03	0.46747	0.03	0.46749
28	0.06	1.11729	0.03	1.11737	0.06	1.11727	0.01	1.11733	0.01	1.11732	0.00	1.11735
29	0.23	0.72270	0.22	0.72271	0.12	0.72279	0.62	0.72279	0.59	0.72279	0.22	0.72280
30	0.42	0.77668	0.25	0.77668	0.11	0.77668	0.44	0.77675	0.26	0.77675	0.14	0.77675

Table 9: (PA vs. mPA vs. BNB) $m = 4$ and $\epsilon = 0.01$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
31	0.34	0.497	0.22	0.498	0.05	0.500	0.41	0.501	0.19	0.501	0.03	0.502
32	6.20	2.193	0.70	2.194	0.06	2.195	60.97	2.201	0.27	2.200	0.04	2.200
33	3.35	0.855	0.19	0.857	0.02	0.859	0.85	0.860	0.02	0.859	0.00	0.860
34	0.73	0.645	0.82	0.643	0.09	0.643	1.28	0.646	0.49	0.646	0.05	0.646
35	0.45	1.337	0.11	1.338	0.04	1.335	0.42	1.337	0.02	1.337	0.00	1.339
36	2.68	0.756	0.21	0.759	0.05	0.756	49.70	0.759	0.31	0.760	0.04	0.761
37	0.43	0.631	0.16	0.630	0.06	0.628	0.70	0.632	0.19	0.632	0.04	0.632
38	2.09	0.705	0.23	0.706	0.06	0.702	20.41	0.709	0.25	0.709	0.06	0.709
39	7.50	0.929	1.44	0.930	0.08	0.928	27.61	0.931	0.28	0.931	0.24	0.931
40	0.66	1.016	0.23	1.013	0.03	1.014	7.27	1.020	0.12	1.019	0.01	1.020

Table 10: (PA vs. mPA vs. BNB) $m = 4$ and $\epsilon = 0.001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
41	9.32	0.7046	2.12	0.7045	0.18	0.7045	45.05	0.7049	6.61	0.7049	0.48	0.7049
42	8.70	1.7349	0.76	1.7356	0.10	1.7348	188.97	1.7358	1.05	1.7358	0.10	1.7357
43	1.74	0.6499	0.97	0.6501	0.11	0.6497	23.72	0.6506	8.65	0.6506	0.37	0.6506
44	6.96	0.9301	2.27	0.9301	0.41	0.9299	76.76	0.9304	10.99	0.9304	0.73	0.9304
45	72.84	1.3034	0.69	1.3030	0.08	1.3028	300.01+	1.3026	0.55	1.3036	0.03	1.3033
46	10.17	1.8351	0.31	1.8343	0.07	1.8346	189.50	1.8352	0.08	1.8350	0.05	1.8352
47	8.38	0.8005	5.75	0.8007	0.29	0.8004	10.96	0.8009	1.94	0.8010	0.22	0.8010
48	6.90	0.8724	0.12	0.8722	0.07	0.8719	39.72	0.8724	0.01	0.8721	0.01	0.8722
49	1.74	0.5553	1.07	0.5554	0.17	0.5555	2.09	0.5560	0.83	0.5560	0.09	0.5560
50	2.17	1.2483	1.83	1.2486	0.22	1.2483	14.06	1.2492	7.75	1.2492	0.77	1.2491

Table 11: (PA vs. mPA vs. BNB) $m = 4$ and $\epsilon = 0.0001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
51	6.06	0.67959	1.10	0.67965	0.19	0.67958	300.02+	0.67967	6.32	0.67967	0.73	0.67966
52	0.40	1.22587	0.17	1.22586	0.09	1.22587	0.25	1.22588	0.04	1.22588	0.00	1.22589
53	0.75	1.05537	0.71	1.05537	0.17	1.05538	0.96	1.05545	0.83	1.05543	0.12	1.05543
54	37.14	0.49927	4.46	0.49923	0.37	0.49925	300.01+	0.49917	300.01+	0.49925	35.90	0.49928
55	0.36	1.74496	0.32	1.74496	0.12	1.74495	0.31	1.74496	0.24	1.74497	0.04	1.74498
56	3.53	0.75945	3.66	0.75947	0.27	0.75944	105.24	0.75953	92.33	0.75953	2.66	0.75953
57	4.95	1.30172	1.71	1.30171	0.18	1.30167	83.07	1.30175	19.42	1.30176	1.18	1.30176
58	7.11	0.87331	0.64	0.87328	0.07	0.87329	122.61	0.87333	0.51	0.87333	0.09	0.87333
59	1.32	0.51714	1.21	0.51718	0.20	0.51712	12.69	0.51720	17.26	0.51719	0.91	0.51720
60	30.67	0.79323	0.53	0.79318	0.10	0.79314	300.02+	0.79180	1.32	0.79321	0.14	0.79319

Table 12: (PA vs. mPA vs. BNB) $m = 5$ and $\epsilon = 0.01$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
61	300.01+	1.294	89.70	1.309	0.08	1.306	300.04+	1.287	300.05+	1.254	0.35	1.313
62	133.65	0.460	5.49	0.461	0.15	0.457	300.01+	0.460	300.05+	0.461	0.17	0.461
63	300.31+	0.837	300.05+	0.836	0.22	0.840	300.05+	0.829	300.05+	0.847	0.95	0.847
64	301.18+	1.628	300.02+	1.655	0.08	1.646	300.06+	1.628	300.18+	1.629	0.09	1.653
65	22.34	0.726	9.45	0.725	0.20	0.725	270.12	0.732	8.49	0.731	0.14	0.731
66	300.01+	1.458	13.64	1.464	0.33	1.461	300.01+	1.458	300.02+	1.439	0.83	1.467
67	6.71	0.824	4.52	0.822	0.16	0.822	77.38	0.827	12.65	0.827	0.25	0.827
68	8.04	1.027	3.85	1.025	0.24	1.024	103.21	1.031	15.81	1.030	0.20	1.031
69	300.10+	1.093	300.06+	1.100	0.19	1.121	300.08+	1.093	300.03+	1.100	0.31	1.122
70	64.76	0.980	2.66	0.985	0.23	0.978	300.22+	0.982	300.14+	0.941	0.20	0.984

Table 13: (PA vs. mPA vs. BNB) $m = 5$ and $\epsilon = 0.001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
71	169.94	1.2737	6.89	1.2738	0.22	1.2736	300.13+	1.2742	10.22	1.2743	0.18	1.2742
72	292.16	1.3698	205.08	1.3700	1.69	1.3697	300.06+	1.3598	300.06+	1.3680	300.01+	1.3703
73	172.26	1.1307	122.44	1.1306	0.69	1.1302	300.02+	1.1302	300.11+	1.1307	300.01+	1.1308
74	326.88+	2.3104	14.49	2.3232	0.27	2.3224	300.02+	2.3115	300.02+	2.3232	0.35	2.3231
75	300.01+	1.1210	125.11	1.1235	0.96	1.1229	300.03+	1.1215	300.01+	1.1182	300.01+	1.1235
76	232.10	1.1327	21.87	1.1326	0.39	1.1327	300.04+	1.1294	300.02+	1.1313	27.20	1.1333
77	301.67+	1.0420	300.01+	1.0824	0.08	1.0827	300.07+	1.0203	300.01+	1.0627	0.14	1.0828
78	300.05+	1.1614	300.11+	1.1667	1.10	1.1768	300.06+	1.1692	300.01+	1.1625	300.01+	1.1770
79	300.78+	1.1155	300.03+	1.1196	0.71	1.1219	300.01+	1.1217	300.03+	1.1115	300.01+	1.1221
80	300.19+	1.2117	300.01+	1.1458	0.64	1.2124	324.74+	1.2087	300.08+	1.2097	7.06	1.2129

Table 14: (PA vs. mPA vs. BNB) $m = 5$ and $\epsilon = 0.0001$

	With Convex Relaxation						Without Convex Relaxation					
	PA		mPA		BNB		PA		mPA		BNB	
No.	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt	CTs	Opt
81	303.31+	0.90460	13.60	0.90497	0.34	0.90496	300.14+	0.90349	251.86	0.90504	2.81	0.90502
82	125.67	1.75116	35.48	1.75112	0.99	1.75115	300.15+	1.74487	300.03+	1.75083	300.01+	1.75120
83	308.39+	1.40153	300.02+	1.41121	0.21	1.41130	300.08+	1.40251	300.17+	1.40831	300.01+	1.41136
84	132.74	1.23560	30.17	1.23557	0.39	1.23557	300.01+	1.22005	300.04+	1.23551	6.47	1.23565
85	300.07+	1.49523	300.02+	1.50620	1.73	1.51416	300.05+	1.50812	300.01+	1.49864	300.01+	1.51416
86	13.85	0.81379	12.05	0.81384	0.41	0.81380	300.07+	0.81384	300.04+	0.81385	3.28	0.81387
87	7.64	0.88167	6.53	0.88169	0.30	0.88169	47.37	0.88174	32.58	0.88171	0.56	0.88173
88	15.04	0.85490	12.11	0.85486	0.43	0.85488	256.66	0.85492	177.61	0.85492	1.46	0.85492
89	300.01+	1.24387	300.01+	1.24344	0.37	1.24521	300.01+	1.23422	300.18+	1.24467	0.89	1.24528
90	300.01+	1.23940	27.90	1.23952	0.47	1.23950	300.03+	1.21914	300.03+	1.23806	7.26	1.23955

A.1.2 conical branching and rectangle branching

In this subsection, we report CPU times of two algorithm — conical algorithm and rectangular branch-and-bound algorithm. In the following tables, columns 'CTs', 'ITs', 'MNs', and 'Opt' denote CPU seconds, iterations required, maximum number of vertices required, and optimal solution, respectively. We generated 60 problems under 6 conditions determined by combinations of number of variables $\{3, 5\}$ and termination tolerances $\{0.01, 0.001, 0.0001\}$. We solved each problem by two algorithms — one is the conical algorithm and the other is the rectangular branch-and-bound algorithm. The test results are shown in Table 15 through Table 20.

Table 15: (Conical vs. Rectangular) $m = 3$ and $\epsilon = 0.01$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
1	0.03	183	45	0.79	0.05	100	30	0.79
2	0.05	247	83	0.59	0.10	167	54	0.59
3	0.02	66	25	0.93	0.01	30	13	0.93
4	0.01	33	16	0.60	0.01	22	8	0.60
5	0.03	116	27	1.88	0.03	66	19	1.88
6	0.02	112	30	0.95	0.03	54	14	0.96
7	0.06	364	106	0.78	0.11	207	55	0.78
8	0.01	89	29	1.52	0.02	37	12	1.52
9	0.01	59	15	1.24	0.02	45	9	1.24
10	0.02	153	45	0.75	0.04	104	27	0.75

A.1.3 Subdivisional bounding

In this subsection, we compare CPU times of three algorithms — branch-and-bound algorithm, branch-and-bound with linear relaxation, and branch-and-bound with Polyblock bounding. In the following tables, column 'CTs', 'ITs', 'MNs', and 'Opt' denote CPU seconds, iterations required, maximum number of vertices required, and optimal solution, respectively. We generated 90 problems under 9 conditions determined by combinations of number of variables $\{3, 5, 6\}$ and termination tolerances $\{0.01, 0.001, 0.0001\}$. The test results are shown in Table 21 through Table 29

Table 16: (Conical vs. Rectangular) $m = 3$ and $\epsilon = 0.001$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
11	0.10	719	169	0.791	0.17	381	79	0.791
12	0.02	132	30	0.935	0.03	64	18	0.935
13	0.01	95	25	0.605	0.04	64	15	0.605
14	0.08	475	94	1.879	0.13	260	58	1.879
15	0.07	414	84	0.955	0.10	202	47	0.955
16	0.31	1754	374	0.781	0.46	915	209	0.781
17	0.04	207	36	1.243	0.07	173	30	1.243
18	0.32	1869	523	0.748	0.44	1132	281	0.748
19	0.98	6166	2057	1.753	1.68	4238	1216	1.753
20	0.19	1158	208	1.395	0.19	498	102	1.395

Table 17: (Conical vs. Rectangular) $m = 3$ and $\epsilon = 0.0001$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
21	1.35	5914	1106	0.7812	1.68	2858	466	0.7812
22	0.02	77	12	1.2463	0.02	45	8	1.2463
23	41.45	63411	21087	1.7533	39.39	47248	12917	1.7533
24	0.12	717	118	0.7288	0.19	362	79	0.7288
25	0.18	1139	227	1.2447	0.27	581	132	1.2447
26	0.08	487	110	0.9124	0.14	306	48	0.9124
27	0.01	95	14	1.2857	0.02	54	10	1.2857
28	0.06	365	87	0.6506	0.19	334	95	0.6506
29	0.48	3282	679	2.3944	0.83	1923	383	2.3944
30	20.18	43201	13205	0.8739	19.19	31284	8535	0.8739

Table 18: (Conical vs. Rectangular) $m = 5$ and $\epsilon = 0.01$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
31	2.52	9910	2624	1.05	1.42	1705	500	1.05
32	0.63	2594	538	2.46	0.39	415	106	2.47
33	300.00+	79197	38313	1.45	10.24	10950	2809	1.45
34	300.00+	63852	48641	1.26	25.06	24264	5873	1.26
35	300.00+	78965	32630	2.45	4.05	4154	1118	2.45
36	300.01+	70075	43533	1.50	20.25	24362	5026	1.50
37	11.94	27497	5576	1.40	1.31	1511	339	1.40
38	2.26	8763	2833	2.35	0.89	1050	277	2.35
39	118.72	82069	16771	0.71	3.41	4342	876	0.71
40	3.51	14304	2931	0.50	0.76	1030	276	0.50

Table 19: (Conical vs. Rectangular) $m = 5$ and $\epsilon = 0.001$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
41	5.60	20003	3511	1.537	1.09	1534	393	1.536
42	300.00+	52391	52162	1.149	23.61	22476	5798	1.164
43	2.72	11505	2158	2.111	1.15	1327	331	2.110
44	19.17	37762	7365	0.988	4.20	4323	983	0.988
45	300.00+	70844	44644	1.638	71.05	52031	14246	1.638
46	300.00+	72916	44177	0.335	10.64	12997	3140	0.335
47	300.01+	58284	49639	0.961	138.11	71935	18054	0.962
48	300.01+	49770	49721	1.049	296.80	123893	25324	1.068
49	0.63	2621	827	0.572	0.36	362	98	0.572
50	16.85	34601	6858	2.072	2.98	3547	780	2.072

Table 20: (Conical vs. Rectangular) $m = 5$ and $\epsilon = 0.0001$

	Conical Branching				Rectangular Branching			
No.	CTs	ITs	MNs	Opt	CTs	ITs	MNs	Opt
51	300.00+	74072	42245	0.4200	39.28	32105	7388	0.4201
52	4.15	17724	2490	1.2266	1.79	2034	430	1.2265
53	300.00+	59972	51817	1.4813	300.00+	73308	43470	1.4824
54	300.00+	79507	40953	1.3685	44.63	38317	7874	1.3687
55	300.01+	68610	45252	0.7871	130.64	70978	16836	0.7872
56	300.00+	64310	49117	2.1565	300.00+	71052	43370	2.1571
57	300.00+	60594	50881	2.4455	300.00+	61221	50052	2.4459
58	300.00+	56871	54847	1.1416	300.01+	55122	51297	1.1424
59	300.00+	60027	51623	1.7572	300.00+	60452	49867	1.7574
60	300.00+	59622	57504	1.1846	300.00+	57214	54006	1.1853

Table 21: (Subdivisional bounding) $m = 4$ and $\epsilon = 0.01$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
1	79	24	0.01	1.12	16	5	0.04	1.12	16	5	0.17	1.12
2	88	27	0.01	0.66	13	3	0.04	0.65	13	3	0.11	0.65
3	61	18	0.01	0.41	15	5	0.04	0.40	15	5	0.13	0.40
4	88	32	0.01	0.80	24	7	0.06	0.80	23	7	0.18	0.80
5	525	133	0.06	1.13	24	7	0.06	1.12	24	7	0.24	1.12
6	164	50	0.02	1.04	14	4	0.04	1.03	14	4	0.12	1.03
7	18	7	0.01	0.90	17	4	0.05	0.90	13	4	0.13	0.89
8	264	71	0.04	0.73	30	6	0.08	0.73	30	6	0.28	0.73
9	27	8	0.01	0.86	15	5	0.05	0.85	15	5	0.14	0.85
10	223	68	0.04	1.27	25	11	0.08	1.26	25	11	0.20	1.26

Table 22: (Subdivisional bounding) $m = 4$ and $\epsilon = 0.001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
11	13380	2742	1.71	0.669	168	67	0.44	0.668	168	67	1.48	0.668
12	874	191	0.11	1.238	32	21	0.09	1.238	22	13	0.30	1.238
13	258	62	0.03	0.586	21	4	0.06	0.585	21	4	0.25	0.585
14	287	49	0.03	0.752	37	5	0.1	0.751	37	5	0.37	0.751
15	3744	841	0.49	0.653	103	33	0.27	0.653	103	33	0.92	0.653
16	44	13	0.01	0.728	21	5	0.06	0.728	21	5	0.23	0.728
17	1006	211	0.17	1.271	53	16	0.16	1.270	53	16	0.61	1.270
18	160	32	0.02	0.522	26	7	0.08	0.522	26	7	0.26	0.522
19	721	165	0.13	0.735	22	6	0.07	0.734	22	6	0.30	0.734
20	1118	255	0.16	0.894	65	18	0.18	0.894	54	15	0.84	0.894

Table 23: (Subdivisional bounding) $m = 4$ and $\epsilon = 0.0001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
21	107	22	0.01	0.4047	39	8	0.1	0.4047	39	8	0.37	0.4047
22	1353	184	0.16	1.1522	54	9	0.14	1.1522	54	9	0.88	1.1522
23	187	27	0.02	0.6483	46	7	0.12	0.6483	46	7	0.51	0.6483
24	1521	313	0.18	1.2374	44	11	0.12	1.2374	43	9	2.24	1.2374
25	13126	2968	1.68	1.2420	66	18	0.18	1.2419	66	18	0.56	1.2419
26	4081	768	0.59	0.8942	97	25	0.27	0.8942	91	25	2.46	0.8942
27	192	35	0.02	1.0164	39	7	0.11	1.0163	39	7	0.44	1.0163
28	587	91	0.09	0.6926	38	10	0.11	0.6926	38	10	0.56	0.6926
29	245696	41491	222.88	1.0398	141	60	0.39	1.0398	141	60	1.67	1.0398
30	262	50	0.04	0.7239	30	6	0.09	0.7238	30	6	0.39	0.7238

Table 24: (Subdivisional bounding) $m = 5$ and $\epsilon = 0.01$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
31	1175	332	0.2	1.30	77	23	0.21	1.30	77	23	0.85	1.30
32	447	157	0.08	1.32	52	16	0.15	1.32	52	16	0.57	1.32
33	3831	1217	0.61	1.26	88	42	0.25	1.26	81	42	1.03	1.26
34	1891	516	0.3	1.80	136	48	0.4	1.80	105	41	2.02	1.80
35	5127	1224	0.84	1.66	84	36	0.23	1.65	84	36	0.73	1.65
36	561	178	0.14	1.10	26	10	0.08	1.09	26	10	0.4	1.09
37	1076	392	0.21	1.27	46	11	0.15	1.27	46	11	0.54	1.27
38	1300	359	0.31	0.92	45	17	0.15	0.91	43	17	0.48	0.91
39	161	37	0.04	0.91	19	5	0.06	0.91	18	5	0.2	0.91
40	669	241	0.15	1.46	46	17	0.15	1.45	46	17	0.59	1.45

Table 25: (Subdivisional bounding) $m = 5$ and $\epsilon = 0.001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
41	180	32	0.03	1.307	40	11	0.12	1.306	40	10	0.48	1.306
42	190011	50193	196.86	0.899	185	47	0.54	0.898	177	45	8.1	0.898
43	169	43	0.03	2.330	32	10	0.09	2.329	35	10	0.52	2.329
44	32607	7352	9.48	1.060	184	64	0.53	1.060	183	64	3.27	1.060
45	126	33	0.01	0.764	40	12	0.11	0.764	40	13	0.36	0.764
46	141045	26855	196.19	0.947	324	110	1.01	0.947	324	110	34.37	0.947
47	11298	2615	2.58	0.633	87	44	0.28	0.632	87	44	2.49	0.632
48	15797	3346	3.52	1.424	118	50	0.36	1.424	118	50	1.8	1.424
49	5243	1219	1.26	0.710	135	46	0.44	0.709	135	46	1.96	0.709
50	4714	1146	1.13	0.936	101	30	0.33	0.936	101	30	1.85	0.936

Table 26: (Subdivisional bounding) $m = 5$ and $\epsilon = 0.0001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
51	3929	822	0.62	1.4591	128	25	0.36	1.4590	159	25	2.47	1.4590
52	92372	81249	300.02+	1.8391	695	109	1.89	1.8391	102	60	301.02+	1.8388
53	98775	62933	300.01+	1.1628	371	98	1.01	1.1628	358	99	66.76	1.1628
54	84450	73458	300.01+	1.1898	815	264	2.3	1.1893	598	250	300.07+	1.1893
55	4663	1022	0.85	1.2995	154	35	0.44	1.2994	190	35	6.88	1.2994
56	7513	1824	1.89	1.0804	78	26	0.24	1.0804	78	26	2.75	1.0804
57	2142	441	0.46	1.3914	72	20	0.23	1.3914	72	20	1.7	1.3914
58	13088	2535	3.37	1.2597	92	26	0.29	1.2596	92	26	5.45	1.2596
59	1470	278	0.37	1.1513	49	10	0.16	1.1513	49	10	0.91	1.1513
60	139710	35917	300.01+	1.1702	363	128	1.09	1.1702	348	128	14.97	1.1702

Table 27: (Subdivisional bounding) $m = 6$ and $\epsilon = 0.01$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
61	34825	8793	14.61	1.99	173	86	0.54	1.98	168	87	2.66	1.98
62	1614	480	0.34	1.48	66	27	0.21	1.47	66	27	0.83	1.47
63	2450	698	0.57	1.54	99	35	0.31	1.54	99	34	1.15	1.54
64	53627	13229	28.74	1.70	255	65	0.8	1.70	255	65	3.49	1.70
65	1511	316	0.33	1.21	58	29	0.18	1.20	58	28	0.9	1.20
66	22000	6865	8.48	2.00	243	84	0.85	1.99	243	84	4.18	1.99
67	43140	12968	27.8	1.27	276	88	0.96	1.26	276	88	4.2	1.26
68	1404	471	0.41	1.36	54	26	0.19	1.36	54	26	0.66	1.36
69	1254	303	0.37	1.35	55	22	0.19	1.34	51	21	0.67	1.34
70	13420	2598	4.48	0.76	168	56	0.59	0.75	168	56	2.13	0.75

Table 28: (Subdivisional bounding) $m = 6$ and $\epsilon = 0.001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
71	59883	14515	33.62	1.333	174	50	0.54	1.332	174	50	4.08	1.332
72	83470	70144	300.01+	2.334	867	245	2.68	2.334	867	245	17.36	2.334
73	77957	74204	300.01+	1.758	1716	305	4.89	1.758	1714	304	171.45	1.758
74	46275	11483	21.95	1.137	301	97	0.93	1.136	301	97	4.75	1.136
75	124631	28887	141.85	1.122	407	140	1.24	1.122	407	140	17.01	1.122
76	110730	27008	122.37	1.935	221	99	0.77	1.934	221	99	5.8	1.934
77	98879	54891	300.01+	1.919	256	102	0.85	1.919	256	102	10.19	1.919
78	88711	62296	300.01+	1.464	1141	382	3.93	1.464	1138	381	35.17	1.464
79	13149	3181	3.74	1.235	188	59	0.61	1.234	188	59	3.64	1.234
80	85073	75245	300.01+	1.025	794	184	2.72	1.025	791	185	22.71	1.025

Table 29: (Subdivisional bounding) $m = 6$ and $\epsilon = 0.0001$.

	BNB				BNB with Relax				BNB with Sb			
No.	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt	ITs	MNs	CTs	Opt
81	89659	62898	300.01+	1.7685	557	153	1.68	1.7686	557	153	26.38	1.7686
82	95149	58807	300.01+	2.4897	496	145	1.52	2.4897	496	145	6.26	2.4897
83	84145	72901	300.01+	0.9923	2563	629	7.61	0.9923	2563	629	236.1	0.9923
84	14488	3294	4.11	1.2276	159	44	0.48	1.2275	159	44	5.79	1.2275
85	35940	7120	13.91	2.2990	164	42	0.51	2.2989	164	42	5.14	2.2989
86	77210	65907	300.02+	0.9486	1123	508	3.72	0.9452	496	283	300.05+	0.9452
87	89153	60281	300.01+	1.6754	511	117	1.82	1.6754	511	117	119.27	1.6754
88	88063	68032	300.01+	1.2184	601	138	2.15	1.2185	601	138	24.4	1.2185
89	63220	13205	39.5	1.4159	249	68	0.87	1.4158	249	68	4.69	1.4158
90	106371	54162	300.01+	1.1215	777	229	2.63	1.1216	771	230	34.94	1.1216

REFERENCES

[1] Beraldi, P. and Ruszczyński, A., "A branch and bound method for stochastic integer problems under probabilistic constraints," *Optimization Methods and Software*, vol. 17, pp. 359–382, 2002.

[2] Beraldi, P. and Ruszczyński, A., "The probabilistic set covering problem," *Operations Research*, vol. 50, pp. 956–967, 2002.

[3] Bertsimas, D. and Tsitsiklis, J. N., *Introduction to Linear Optimization.* Belmont, Massachusetts: Athena Scientific, 1997.

[4] Charnes, A. and Cooper, W. W., "Chance-constrained programming," *Management Science*, vol. 6, pp. 73–89, 1959.

[5] Dentcheva, D., Prékopa, A., and Ruszczyński, A., "Concavity and efficient points of discrete distributions in probabilistic programming," *Mathematical Programming*, vol. 89, pp. 55–77, 2000.

[6] Dentcheva, D., Prékopa, A., and Ruszczyński, A., "On convex probabilistic programming with discrete distributions," *Nonlinear Analysis*, vol. 47, pp. 1997–2009, 2001.

[7] Falk, J. E. and Soland, R. M., "An algorithm for separable nonconvex programming problems," *Management Science*, vol. 15, no. 9, pp. 500–569, 1969.

[8] Horst, R., Pardalos, P. M., and Thoai, N. V., *Introduction to Global Optimization.* Kluwer Academic Publishers, Dordrecht, 2001.

[9] Horst, R. and Tuy, H., *Global Optimization: Deterministic Approaches.* Berlin, Germany: Springer-Verlag, 1996.

[10] Land, A. and Doig, A., "An automatic method for solving discrete programming problems," *Econometrica*, pp. 497–520, 1960.

[11] Lasserre, J. B., "Semidefinite programming vs. lp relaxations for polynomial programming," *Mathematics of Operations Research*, vol. 27, no. 2, pp. 347–360, 2002.

[12] Li, D., Sun, X. L., Biswal, M. P., and Gao, F., "Convexification, concavification and monotonization in global optimization," *Annals of Operations Research*, vol. 105, pp. 213–226, 2001.

[13] LINDO Systems Inc., "LINDO API 2.0 and LINGO 8.0," `http://www.lindo.com/`.

[14] Nemhauser, G. L. and Wolsey, L. A., *Integer and Combinatorial Optimization.* New York, USA: Wiley-Interscience, 1999.

[15] Neumaier, A., "Complete search in continuous global optimization and constraint satisfaction," to appear in: Acta Numerica 2004 (A. Iserles, ed.) Cambridge University Press 2004; `http://www.mat.univie.ac.at/~neum/papers.html#glopt03`.

[16] Pardalos, P. M., Romeijn, H. E., and Tuy, H., "Recent developments and trends in global optimization," *Journal of Computational and Applied Mathematics*, vol. 124, pp. 209–228, 2000.

[17] Phuong, N. T. H. and Tuy, H., "A unified monotonic approach to generalized linear fractional programming," *Journal of Global Optimization*, vol. 26, pp. 229–259, 2004.

[18] Prékopa, A., "Contributions to the theory of stochastic programming," *Mathematical Programming*, vol. 4, pp. 202–221, 1973.

[19] Prékopa, A., "Sharp bounds on probabilities using linear programming," *Operations Research*, vol. 38, pp. 227–239, 1990.

[20] Prékopa, A., *Stochastic Programming.* Dordrecht, Ther Netherlands: Kluwer Academic Publishers, 1995.

[21] Ruszczyński, A., "Probabilistic programming with discrete distributions and precedence constrained knapsack polyhedra," *Mathematical Programming*, vol. Ser. A93, pp. 195–215, 2002.

[22] Sahinidis, N. V., "Optimization under uncertainty: state-of-art and opportunities," *Computers & Chemical Engineering*, vol. 28, pp. 971–983, 2004.

[23] Sahinidis, N., "BARON: A global optimization software," `http://archimedes.scs.uiuc.edu/baron/baron.html`.

[24] Sen, S., "Relaxations for probabilistically constrained programs with discrete random variables," *Operations Research Letters*, pp. 81–86, 1992.

[25] Sen, S. and Higle, J. L., "An introductory tutorial on stochastic linear programming models," *Interfaces*, vol. 29, no. 2, pp. 31–61, 1999.

[26] Sherali, H. D., "Global optimization of nonconvex polynomial programming problems having rational exponents," *Journal of Global Optimization*, vol. 12, pp. 267–283, 1998.

[27] Sherali, H. D. and Tuncbilek, C. H., "Comparison of two reformulation-linearization technique based linear programming relaxations for polynomial programming problems," *Journal of Global Optimization*, vol. 10, pp. 381–390, 1997.

[28] Sherali, H. D. and Tuncbilek, C. H., "New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems," *Operations Research Letters*, vol. 21, pp. 1–9, 1997.

[29] Soland, R. M., "An algorithm for separable nonconvex programming problems ii: Nonconvex constraints," *Management Science*, vol. 17, no. 11, pp. 759–773, 1971.

[30] Tawarmalani, M. and Sahinidis, N. V., *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications.* Kluwer Academic Publishers, Dordrecht, 2002.

[31] Toh, K.-A., "Global optimization by monotonic transformation," *Computational Optimization and Applications*, vol. 23, pp. 77–99, 2002.

[32] Tuy, H., "Monotonic optimization: Problems and solution approaches," *SIAM Journal of Optimization*, vol. 11, no. 2, pp. 464–494, 2000.

[33] Tuy, H. and Al-Khayyal, F., "Monotonic optimization revisited," 2004. `http://www.math.ac.vn/library/download/e-print/03/pdf/htuy23.pdf`.

[34] Tuy, H., Al-Khayyal, F., and Ahmed, S., "Polyblock algorithms revisited," *Working paper*, 2003.

[35] Tuy, H., Al-Khayyal, F., and Thach, P. T., "Monotonic optimization: Branch and cut methods," Essays on Global Optimization, (C. Audet, B. Jaumard and P. Hausen, Eds.), Springer-Verlag, to appear in 2005.

[36] Tuy, H., Minoux, M., and Hoai-Phuong, N. T., "Discrete monotonic optimization with application to a discrete location problem," `http://www.math.ac.vn/library/download/e-print/04/pdf/htuy0403.pdf`.

[37] Tuy, H., Thach, P. T., and Konno, H., "Optimization of polynomial fractional functions," *Journal of Global Optimization*, vol. 29, pp. 19–44, 2004.

VITA

Myun-Seok Cheon was born in Ulsan, South Korea on August 20, 1973. In 1992, he entered University of Ulsan and received a Bachelors degree in Industrial Engineering in 1999. He left for Atlanta to continue his education at the Georgia Institute of Technology in the School of Industrial and Systems Engineering. At Georgia Tech, he earned a Master of Science in Industrial Engineering in December, 2000. In May, 2005, he was awarded the Doctor of Philosophy in Industrial Engineering.