Discrete and Continuous Nonconvex Optimization: Decision Trees, Valid Inequalities, and Reduced Basis Techniques

Evrim Dalkiran

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Industrial and Systems Engineering

Hanif D. Sherali, Chair
Douglas R. Bish
Barbara M. Fraticelli
Subhash C. Sarin

March 31, 2011
Blacksburg, Virginia

Keywords: Decision tree optimization; Reformulation-Linearization Technique; polynomial programs; semidefinite cuts; bound-grid-factor constraints; reduced basis techniques; mixed-integer nonlinear program; decision analysis; global optimization; branch-and-bound.

Discrete and Continuous Nonconvex Optimization: Decision Trees, Valid Inequalities, and Reduced Basis Techniques

Evrim Dalkiran

(ABSTRACT)

This dissertation addresses the modeling and analysis of a strategic risk management problem via a novel decision tree optimization approach, as well as development of enhanced Reformulation-Linearization Technique (RLT)-based linear programming (LP) relaxations for solving nonconvex polynomial programming problems, through the generation of valid inequalities and reduced representations, along with the design and implementation of efficient algorithms. We first conduct a quantitative analysis for a strategic risk management problem that involves allocating certain available failure-mitigating and consequence-alleviating resources to reduce the failure probabilities of system safety components and subsequent losses, respectively, together with selecting optimal strategic decision alternatives, in order to minimize the risk or expected loss in the event of a hazardous occurrence. Using a novel decision tree optimization approach to represent the cascading sequences of probabilistic events as controlled by key decisions and investment alternatives, the problem is modeled as a nonconvex mixed-integer 0-1 factorable program. We develop a specialized branch-and-bound algorithm in which lower bounds are computed via tight linear relaxations of the original problem that are constructed by utilizing a polyhedral outer-approximation mechanism in concert with two alternative linearization schemes having different levels of tightness and complexity. We also suggest three alternative branching schemes, each of which is proven to guarantee convergence to a global optimum for the underlying problem. Extensive computational results and sensitivity analyses are presented to provide insights and to demonstrate the efficacy of the proposed algorithm. In particular, our methodology outperformed the commercial software BARON (Version 8.1.5), yielding a more robust performance along with an 89.9% savings in effort on average.

Next, we enhance RLT-based LP relaxations for polynomial programming problems by developing two classes of valid inequalities: *v-semidefinite cuts* and *bound-grid-factor* constraints. The first of these uses concepts derived from semidefinite programming. Given an RLT relaxation, we impose positive semidefiniteness on suitable dyadic variable-product matrices, and correspondingly derive implied semidefinite cuts. In the case of polynomial programs, there are several possible variants for selecting such dyadic variable-product matrices for imposing positive semidef-

initeness restrictions in order to derive implied valid inequalities, which leads to a new class of cutting planes that we call *v-semidefinite cuts*. We explore various strategies for generating such cuts within the context of an RLT-based branch-and-cut scheme, and exhibit their relative effectiveness towards tightening the RLT relaxations and solving the underlying polynomial programming problems, using a test-bed of randomly generated instances as well as standard problems from the literature. Our results demonstrate that these cutting planes achieve a significant tightening of the lower bound in contrast with using RLT as a stand-alone approach, thereby enabling an appreciable reduction in the overall computational effort, even in comparison with the commercial software BARON. Empirically, our proposed cut-enhanced algorithm reduced the computational effort required by the latter two approaches by 44% and 77%, respectively, over a test-bed of 60 polynomial programming problems.

As a second cutting plane strategy, we introduce a new class of *bound-grid-factor* constraints that can be judiciously used to augment the basic RLT relaxations in order to improve the quality of lower bounds and enhance the performance of global branch-and-bound algorithms. Certain theoretical properties are established that shed light on the effect of these valid inequalities in driving the discrepancies between RLT variables and their associated nonlinear products to zero. To preserve computational expediency while promoting efficiency, we propose certain concurrent and sequential cut generation routines and various grid-factor selection rules. The results indicate a significant tightening of lower bounds, which yields an overall reduction in computational effort of 21% for solving a test-bed of 15 challenging polynomial programming problems to global optimality in comparison with the basic RLT procedure, and over a 100-fold speed-up in comparison with the commercial software BARON.

Finally, we explore equivalent, reduced size RLT-based formulations for polynomial programming problems. Utilizing a basis partitioning scheme for an embedded linear equality subsystem, we show that a strict subset of RLT defining equalities imply the remaining ones. Applying this result, we derive significantly reduced RLT representations and develop certain coherent associated branching rules that assure convergence to a global optimum, along with static as well as dynamic basis selection strategies to implement the proposed procedure. In addition, we enhance the RLT relaxations with *v*-semidefinite cuts, which are empirically shown to further improve the relative performance of the reduced RLT method over the usual RLT approach. Computational results presented using a test-bed of 10 challenging polynomial programs to evaluate the different reduction strategies demonstrate that our superlative proposed approach achieved more than a four-fold

improvement in computational effort in comparison with both the commercial software BARON and a recently developed open-source code, Couenne, for solving nonconvex mixed-integer nonlinear programming problems. Moreover, our approach robustly solved all the test cases to global optimality, whereas BARON and Couenne were jointly able to solve only a single instance to optimality within the set computational time limit, having an unresolved average optimality gap of 260% and 437%, respectively, for the other nine instances.

This dissertation makes several broader contributions to the field of nonconvex optimization, including factorable, nonlinear mixed-integer programming problems. The proposed decision tree optimization framework can serve as a versatile management tool in the arenas of homeland security and health-care. Furthermore, we have advanced the frontier for tackling formidable nonconvex polynomial programming problems that arise in emerging fields such as signal processing, biomedical engineering, materials science, and risk management. An open-source software using the proposed reduced RLT representations, semidefinite cuts, bound-grid-factor constraints, and range reduction strategies, is currently under preparation. In addition, the different classes of challenging polynomial programming test problems that are utilized in the computational studies conducted in this dissertation have been made available for other researchers via the Web-page http://filebox.vt.edu/users/dalkiran/website/. It is our hope and belief that the modeling and methodological contributions made in this dissertation will serve society in a broader context through the myriad of widespread applications they support.

This work has been partially supported by the *National Science Foundation* under Grant No. CMMI-0969169.

Dedication

To My Parents and Brother

Acknowledgments

I would like to extend a special thanks to my advisor and mentor Prof. Hanif D. Sherali. I can see how far I have come along in my career with your support and guidance throughout my Ph.D. journey. I must have done some incredibly good things in the first 25 years of my life that I have the gift of having you as an advisor. You have been always very patient and considerate, even when I made trivial mistakes. I was so motivated after every chat in the hallway or in your office that I would fly back to my office and start working as a super scientist. I have learned a lot from your wise, humble, hardworking, and thoughtful character. My only wish is to be strong and intrepid to follow your path. I cannot thank you enough for taking me as a graduate student. You have changed my life and I will be indebt forever for what you have done for me. I would also like to extend my gratitude to my committee members Prof. Subhash C. Sarin, Dr. Douglas R. Bish, Dr. Barbara M. Fraticelli, and Dr. Yasemin Merzifonluoglu Uzgoren for their support and contribution. You have always been very collaborative throughout my journey. Finally, I want to thank ISE administrative staff members, especially Kim Ooms and Will West, for their help.

I would like to thank my parents for supporting me with my decisions. They let me follow the path I dreamed, but also make sure that I know they are always with me when I need them. I am also grateful to my parents for giving me such a wonderful brother. I am very happy to see that my little brother has already become a very mature friend along the way.

I have precious friends who shared my happiness and sadness. I would like to express my gratitude to Saeideh Fallah Fini, Ozlem Cavus, Buket Avci, Mana Afshari, Sebnem Gurun Ozeren, Saliha Oral, and Ali Hashemi (in completely random order). I also want to send my heartfelt thanks to my friends: Brian, Justin, Eser, Melis, Caner, Sedef, Atila, Mustafa, Elif, Ahmed, Moataz, Weiping, Esra, Guray, Burak, Onur, and Susan.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This dissertation is concerned with discrete and continuous nonconvex optimization problems and, in particular, addresses four specific contributions in this context: decision tree optimization approach, semidefinite cuts, bound-grid-factor constraints, and reduced basis techniques. In this chapter, we first discuss the overarching framework that unifies these topics, and then provide an introduction to each of these in turn. We conclude the chapter by describing the organization of the remainder of this dissertation.

1.1 Motivation

Discrete and continuous nonconvex problems arise in various science and engineering disciplines, including risk management, option pricing, chemical process design, production planning, and location-distribution contexts. In this study, we utilize branch-and-bound or branch-and-cut algorithms for solving discrete and continuous nonconvex programs. In order to effectively optimize the problem, we need to simultaneously address three characteristics of the model: good mathematical formulation, tight relaxations of the original problem, and computational tractability.

First and foremost, good mathematical formulations of discrete and continuous nonconvex programming problems strongly govern its effective solvability. Facility location, production planning and control, and network design problems are among discrete optimization problems that

are extensively discussed as striking examples for which good mathematical formulations provide a manyfold increase in the size of problems that can be solved in practice (see Nemhauser and Wolsey [56], for example). Alternative relaxations pertaining to equivalent formulations for nonconvex continuous problems have likewise aroused the attention of researchers. For example, Shor [79] introduced a quadrification process for polynomial programming problems, where Sherali and Tuncbilek [73] demonstrated that the RLT approach applied directly to the original formulation generates theoretically tighter relaxations than simultaneously accommodating all possible quadratic reformulations, thereby greatly enhancing problem solvability. In various parts of this dissertation, we theoretically and empirically compare equivalent formulations for discrete and continuous nonconvex optimization problems. In particular, we introduce two equivalent formulations for a decision tree optimization problem. Moreover, we develop two alternative, equivalent formulations for nonconvex polynomial programming problems that contain a linear equality subsystem, and we present theoretical as well as extensive computational analyses for several judiciously designed algorithmic strategies. In particular, a hybrid algorithmic approach that suitably composes the aforementioned two alternative formulations using problem specific attributes is shown to greatly enhance the overall computational performance.

Second, it is critical to derive tight relaxations of the original problem that can be embedded within branch-and-bound or branch-and-cut approaches. Linear programming (LP) relaxations, Lagrangian relaxations, and semidefinite programming (SDP) relaxations afford some popular prevailing lower bounding mechanisms for designing solution procedures for discrete and continuous nonconvex (minimization) problems. Among the LP-based relaxation approaches, we utilize polyhedral outer-approximation methods as well as the Reformulation-Linearization Technique (RLT) of Sherali and Adams [68] and Sherali and Tuncbilek [71]. From a practical point of view, LP relaxations have the advantage of ease and robustness in solution using widely available commercial software. Although we do not pursue this option herein, such LP-based bounds can be further tightened through the use of suitable Lagrangian relaxations that do not satisfy the so-called integrality property (see Fisher [20] and Geoffrion [25]). Furthermore, over the last two decades, SDP relaxations have enjoyed a flurry of attention with applicability in various areas including combinatorial, quadratic, and polynomial optimization, and control theory, just to name a few (see Wolkowicz et al. [91] and Vandenberghe and Boyd [86]). Whereas SDP relaxations are theoretically appealing, they are yet computationally challenging to solve for instances of practical size. In our work, we tighten RLT-based linear programming relaxations by appending cuts that are generated using semidefinite programming constructs. Hence, we capture the strength of both SDP-

and RLT-based relaxations while retaining linearity, thus preserving computational tractability. In a similar fashion, we enhance RLT-based relaxations via a new class of valid inequalities called *bound-grid-factor constraints*, where we advocate a judicious selection of grid-factors and develop cut generation strategies that compose these factors with the usual RLT bound-factor product constraints in order to enhance computational performance. Furthermore, in our analysis of decision tree optimization problems, we employ LP-based outer-approximations and linearizations for logarithmic functions and monomials of binary variables. More specifically, for the logarithmic functions, we utilize a polyhedral outer-approximation using an elegant cut generation scheme that guarantees a user-defined maximum error level, and for monomials of 0-1 variables, we theoretically and computationally compare two alternative linearization strategies.

Finally, the tightness of relaxations should be carefully balanced with computational tractability. Indeed, the RLT approach for discrete optimization problems is capable of algebraically generating the ideal convex hull representation, but aside from specially structured problems, such a reformulation can be exponentially large, thus inhibiting computational tractability and defeating the purpose of generating tight relaxations. Since a series of relaxations is typically solved within a branch-and-bound framework, the overall computational effectiveness depends on the solvability of the relaxations as well as on the required number of nodes that need to be enumerated to achieve a given solution accuracy. Hence, for example, selecting only a subset of highly effective cuts, or constructing partial convex hull representations could effectively balance the tightness and computational tractability of relaxations. We implement such a reduction strategy for RLT-based LP relaxations of polynomial programming programs that contain a linear equality subsystem by partitioning the variables according to a suitably defined basis for the latter system. The resulting relaxation eliminates a significant portion of the RLT constraints, while still retaining the strength of the derived bounds, and guaranteeing theoretical global convergence. Similarly, we provide mechanisms for selecting only a specific subset of v-semidefinite cuts and bound-grid-factor constraints that enable the design of the most effective relaxations for polynomial programming problems. Moreover, the sparsity of cuts is also often critical in improving computational tractability. Relaxations that are formulated with dense cuts typically become impractical for branch-and-bound approaches due to an increase in computational effort to solve them, even though these cuts might significantly tighten the model representation.

1.2 Scope of Research

This dissertation studies discrete and continuous nonconvex problems, particularly decision tree and polynomial optimization problems, and develops tight and computationally tractable relaxations that rely on good initial mathematical formulations, and designs effective algorithmic strategies for the resulting model that guarantee convergence to global optimum. The first part of this dissertation conducts a quantitative analysis for a strategic risk management problem that seeks to minimize the expected loss in the aftermath of a hazardous occurrence. Several such risk management problems arise in the context of chemical process industries, power plants, and biological invasions, to name a few. Also, particularly in the finance literature, risk is commonly represented as a combination of expected loss and its variance (see Markowitz [50, 51]). Although we focus on minimizing the expected loss in our model, we provide a discussion on how variance aspects can readily be accommodated within our analysis. For the considered application, the novel idea modeled is to reduce the failure probabilities of system safety components and subsequent losses by suitably allocating certain available failure-mitigating and consequence-alleviating resources, respectively, along with selecting optimal decision alternatives, subject to additional budgetary restrictions. Using a novel decision tree approach, the risk management problem is modeled as a mixed-integer 0-1 factorable program. We perform a polyhedral analysis for the formulated model by introducing suitable outer-approximation mechanisms for the inherent logarithmic functions in a manner that bounds the maximum error to a user-specified level, and by deriving tight linearizations for the monomial terms that effectively define the objective function. Moreover, we tighten the model representation via optimality-induced valid inequalities and by utilizing variable fixing procedures. As demonstrated by recent studies (see also [16, 65, 72, 74]), range reduction strategies result in significantly enhancing computational performance. In the decision tree optimization context, we perform domain reduction by sequentially solving a pair of linear programs to tighten the variable bounding intervals. We then develop a specialized branch-and-bound algorithm in which lower bounds are computed via tight relaxations of the mixed-integer 0-1 factorable program, and where we explore three alternative branching schemes, all of which guarantee convergence to a global optimum. We present extensive computational results, along with insightful sensitivity analyses, to demonstrate the efficacy of the proposed approach.

Next, we turn our attention to the solution of polynomial programming problems, focusing on the Reformulation-Linearization Technique (RLT) of Sherali and Adams [68] and Sherali and

Tuncbilek [71]. This class of programs arises in various applications in the context of production planning and control, location and distribution, chemical process and engineering design problems, signal processing, biomedical engineering, and risk management (He et al. [28] and Sherali and Adams [68]). Consider the following generic formulation of a polynomial program:

$$\textbf{PP:}\quad \text{Minimize}\quad \phi_0(x) \tag{1.1a}$$

subject to

$$\phi_r(x) \geq \beta_r, \forall r = 1, \ldots, R_1 \tag{1.1b}$$

$$\phi_r(x) = \beta_r, \forall r = R_1 + 1, \ldots, R \tag{1.1c}$$

$$Ax = b \tag{1.1d}$$

$$x \in \Omega \equiv \{0 \leq l_j \leq x_j \leq u_j < \infty, \forall j \in \mathcal{N}\}, \tag{1.1e}$$

where

$$\phi_r(x) \equiv \sum_{t \in T_r} \alpha_{rt} \Big[\prod_{j \in J_{rt}} x_j \Big], \text{ for } r = 0, \ldots, R.$$

Here, T_r is an index set for the terms defining $\phi_r(.)$, and α_{rt} are real coefficients for the monomial terms $\prod_{j \in J_{rt}} x_j$, $t \in T_r$, $r = 0, \ldots, R$, where each J_{rt} is a multi-set that might contain a repetition of indices from $\mathcal{N} \equiv \{1, \ldots, n\}$. Denoting $\mathcal{N}^\delta \equiv \{\mathcal{N}, \ldots, \mathcal{N}\}$ to be composed of δ replicates of $\mathcal{N}$, where δ is the maximum degree of the polynomial functions defining PP, we have that each $J_{rt} \subseteq \mathcal{N}^\delta$, with $1 \leq |J_{rt}| \leq \delta$, $\forall t \in T_r$, $r = 0, \ldots, R$. Here, we have especially identified the presence of a linear equality system (1.1d), where A is $m \times n$ of rank $m < n$, and where (1.1c) then accommodates any other nonlinear equality restrictions defining the model formulation. Sherali and Tuncbilek [71] describe an RLT methodology for constructing tight linear relaxations for solving polynomial programs to global optimality. Here, we significantly enhance this RLT-based approximation by introducing two classes of valid inequalities, namely, semidefinite cuts and bound-grid-factor constraints.

Recent works underscore the strength of SDP relaxations for polynomial programming problems (e.g., see Anstreicher [5], Lasserre [43], Laurent and Rendl [45]). However, SDP relaxations, or RLT relaxations enhanced with SDP constraints in the form of linear matrix inequalities, are computationally challenging to solve, as compared with LP-based RLT relaxations. Yet, as demonstrated by Anstreicher [6], a combination of RLT and SDP relaxations theoretically dominates many other alternative relaxations presented in the literature, which variously utilize convex envelopes, convex under approximations such as the αBB approach (Androulakis et al. [4]), and

purely SDP relaxations. Hence, instead of relying on SDP solvers, we utilize RLT-based LP relaxations enhanced via *v*-semidefinite cuts that capture the strength of the underlying SDP relaxation, but retain linearity, and thereby facilitate the use of robust, efficient LP solvers. Given an RLT relaxation, we impose positive semidefiniteness on suitable dyadic variable-product matrices. In this procedure, given an optimal solution to the relaxed problem, we compose a suitable dyadic variable-product matrix, and in case this matrix evaluated at the current LP solution is not positive semidefinite, the procedure automatically generates a *v*-semidefinite cut to delete this LP solution. There are several possible alternatives for deriving such variable-product matrices for imposing positive semidefiniteness, thus yielding different classes of cuts. We propose and compare different cut generation strategies based on defining several attractive variable-product matrices. The computational results demonstrate that *v*-semidefinite cuts achieve a significant tightening for RLT-based LP relaxations and enable an appreciable reduction in the overall computational effort. Note that the proposed *v*-semidefinite cuts can be likewise utilized to solve the more general class of factorable programming problems within the framework of the sequential polynomial programming approximation approach discussed in Sherali and Wang [75].

As another class of valid inequalities, we introduce bound-grid-factor constraints that can be used to judiciously augment the usual RLT bound-factor constraints in order to strengthen the derived LP relaxations. Toward this end, we identify suitable grid-points for constructing grid-factors in a fashion similar to the definition of bound-factors. We then derive bound-grid-factor constraints by taking the products of the squared grid-factors at most $\lfloor \delta/2 \rfloor$ at a time, where the resultant constraint expression is further multiplied individually by the bound-factors until the degree of the constraint becomes δ. We prudently select a subset of variables for which suitable grid-points are identified, and we generate a particular class of associated bound-grid-factor constraints with the motivation to curtail the overall computational effort. Computational results demonstrate that a significant improvement accrues with an appropriate inclusion of such bound-grid-factor constraints.

Finally, we address equivalent, reduced size RLT-based reformulations for polynomial programming problems that utilize a basis partitioning scheme for an embedded linear equality subsystem, for which we prove that a strict subset of RLT defining identities imply the remaining ones. Many such representations are available by using different bases, thereby yielding relaxations of differing strengths. We explore several alternative basis selection techniques in the context of both static and dynamic approaches. As an alternative to deriving such reduced RLT representations,

the linear equality subsystem could be directly employed to eliminate variables via a substitution process to formulate an equivalent problem in a lower dimensional space, to which the usual RLT procedure can be applied. Here too, the resulting reduced size relaxation is basis dependent, and empirically turns out to be highly sensitive to the selection of the basis and the size of the underlying relaxations. An extensive computational analysis is conducted to compare these three equivalent representations, including the original RLT procedure. Based on these results, a hybrid algorithm that judiciously selects either the reduced RLT representation, or the RLT approach applied to an equivalent problem reformulated in a lower dimensional space, both for a particular basis selection mechanism, emerges as a highly effective approach that significantly reduces the overall computational effort.

1.3 Organization of the Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 presents a brief literature review on decision trees and risk management problems, semidefinite programming relaxations for polynomial programming problems, grid-factor constraints, and reduced RLT representations.

In Chapter 3, we formulate a mathematical model for the Decision Tree Optimization (DTO) problem, and then develop a suitable reformulation along with a tight lower-bounding representation through some transformations, polyhedral outer-approximations, and valid inequalities, in concert with two alternative linearization schemes. We next derive a specialized branch-and-bound procedure to solve the DTO problem and establish its convergence. Sensitivity analyses and computational results are presented for a hypothetical case study, in addition to algorithmic performance results and comparisons using a set of simulated problems.

Chapter 4 begins with a description of the semidefinite programming cuts previously proposed by Sherali and Fraticelli [70], and then develops a new cut generation routine using restricted eigenvalue-eigenvector computations. The discussion next proceeds to motivate the class of v-semidefinite cuts via an illustrative example. Accordingly, the generation of several such cuts based on appropriately composed vectors v is explored, and we illustrate these cuts using an example. Extensive computational results are reported using randomly generated test cases, as well as standard polynomial programming problems from the literature that arise in various applications.

Chapter 5 introduces the class of bound-grid-factor constraints. We begin by presenting some related theoretical results to glean insights into the nature and strength of these constraints. We then propose certain concurrent and sequential constraint generation routines as well as various schemes for locating grid-points, and we provide an illustrative example. The implemented branch-and-bound algorithm that incorporates bound-grid-factor constraints is described next, and detailed computational results are reported for the proposed procedure as well as for the basic RLT method and for the commercial software BARON.

Chapter 6 addresses equivalent, reduced size RLT representations for polynomial programming problems. To set the stage and facilitate the presentation, we first consider quadratic polynomial programming problems having an embedded linear equality subsystem, and present the proposed equivalent, reduced RLT representation for this structured problem. The theory is then extended to address more general higher order polynomial programs, which is the main focus of this study, and several static and dynamic basis selection methods are described for constructing such reduced representations. Computational results are presented next to compare the different relaxations, which prompts the design of a particularly effective hybrid approach that is recommended for future implementation.

Chapter 7 summarizes the scope of this dissertation and reiterates the main conclusions of this research effort, and also provides recommendations for future research directions.

Chapter 2

Literature Review

In this chapter, we provide a brief literature review on decision trees and risk management problems, semidefinite programming relaxations, grid-factor constraints, and reduced RLT representations.

2.1 Decision Trees and Risk Management Problems

In this section, we provide a brief review of the literature pertaining to the field of risk management and its applications. In particular, Jiang et al. [34] studied maintenance selection and scheduling under tight budgetary and labor constraints to maximize the risk reduction. A composite heuristic using linear programming relaxations and dynamic programming was proposed to solve the large-scale integer programming problem. In another study, Dillon et al. [17] developed a decision tool for allocating a limited budget amongst design and residual investments that involve reinforcement (for mitigating technical failures) and initial budget reserves (for mitigating managerial failures). Mehr and Tumer [54] analyzed resource allocations in the form of a portfolio selection problem in which risk was related to both the likelihood and consequence of an undesirable event, and where each unit of resource allocation reduced the risk by a random amount. Sherali et al. [78] also considered resource allocations to minimize risk or the expected consequence in a given system. In contrast to Mehr and Tumer [54], they differentiated between investment decisions for failure-

mitigation and consequence-alleviation. A logit model was used to represent the relationships between investments and failure probabilities instead of adopting a linearity assumption as in Mehr and Tumer [54]. Furthermore, the probability of a final consequence was given via an event tree as a polynomial function of the probabilities of cascading events as opposed to using an assigned probability. The overall event tree optimization problem was modeled as a continuous nonconvex factorable program, and an equivalent transformed reformulation of the problem was solved using the commercial global optimization software BARON (Tawarmalani and Sahinidis [84]).

Several other risk management applications where decision trees arise are discussed in the literature. Beim and Hobbs [11] utilized event trees to estimate the lock closure risks due to vessel accidents and nonstructural failures, while Acosta and Siu [1] analyzed steam generator tube rupture accidents in a power plant. Alternatively, Ulerich and Powers [85], Andrews and Bartlett [2], and Hayes [27] adopted fault trees for modeling chemical processes, firewater deluge systems, and biological invasions, respectively. Dugan et al. [18] developed a fault tree modeling and analysis tool, which decomposes the tree into static and dynamic subtrees that were solved using binary decision diagrams and Markov models, respectively. Another such computer-automated fault tree analysis tool that can be applied to chemical process industries was designed by Khan and Abbasi [37]. For fault tree management problems, Rauzy [62] proposed new algorithms based on binary decision diagrams that enable an efficient computation of minimal cuts along with deducing the probability of the root event. Dutuit and Rauzy [19] described a linear-time algorithm to detect independent subtrees in a fault tree by which computational costs can be reduced by means of a divide-and-conquer technique. Using the binary decision diagram approach of Rauzy [62] and Dutuit and Rauzy [19], Sinnamon and Andrews [81, 82] calculated the exact values of the top event parameters efficiently in contrast with the approximations produced by using traditional kinetic tree theory approaches. In addition, Furuta and Shiraishi [24], Kenarangui [36], and Huang et al. [33] suggested using the fuzzy-set approach in order to enable the use of verbal statements for possibility measures instead of requiring the specification of event probabilities.

The concept of risk is used in different forms within diverse fields. Similar to the aforementioned papers, we adopt the classical definition of *risk* as the *expected loss* (see Nikulin [57], Ross [63]), or the expected value of the damage (Kaplan and Garrick [35]), whereas a combination of expected loss and its variance is another common representation of risk, especially in finance. Markowitz [50, 51] pointed out that considering expected returns alone in portfolio optimization fails to sufficiently accommodate diversification, and accordingly proposed the use of

mean-variance models in this context. Artzner et al. [9] indicated that risk is often interpreted in finance as a change in value between two time periods. They defined risk in terms of the future net worth of a set of financial instruments at a specific point in time, and they introduced coherent measures of risk that satisfy certain translation invariance, subadditivity, positive homogeneity, and monotonicity properties. Föllmer and Schied [23] proposed alternative convex measures of risk that satisfy the weaker property of convexity in lieu of the foregoing subadditivity and positive homogeneity properties. Ruszczyński and Shapiro [64] studied the optimization of convex risk functions, and investigated certain continuity, differentiability, and duality relationships. Note that the expected loss risk measure is coherent, and hence, convex.

2.2 Semidefinite Cuts and Bound-Grid-Factor Constraints

This section presents a brief literature review pertaining to semidefinite programming relaxations, as well as related to our proposed valid inequalities derived for enhancing RLT-based LP relaxations, viz, *v*-semidefinite and bound-grid-factor cuts for solving polynomial programming programs.

Consider the polynomial program **PP** of order δ given by Equation (1.1) in Chapter 1. Observe that the dyadic variable-product matrix $[xx^T]$ is *symmetric and positive semidefinite* (denoted $\succeq 0$), and we can require that $M_0 \equiv [xx^T]_L \succeq 0$, where $[.]_L$ represents the linearization of the expression $[.]$ under the substitution $X_{ij} = x_i x_j, \forall i, j = 1, \ldots, n,\ i \leq j$. As an alternative variable-product matrix, we can define $[x_{(1)} x_{(1)}^T]$, where $x_{(1)}^T = [1, x^T]$, and require that $M_1 \equiv [x_{(1)} x_{(1)}^T] \succeq 0$. In lieu of solving the resulting semidefinite programming relaxations obtained by imposing such relationships, which would detract from the robustness and efficiency of the algorithm (especially as the size of the problem increases), Sherali and Fraticelli [70] recommended utilizing linear programming relaxations in concert with a class of RLT constraints known as *semidefinite cuts* that are predicated on the fact that

$$M_1 \succeq 0 \Leftrightarrow \alpha^T M_1 \alpha = [(\alpha^T x_{(1)})^2]_L \geq 0, \forall \alpha \in \mathbb{R}^{n+1}, \|\alpha\| = 1. \tag{2.1}$$

Accordingly, given a certain solution $(\bar{x}, \bar{X})$ to the RLT relaxation of the underlying quadratic polynomial program that violates some RLT defining identities, Sherali and Fraticelli [70] invoked

(2.1) to check in polynomial time, having a worst-case complexity $O(n^3)$, whether or not $\bar{M}_1 \succeq 0$, where $\bar{M}_1$ evaluates M_1 at the solution $(\bar{x}, \bar{X})$. In case $\bar{M}_1$ is not positive semidefinite, then the foregoing process was shown to also automatically generate an $\bar{\alpha} \in \mathbb{R}^{n+1}$ such that $\bar{\alpha}^T \bar{M}_1 \bar{\alpha} < 0$, which in turn yields the *semidefinite cut*:

$$\bar{\alpha}^T M_1 \bar{\alpha} = [(\bar{\alpha}^T x_{(1)})^2]_L \geq 0. \tag{2.2}$$

Several alternative polynomial-time schemes for generating rounds of cuts (2.2) based on suitable vectors $\bar{\alpha}$ were described and computationally exhibited to yield a substantial reduction in effort (by a factor of 2-3) for solving the class of quadratic programs defined over a simplex, in comparison with an RLT approach that does not employ such cuts.

For linearly constrained quadratic programming problems, relaxations derived from constructing convex envelopes of partitioned product terms in the objective function are most popularly used in the literature. As shown in Anstreicher [6], these types of relaxations are dominated by instead incorporating within the constraints the set $\mathscr{C} \equiv$ convex hull of $\left\{ \binom{1}{x} \binom{1}{x}^T : x \in F \right\}$, where F is the feasible region defined by the linear constraints, while adopting the RLT linearization step. Unfortunately, the latter relaxation is not computable except for some special cases. Anstreicher and Burer [7] showed that if F is a simplex with $n \leq 4$, then $\mathscr{C}$ can be represented in terms of a positive semidefinite and componentwise nonnegative matrix X that represents the dyadic product xx^T. Furthermore, semidefinite constraints together with RLT constraints were shown to precisely define $\mathscr{C}$ for box-constrained quadratic programs with $n = 2$. In Anstreicher [6], two computable relaxations were examined and compared against a synergistic use of RLT and semidefinite constraints. The first such relaxation employed αBB underestimators (see Androulakis et al. [4]), and was shown to be dominated by one utilizing semidefinite constraints and certain RLT linear inequalities. The second relaxation utilized underestimators derived via a difference of convex functions (DC), which represents a stronger generalization of αBB-based bounds, but was also shown to be dominated by an RLT relaxation enhanced via semidefiniteness constraints, $X \succeq 0$. Related computational results were presented in Anstreicher [5], where the optimality gaps at the root node for SDP, RLT, and joint SDP and RLT relaxations (denoted SDP+RLT) were compared for solving box-constrained quadratic programs. The average optimality gap of RLT was reduced by an order of magnitude using SDP relaxations. A further order of magnitude reduction in the average optimality gap was observed for the SDP+RLT relaxations. However, a major drawback of utilizing SDP+RLT relaxations is the high computational effort; they required over 200 CPU

seconds on average, compared to 1 CPU second for the separate RLT and SDP relaxations. This experience motivates our approach of employing SDP-induced cuts to tighten RLT representations while maintaining manageably sized LP relaxations.

Konno et al. [40] proposed using a similar cut of the type $\tilde{\alpha}^T M_0 \tilde{\alpha} \geq 0$, where $\tilde{\alpha} \in \mathbb{R}^n$ is the normalized eigenvector corresponding to the smallest eigenvalue of the matrix $\bar{M}_0$ (which evaluates M_0 at the solution $(\bar{x}, \bar{X})$), given that $\bar{M}_0$ is not positive semidefinite. However, computing $\tilde{\alpha}$ can be relatively burdensome, and moreover, the round of cuts (2.2) generated for the augmented matrix M_1 tends to yield tighter relaxations.

As a precursor to this approach, Shor [80] recommended examining the minimum eigenvalue of M_0 as a function $\lambda_1(M_0)$, where $\lambda_1(M_0) \equiv$ minimum $\{\alpha^T M_0 \alpha : \|\alpha\| = 1\}$ by the Rayleigh-Ritz formula. Noting that $\lambda_1(M_0)$ is a concave, but nondifferentiable, function of M_0 that is characterized by the minimum of a family of linear functions, Shor [80] developed a nondifferentiable optimization approach for quadratic programs in which $\lambda_1(M_0) \geq 0$ is incorporated within the model formulation in lieu of $M_0 \succeq 0$.

Using a similar thrust, Vanderbei and Benson [87] proposed an alternative, smooth, convex, finite linear programming representation for the relationship $M_0 \succeq 0$, noting that $M_0 \succeq 0$ if and only if it can be factored as $M_0 = LDL^T$, where L is a lower triangular matrix having unit diagonal elements and D is a nonnegative diagonal matrix. Denoting $d_j(M_0), \forall j = 1, \ldots, n$, as the diagonal elements of D for a given $n \times n$ symmetric matrix $M_0 \succeq 0$, Vanderbei and Benson [87] showed that $d_j(M_0)$ is a concave, twice differentiable function on the set of positive semidefinite matrices. Accordingly, the authors replaced $M_0 \succeq 0$ with the nonlinear, smooth (but only implicitly defined) constraints $d_j(M_0) \geq 0, \forall j = 1, \ldots, n$, and developed a specialized interior point algorithm to solve the resulting problem.

As a further generalization of the RLT procedure, Lasserre [41, 43] discussed the generation of tight relaxations for polynomial programming problems using *linear matrix inequalities* (**LMIs**). Let $x_{(m)}$ represent the vector that further augments $x_{(1)}$ with all quadratic terms involving the x-variables, then all such cubic terms, and so on, up to all possible monomials of order m. Accordingly, define the *moment matrix*

$$M_m \equiv [x_{(m)} x_{(m)}^T]_L. \tag{2.3}$$

Then, for the case of an unconstrained polynomial programming problem of the type: Minimize

$\{\phi_0(x) : x \in \mathbb{R}^n\}$, where $\phi_0(x)$ is a polynomial of degree δ, Lasserre [41] considered the relaxation $\mathbf{R_m}$ given below, where $m \geq \lceil \delta/2 \rceil$, and where $\theta(x) \equiv a^2 - \|x\|^2$, with $a > 0$ being the radius of a ball that is known to contain an optimal solution (so that $\|x\|^2 \leq a^2$, i.e., $\theta(x) \geq 0$, can be validly imposed):

$$\mathbf{R_m}\text{: Minimize}\{[\phi_0(x)]_L : M_m \succeq 0, [\theta(x)x_{(m-1)}x_{(m-1)}^T]_L \succeq 0\}. \tag{2.4}$$

Lasserre [41] proved that the relaxation $\mathbf{R_m}$ is asymptotically exact, in that, as $m \to \infty$, the optimal value of $\mathbf{R_m}$ approaches that of the underlying polynomial program. In fact, for a *univariate* polynomial program, to minimize $\phi_0(x_1)$ subject to $l_1 \leq x_1 \leq u_1$, where $\phi_0(x_1)$ is of odd degree $2m+1$ (some additional manipulation is required for even degree problems), Lasserre [43] showed that the optimal value is recovered via the semidefinite program:

$$\text{Minimize } \{[\phi_0(x_1)]_L : [(x_1 - l_1)x_{(m)}x_{(m)}^T]_L \succeq 0, [(u_1 - x_1)x_{(m)}x_{(m)}^T]_L \succeq 0\}. \tag{2.5}$$

For multivariate constrained polynomial programming problems of the type: Minimize $\{\phi_0(x) : \phi_r(x) \geq \beta_r, \forall r = 1, \ldots, R\}$, having a degree δ, a similar relaxation to that in (2.4) was generated as follows, where $m \geq \lceil \delta/2 \rceil$:

$$\mathbf{CR_m}\text{: Min}\left\{[\phi_0(x)]_L : M_m \succeq 0, [(\phi_r(x) - \beta_r)x_{(m-\lceil \delta_r/2 \rceil)}x_{(m-\lceil \delta_r/2 \rceil)}^T]_L \succeq 0, \forall r = 1, \ldots, R\right\}, \tag{2.6}$$

where δ_r is the degree of $\phi_r(x), \forall r = 1, \ldots, R$. Under certain stringent conditions imposed on the polynomials defining the feasible region, Lasserre [41, 43] exhibited that the relaxation $\mathbf{CR_m}$ becomes asymptotically exact as $m \to \infty$.

In his seminal work, Shor [80] introduced the construction of quadratically constrained quadratic programming (QCQP) representations for univariate and multivariate polynomial programs. For the univariate case, upon appending all the related transformation-induced quadratic constraints to the QCQP representation, the dual quadratic bound was shown to be exact using the fact that nonnegative even degree polynomials can be decomposed into sum of squares polynomials. This theory was extended to multivariate polynomials for a special case where a sum of squares decomposition of the underlying polynomial functions exists. The main result stemming from this work is that the QCQP dual bound for multivariate polynomial programs is exact if and only if there exists a sum of squares decomposition of the defining polynomial functions.

Parrilo [58] investigated the equivalence of nonnegative polynomials and sum of squares poly-

nomials, and the computational tractability of the sum of squares decomposition process using SDP solvers. Given a polynomial of even degree $2d$, the sum of squares decomposition procedure boils down to finding a $Q \succeq 0$ such that

$$F(x) = z^T Q z, \text{ where } z \equiv [1, x_1, x_2, \ldots, x_n, x_1 x_1, x_1 x_2, \ldots, x_n^d].$$

By solving a semidefinite feasibility problem, it can be decided if there exists a sum of squares decomposition of a given polynomial function. The main contribution of this study is proposing a (relatively) computationally tractable procedure for the sum of squares decomposition process. Since every nonnegative polynomial cannot be decomposed into a sum of squares representation, the method can produce suboptimal solutions. For these cases, Parrilo [58] suggested using Artin's work ([8]) on Hilbert's 17th problem, albeit at a higher computational cost. Related computational results for this procedure were presented in Parrilo and Sturmfels [59]. In comparisons with algebraic methods based on Gröbner bases and homotopy algorithms, and using randomly generated test problems having the structure:

$$f(x_1, x_2, \ldots, x_n) = x_1^{2d} + x_2^{2d}, \ldots, x_n^{2d} + g(x_1, x_2, \ldots, x_n),$$

the effort was shown to be dramatically reduced when implementing semidefinite programming relaxations. Although the semidefinite programming relaxations only provide a lower bound in theory (and asymptotically achieve optimality as the order of the relaxation increases), the computational results demonstrated that global optimality was obtained for all test instances.

The computational tractability of utilizing the sum of squares decomposition procedure for solving polynomial programming problems worsens drastically as the size of the linear matrix inequalities increases, which grows rapidly as the number of variables or the degree of the polynomial program increases. In many practical applications, it is observed that the problems exhibit a sparse structure. By exploiting this inherent sparsity of the problem, the size of the generated SDP relaxations can be significantly reduced. In this research direction, Kojima et al. [39] explored the decomposition of sparse polynomials as a sum of squares representation in order to enhance computational efficiency. Given that a polynomial function has a feasible but unknown sum of squares representation in terms of r polynomials, the authors defined F and G^0 as a support for the original polynomial function and a common support for the union of the r polynomials, respectively. The main idea pursued was to find a minimal-sized $G^* \subseteq G^0$ for reducing the size of the linear matrix inequalities. Kojima et al. [39] employed a two phase method for this purpose, where the first

phase computes G^0, and where the second phase utilizes a digraph-based approach to obtain G^*. In this process, nodes having no outgoing edges are eliminated from the suitably defined graph (and also from G^0), and the algorithm terminates with G^*. The effectiveness of the algorithm in reducing the size of the common support was computationally demonstrated using some specific examples having up to 15 variables.

Along the same research direction, Waki et al. [89] introduced alternative techniques for performing sum of squares decomposition and formulating semidefinite programming relaxations for polynomial optimization problems having structured sparsity properties. The authors defined a correlative sparsity pattern (csp) matrix R, where a nonzero value is assigned to R_{ij} if the variables x_i and x_j simultaneously appear in a term in the objective function or constraints. The method accordingly constructs a csp graph whereby maximal cliques form the supports for the required sum of squares polynomials. Given that finding all maximal cliques is a difficult problem, Waki et al. [89] generated a chordal extension of the original csp graph for which maximal cliques can be determined in polynomial-time. Upon finding such maximal cliques and associated supports for the sum of squares polynomials, the second phase of the method introduced in Kojima et al. [39] for computing G^* was likewise utilized to reduce the size of the supports. Waki et al. [89] indicated that for quadratic programming problems, sparse and dense representations provide the same quality of lower bounds. Although this statement does not extend to higher order polynomials, observed differences between these two bounds were not significant in practice. Extensive computational results using unconstrained and constrained polynomial programming test problems were provided to demonstrate that the proposed sparse sum of squares decomposition technique drastically decreases the required computational effort in comparison with using dense SDP relaxations.

Further improvements on the sparse SDP relaxations of Kojima et al. [39] and Waki et al. [89] were proposed in Lasserre [44], along with a proof of convergence to a global optimum for polynomial programming problems, based on two modifications to the methodology presented by Waki et al. [89]. First, the sparse SDP relaxations were enhanced by appending p quadratic constraints, where p is the number of support sets. Second, noting that the convergence result of Lasserre [44] requires the Running Intersection Property for maximal cliques, which is satisfied if the intersection of the $(k+1)$th clique with the union of the first k cliques is contained in one of the previous cliques, it was shown that this property can be satisfied via a reordering of the cliques or by enlarging certain cliques. Given that the Running Intersection Property holds for the generated

sets of maximal cliques, the sparse SDP relaxations enhanced with the p quadratic constraints were proven to converge to a global optimum.

Next, we discuss three software packages that are specialized for solving polynomial programming problems via SDP relaxations, all of which convert the original problem into an SDP problem (through a sum of squares representation technique for SOSTOOLS and SparsePOP, and via a generalized problem of moments for GloptiPoly). These software packages call certain SDP solvers (SeDuMi or SDPT3) to retrieve a solution to the formulated SDP, and then convert this solution back to that for the original problem. The main difference between the efficiency of these software tools depends on the methodology used for handling the problem structure of the original program.

SOSTOOLS (Prajna et al. [61]) was developed based on the theory introduced in Parrilo [58]. SOSTOOLS handles the sparsity in the problem structure via the concept of a Newton polytope, which is defined as the convex hull of the exponents in the given polynomial program. In particular, SOSTOOLS recognizes multipartite structures, if this exists within the problem.

GloptiPoly (Henrion and Lasserre [30] and Henrion et al. [31]) solves a hierarchy of relaxations that produces a series of lower bounds converging to the global optimum value for polynomial programming problems having inequality, equality, and integrality constraints based on the methodology introduced in Lasserre [41-43]. For sparse problem structures, the coefficient matrices are defined using sparse representations within Matlab, thus enabling substantial memory savings. However, this does not affect the size of the underlying linear matrix inequalities. Although SeDuMi solves Shor's ([80]) relaxation for max-cut problems with hundreds of variables, GloptiPoly cannot handle quadratic programming problems having more than 19 variables.

SparsePOP (Waki et al. [90]) differs from SOSTOOLS and GloptiPoly in that it exploits the sparse structure of the problem based on the theory developed in Kojima et al. [39] and Waki et al. [89]. SparsePOP can handle problems having up to 1000 variables (in comparison to 20 variables with GloptiPoly) if the polynomial programming problem structure satisfies certain correlative sparsity patterns. Although the quality of lower bounds is not necessarily as good as that obtained using dense SDP relaxations in theory, the computational results demonstrated that the desired accuracy was yet obtained for many test instances.

Several pros and cons of using semidefinite programming versus LP relaxations for solving polynomial programs are discussed in Lasserre [43], where it is surmised that although semidef-

inite representations are more theoretically appealing, LP relaxation-based approaches have more practical significance due to the availability of effective LP software capable of robustly handling large-scale problem instances. Further discussions on semidefinite programming relaxations versus LP relaxations are provided in Laurent and Rendl [45].

The second class of valid inequalities proposed in this dissertation is motivated by the *squared grid-factor* (SGF) constraints introduced in Sherali and Tuncbilek [74] for the more simple class of box-constrained univariate polynomial programming problems, where the grid-points were located uniformly within the range of the single variable $x \in [l, u]$. Specifically, the SGF constraints were derived by taking the products of the (squared) grid-factors $\lfloor \delta/2 \rfloor$ at a time, where the resultant constraint expression was further multiplied individually by the bound-factors $(x-l)$ and $(u-x)$ in case δ is odd. When compared to various convex variable-bounding and squared Lagrangian interpolation constraints for univariate polynomial programs, the SGF constraints produced tighter lower bounds with reasonable computational effort in a root node analysis.

In Chapters 4 and 5 of this dissertation, we significantly extend the foregoing ideas pertaining to the generation of valid inequalities, and propose new classes of semidefinite cuts and bound-grid-factor constraints in concert with branch-and-bound strategies for solving multivariate polynomial programming problems to global optimality.

2.3 Reduced RLT Representations for Polynomial Programming Problems

The RLT methodology generates a hierarchy of increasingly tighter relaxations, which is, however, accompanied with an increase in problem size. Accordingly, the RLT algorithm can be potentially enhanced by suitably imposing only a subset of certain identified key RLT constraints. In order to accelerate the RLT solution approach for polynomial programs, Sherali and Tuncbilek [74] proposed constraint filtering techniques that select only a subset of the RLT bound-factor restrictions to incorporate within the model formulation, based on the signs of the coefficients of the original monomials in the objective function or constraints. For example, in an inequality constraint (1.1b), if α_{rt} is positive (negative), then the associated RLT variable $X_{J_{rt}}$ that is used to linearize the monomial $\prod_{j \in J_{rt}} x_j$ might have a tendency to be larger (smaller) than $\prod_{j \in J_{rt}} x_j$ in the RLT-based

LP relaxation, and so, restrictions that provide upper (lower) bounds on $X_{J_{rt}}$ are more likely to be relevant in tightening the relaxation. Rules of this type for retaining only such bound-factor product constraints along with certain additional restrictions that provide significant support for the monomials involved in the retained constraints were developed by Sherali and Tuncbilek [74]. However, for the sake of theoretical convergence, the algorithm must ultimately consider the entire set of bound-factor constraints.

Another RLT constraint reduction strategy was explored by Sherali et al. [77] for first-level RLT representations (RLT-1) of 0-1 linear mixed-integer programs. Employing optimal dual multipliers to the LP relaxation $\overline{\text{RLT-1}}$ of RLT-1, it was shown that there exists an *equivalent* representation using only one of each identified pair of RLT restrictions within RLT-1, which provides the same lower bound as that obtained via $\overline{\text{RLT-1}}$. Promising computational results were presented for a proposed constraint selection strategy that attempts to *a priori* predict such an existing equivalent reduced size RLT representation.

For quadratic or bilinear programs having an embedded linear equality system of the form $Ax = b$, where A is $m \times n$ of rank m, a different technique was developed by Liberti [46] to reduce the number of bilinear terms in the RLT formulation while maintaining an equivalent representation. Specifically, consider the RLT constraints obtained by multiplying the linear equality restrictions with the variable $x_k,\ \forall k \in \mathcal{N}$, to yield $AX_{(.k)} - bx_k = 0,\ \forall k \in \mathcal{N}$, where the RLT variable vector $X_{(.k)} \equiv (X_{jk}, \forall j \in \mathcal{N})$ represents the product terms $(x_j x_k, \forall j \in \mathcal{N})$. Accordingly, examining a so-called *companion system* $Az^k = 0,\ \forall k \in \mathcal{N}$, where $z^k_j \equiv X_{jk} - x_j x_k, \forall j \in \mathcal{N}$, and where this system has a rank between $\frac{m(m+1)}{2}$ and $(n-m)m$, it was shown that such a companion system with rank r implies r of the defining RLT substitution identities, which can therefore be dropped from the formulation without weakening it. Based on this result, basic feasible solutions were explored to the companion system of size $mn \times n^2$ to determine which bilinear terms to explicitly retain in the formulation, where the latter correspond to the nonbasic variables for this expanded system. Liberti [46] also discussed the efficiency and applicability of the foregoing method for solving sparse problems in which the number of X-variables might be considerably larger than the rank of the associated companion system. To handle such cases more effectively, Liberti and Pantelides [48] introduced certain graph theoretical algorithms to select valid sets of RLT constraints that contain a reduced number of X-variables. For quadratic programming problems, an algorithm was proposed by Liberti [47] to construct a basis for the companion system by selecting a set of variables that maximizes the convexity gap, where for each RLT variable, the convexity gap was

formulated as the area (or volume) between the corresponding RLT substitution constraint and the associated bound-factor product constraints. Cafieri et al. [15] further extended the concepts developed by Liberti [46, 47] to address polynomial programming programs, wherein each linear equality restriction was multiplied with monomials up to degree $\delta - 1$ in order to generate a similar companion system of size $m[\binom{n+\delta-1}{\delta-1} - 1] \times [\binom{n+\delta}{\delta} - (n+1)]$, and then a basis of this companion system was utilized to derive a reduced RLT representation. A technique for deriving a convexity gap estimate for each candidate basis was also proposed, where the recommended procedure then selected a basis having the maximum such value. Using randomly generated continuous quadratic knapsack problems having up to 10 variables, it was demonstrated that the computational effort decreased substantially when using the reduced RLT approach in lieu of the full RLT representation in six out of the nine test cases.

Chapter 3

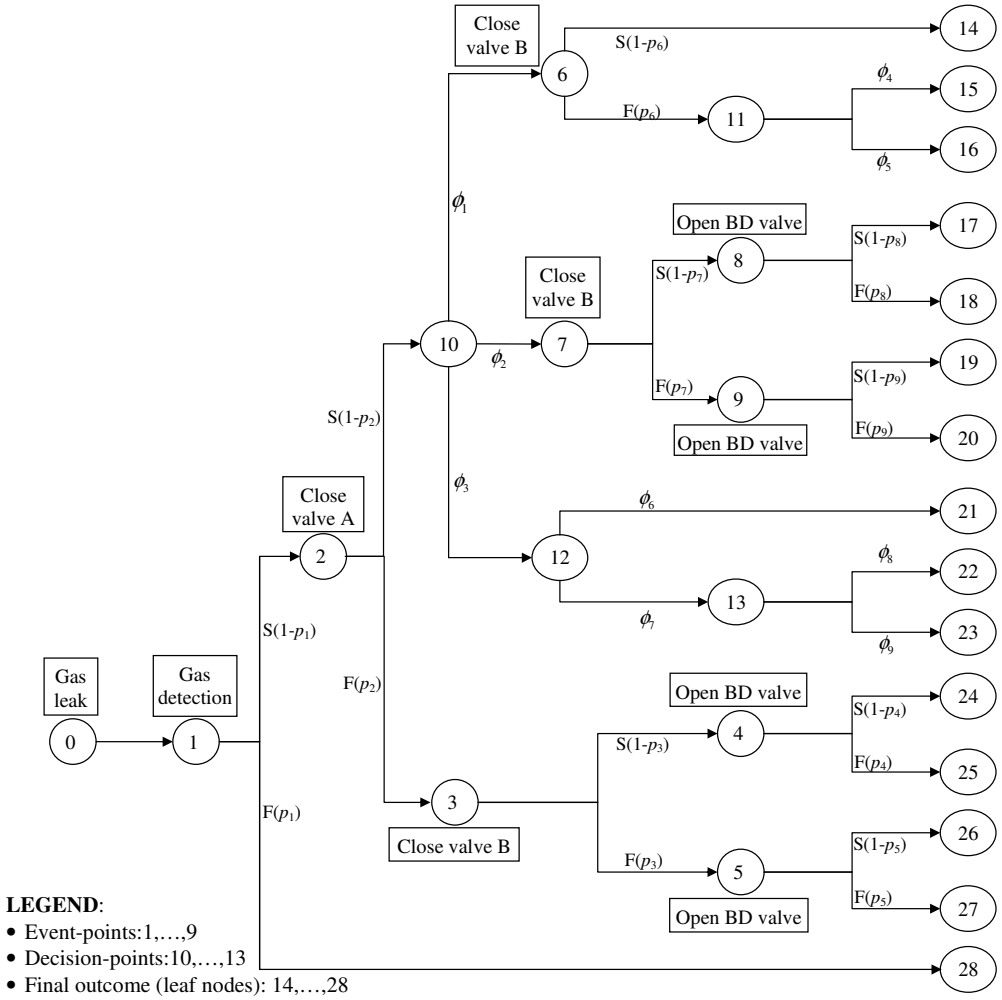
Selecting Optimal Alternatives and Risk Reduction Strategies in Decision Trees

In this chapter, we conduct a quantitative analysis for a strategic risk management problem that involves allocating certain available failure-mitigating and consequence-alleviating resources to reduce the failure probabilities of system safety components and subsequent losses, respectively, together with selecting optimal strategic decision alternatives, in order to minimize the risk or expected loss in the event of a hazardous occurrence. Using a novel decision tree optimization approach to represent the cascading sequences of probabilistic events as controlled by key decisions and investment alternatives, the problem is modeled as a nonconvex mixed-integer 0-1 factorable program. We develop a specialized branch-and-bound algorithm in which lower bounds are computed via tight linear relaxations of the original problem that are constructed by utilizing a polyhedral outer-approximation mechanism in concert with two alternative linearization schemes having different levels of tightness and complexity. We also suggest three alternative branching schemes, each of which is proven to guarantee convergence to a global optimum for the underlying problem. Extensive computational results and sensitivity analyses are presented to provide insights and to demonstrate the efficacy of the proposed algorithm.

3.1 Introduction

This chapter addresses the strategic reduction of risk in a system that is characterized by a *decision tree*. Such a tree contains two types of nodes that represent either *event-points* or *decision-points*. At an event-point, some safety feature or measure is invoked that might lead to one of several outcomes, each represented by an arc having a specified probability of occurrence. For example, in the particular context of *Bernoulli events*, an event-point i would trigger either a failure or a success outcome state with respective probabilities p_i and $1 - p_i$ corresponding to the particular safety feature applied at this stage. On the other hand, the arcs emanating from each decision-point node are deterministic, and represent a selection among different available alternatives. These choices might involve making certain strategic decisions or system design selections at the particular stage in the process, which then govern the subsequent sequence of events and decisions. The decision tree is rooted at a *root node* or *node zero* that represents some component failure in the system under study or an external hazardous occurrence, which triggers a cascading sequence of strategic decisions and event outcomes based on applied actions or safety features that are invoked to control the damage. Hence, each branch in the tree represents a specific sequence of decision choices and event outcomes starting from the root node, and culminating in a *final outcome* or *leaf node* of the tree, where the latter entails an associated *consequence* or *loss*.

Figure 3.1 displays a particular example of a decision tree pertaining to the rupture of a gas-line in an offshore oil and gas platform, which has been adapted from a simpler *event tree* representation (i.e., one having only event-point nodes) as described in Andrews and Dunnett [3], and that involves Bernoulli events. Here, each event-point, represented by the nodes indexed $1, \ldots, 9$, corresponds to applying some safety measure such as closing an isolation valve to localize the damage or opening a blow-down (BD) valve to depressurize certain critical sections of the system, and can lead to one of two immediate scenarios or outcomes depending on the success or failure of this measure. Note that a particular safety measure can be activated at different event-points in the tree to counteract the hazard at that particular stage in the process. At any decision-point in the tree (nodes $10, \ldots, 13$ in Figure 3.1), depending on the situation at that particular stage as governed by the sequence of events and decisions leading up to it, one of several strategic alternative options can be selected, where each such option is represented by a binary variable that takes on a value of one if this option is selected and zero otherwise. Hence, at the decision node 10 for example, one of three choices can be made with respect to subsequently activating only the closure of valve B, or



LEGEND:

- Event-points:1,…,9
- Decision-points:10,…,13
- Final outcome (leaf nodes): 14,…,28
- Binary variables:

ϕ_1 : Activate only valve B
ϕ_2 : Activate valves B and BD
ϕ_3 : Activate neither of valves B and BD
$\Big\}\ \phi_1 + \phi_2 + \phi_3 = 1$

$\left.\begin{matrix}\phi_4: \\ \phi_5:\end{matrix}\right\}$ Emergency response choices, with $\phi_4 + \phi_5 = \phi_1$

$\left.\begin{matrix}\phi_6: \\ \phi_7:\end{matrix}\right\}$ Primary cleaning options, with $\phi_6 + \phi_7 = \phi_3$

$\left.\begin{matrix}\phi_8: \\ \phi_9:\end{matrix}\right\}$ Secondary cleaning options, with $\phi_8 + \phi_9 = \phi_3\phi_7$

Figure 3.1: A decision tree representing a gas-line rupture.

the closure of valve B and the opening of the blow-down (BD) valve, or neither of these, which are respectively represented by the binary variables ϕ_1, ϕ_2, and ϕ_3 with $\phi_1 + \phi_2 + \phi_3 = 1$. The binary variables designating the alternative choices at the other decision nodes 11, 12, and 13 are specified in the legend of Figure 3.1 along with their constraining relationships. For example, assume that the first option is chosen at decision node 10. At the event-point represented by node 6, the valve B either fails or succeeds to operate with respective probabilities p_6 and $1 - p_6$. Following the success outcome, the leaf node 14 is reached, while the failure event leads to another decision node at which one of two emergency response choices can be made, which are represented by the binary variables ϕ_4 and ϕ_5, respectively, with $\phi_4 + \phi_5 = \phi_1$. Note that when $\phi_1 = 0$, the choice represented at decision node 11 does not arise. In this fashion, the cascading sequences of events and decisions lead to final leaf nodes 14, ..., 28, each entailing a particular loss or consequence.

The main contributions of this chapter are as follows. First, we introduce a novel *decision tree optimization* (DTO) model that significantly extends the event tree model analyzed in Sherali et al. [78] by accommodating decision nodes having associated multiple options to select from, which enables a strategic restructuring of the tree-based system by way of judiciously composing the operational components of the system. Concurrently, the DTO model considers the allocation of certain available resources within the resulting tree to mitigate failure probabilities and consequence losses in order to minimize the overall risk. Second, we perform a polyhedral analysis for this more complex model to enhance its solvability by (a) analyzing alternative methods for effectively linearizing the monomials of binary variables within the formulation; (b) exploring suitable polyhedral outer-approximations for the region defined by the logarithmic functions in the model; (c) deriving classes of optimality-induced valid inequalities, and (d) designing range reduction strategies for tightening the model representation. Third, we develop a specialized branch-and-bound algorithm that adopts effective alternative partitioning strategies, and we establish its convergence to a global optimum based on several derived theoretical properties of the proposed formulation and its relaxation. Fourth, we present extensive computational results (including insightful sensitivity analyses) using the proposed branch-and-bound algorithm, along with comparisons against the commercial global optimization software BARON, Version 8.1.5 [84]. In particular, we demonstrate a significant improvement in algorithmic performance using the proposed approach.

The remainder of this chapter is organized as follows. In Section 3.2, we formulate a mathematical model for the DTO problem and then develop a suitable reformulation along with a tight lower-bounding representation through some transformations, polyhedral outer-approximations,

and valid inequalities in concert with two alternative linearization schemes. In Section 3.3, we derive a specialized branch-and-bound procedure to solve the DTO problem and establish its convergence. Sensitivity analyses and computational results are presented in Section 3.4 for a hypothetical case study, along with algorithmic performance results and comparisons using a set of simulated problems.

3.2 Model Formulation: Analysis and Enhancements

To model the proposed decision tree optimization problem, we begin by defining the risk associated with each possible final outcome or leaf node of the tree. Toward this end, consider the following notation:

- $I_e \equiv \{1,\ldots,I\}$: set of indices representing the event-point nodes.
- p_i and $(1-p_i)$: respectively, the conditional probabilities of failure and success associated with the outcomes resulting from the application of the particular safety feature at event-point i, given that the sequence of decisions and events on the chain from the root node to node i has occurred, $i \in I_e$.

Remark 3.1. Note that for the sake of notational simplicity and clarity in exposition, we consider Bernoulli events in our model, although our analysis readily generalizes to multi-state events. For example, we could consider multiple levels of failure indexed by $r \in F_i$, having respective probabilities $p_{ir}, r \in F_i, \forall i \in I_e$, with the success probability given by $p_i^s \equiv (1-\sum_{r\in F_i} p_{ir}) \in (0,1)$. Some comments on accommodating this feature in the model are provided in our discussion. □

- $\tau = \{j$: node j is a final outcome or leaf node index of the tree$\}$.
- $d = 1,\ldots,D$: indices for the collective set of decisions made over the entire tree.
- $\phi_d = \begin{cases} 1 & \text{if decision } d \text{ is selected} \\ 0 & \text{otherwise.} \end{cases}$.
- K: index set of decision-point nodes in the tree.

- $J_k = \{$decisions d (with corresponding binary variables ϕ_d) associated with the alternative option-based arcs that emanate from node $k\}$, $\forall k \in K$.
- $B_k = \{$ordered set of decisions d (with corresponding binary variables ϕ_d) that occur on the (unique) path from the root node 0 to node $k\}$, $\forall k \in K \cup I_e \cup \tau$.

For each $j \in \tau$:

- $C_j = \{i$: node i lies on the (unique) path from the root node 0 to node j, excluding nodes 0 and $j\}$.
- $A_{ij} =$ arc emanating from node $i \in C_j$ that lies on the path joining the root node 0 to node $j \in \tau$.
- $S_{1j} = \{i \in C_j \cap I_e$: the (conditional) probability associated with arc A_{ij} is $p_i\}$, $\forall j \in \tau$.
- $S_{2j} = \{i \in C_j \cap I_e$: the (conditional) probability associated with arc A_{ij} is $(1-p_i)\}$, $\forall j \in \tau$.
- $S_{3j} = \{d$: decision d is associated with arc $A_{ij}, \forall i \in C_j \cap K\}$, $\forall j \in \tau$.
- $l_j =$ loss or consequence (in dollars, say) associated with leaf node $j \in \tau$.
- $c_d =$ fixed cost associated with selecting decision d, $\forall d = 1, \ldots, D$. (Note that for the sake of simplicity, we consider here only linear cost terms; however, the methodology described below readily extends to the case where we might have polynomial cost terms that are predicated on products of the ϕ-variables that occur along (partial) paths from node 0 to any leaf node.)

Then the *risk* associated with the leaf node $j \in \tau$ is the *expected consequence/loss* that is conditioned on the binary decisions as given by:

$$\psi_j \equiv l_j \prod_{i \in S_{1j}} p_i \prod_{i \in S_{2j}} (1-p_i) \prod_{d \in S_{3j}} \phi_d, \ \forall j \in \tau, \tag{3.1}$$

and the *overall risk* is the total expected consequence given by $\sum_{j \in \tau} \psi_j$.

Thus far, the only decisions identified with respect to the decision tree analysis involve selecting binary values for the variables $\phi_d,\ d = 1,\ldots,D$, subject to the restrictions that

$$\sum_{d\in J_k} \phi_d = \prod_{d\in B_k} \phi_d,\ \forall k \in K, \tag{3.2}$$

where $\prod_{d\in B_k} \phi_d \equiv 1$ whenever $B_k = \varnothing,\ \forall k \in K$. Now, as in Sherali et al. [78], suppose that we can additionally bring to bear certain preventive or protective *event-related resources* such as investments in improved technologies or supporting equipment for the various safety features involved so as to mitigate the conditional failure probabilities, $p_i, i \in I_e$, given the sequence of preceding actions and events. Likewise, suppose that we can invest in certain available *consequence-related resources* such as clean-up mechanisms and trained emergency response personnel in order to ameliorate the potential consequences or losses l_j associated with the leaf nodes $j \in \tau$. Accordingly, the goal would be to determine how to effectively deploy the available limited resources under some budgetary restrictions so as to appropriately manipulate the failure probabilities and the resultant consequences, as well as devise a plan for making suitable strategic decision option choices, in order to minimize the overall risk. More specifically, consider the following related notation and concepts:

- $m = 1,\ldots,M$: index set for the available event-related resources.
- s_m = total available units of the event-related resource $m,\ \forall m = 1,\ldots,M$.
- q_{im} = *decision variable* representing the quantity of event-related resource m that is allocated to reduce the associated (conditional) failure probability p_i of the safety feature deployed at event-point or node $i,\ \forall i \in I_e,\ m = 1,\ldots,M$.
- c_{im} = cost (in dollars) per unit of q_{im}.
- $n = 1,\ldots,N$: index set for the available consequence-related resources.
- t_n = total available units of the consequence-related resource $n,\ \forall n = 1,\ldots,N$.
- r_{jn} = *decision variable* representing the quantity of consequence-related resource n that is allocated to reduce the loss magnitude l_j associated with the leaf node $j,\ \forall j \in \tau,\ n = 1,\ldots,N$.
- d_{jn} = cost (in dollars) per unit of r_{jn}.

- β = total available budget (in dollars).

- *Logit model for relating* p_i *to* $\{q_{im},\ m = 1,\ldots,M\},\ \forall i \in I_e$*:*

$$ln\left[\frac{p_i}{1-p_i}\right] = a_{i0} - \sum_{m=1}^{M} a_{im}q_{im},\ \forall i \in I_e,\ \text{where } (a_{i0},...,a_{iM}) \geq 0. \tag{3.3}$$

- $[p_i^l, p_i^u]$: lower and upper bounds on p_i, where $0 < p_i^l \leq p_i \leq p_i^u < 1,\ \forall i \in I_e$, and where these bounds are either imposed or are implied by (3.3) and the available resources; in particular, we assume that $p_i^u = p_i^{u(3.3)} \equiv e^{a_{i0}}/(1+e^{a_{i0}}), \forall i \in I_e$, as implied by (3.3), i.e., the model seeks further possible reductions in p_i below p_i^u by using suitable event-related resource allocations as necessary.

- *Model for relating* l_j *to* $\{r_{jn}, n = 1,...,N\},\ \forall j \in \tau$ *:*

$$l_j = b_{j0} - \sum_{n=1}^{N} b_{jn}r_{jn},\ \forall j \in \tau, \text{where } (b_{j0},...,b_{jN}) \geq 0. \tag{3.4}$$

Remark 3.2. As an alternative to (3.4), we could consider a diminishing marginal return loss function of the type $l_j = b_{j0}\,exp\{-\sum_{n=1}^{N} b_{jn}r_{jn}\},\ \forall j \in \tau$, having nonnegative parameter values. Our proposed algorithm can be identically applied for such loss functions upon taking logarithms. Likewise, for the general multi-failure state scenario discussed in Remark 3.1, the corresponding logit model would take the form $ln[p_{ir}/p_i^s] = a_{ir0} - \sum_{m=1}^{M} a_{irm}q_{im}, \forall r \in F_i, i \in I_e$, with the analysis described below following identically. □

- $[l_j^l, l_j^u]$: lower and upper bounds on l_j, where $0 < l_j^l \leq l_j \leq l_j^u < \infty,\ \forall j \in \tau$, and where these bounds are either imposed or are implied by (3.4) and the available resources; in particular, we assume that $l_j^u = l_j^{u(3.4)} \equiv b_{j0}, \forall j \in \tau$, as implied by (3.4), i.e., the model seeks further possible reductions in l_j below l_j^u by using suitable consequence-related resource allocations as necessary.

The *decision tree optimization* problem, **DTO**, can then be formulated as the following 0-1 mixed-integer nonlinear programming problem:

$$\textbf{DTO:}\quad \text{Minimize}\quad \sum_{d=1}^{D} c_d\phi_d + \sum_{j\in\tau} l_j \prod_{i\in S_{1j}} p_i \prod_{i\in S_{2j}} (1-p_i) \prod_{d\in S_{3j}} \phi_d \tag{3.5a}$$

subject to

$$\sum_{i\in I_e} q_{im} \le s_m,\ \forall m = 1,\ldots,M \tag{3.5b}$$

$$\sum_{j\in\tau} r_{jn} \le t_n,\ \forall n = 1,\ldots,N \tag{3.5c}$$

$$\sum_{i\in I_e}\sum_{m=1}^{M} c_{im}q_{im} + \sum_{j\in\tau}\sum_{n=1}^{N} d_{jn}r_{jn} \le \beta, \tag{3.5d}$$

$$ln\left[\frac{p_i}{1-p_i}\right] = a_{i0} - \sum_{m=1}^{M} a_{im}q_{im},\ \forall i \in I_e \tag{3.5e}$$

$$l_j = b_{0j} - \sum_{n=1}^{N} b_{jn}r_{jn},\ \forall j \in \tau \tag{3.5f}$$

$$\sum_{d\in J_k} \phi_d = \prod_{d\in B_k} \phi_d,\ \forall k \in K \tag{3.5g}$$

$$(p,l) \in \Omega \equiv \left\{ (p,l):\ \begin{array}{l} 0 < p_i^l \le p_i \le p_i^u < 1,\ \forall i \in I_e \\ 0 < l_j^l \le l_j \le l_j^u < \infty,\ \forall j \in \tau \end{array} \right\}, \tag{3.5h}$$

$$(q,r) \in \mathscr{P} \tag{3.5i}$$

$$\phi_d \text{ binary},\ \forall d = 1,\ldots,D. \tag{3.5j}$$

In this formulation, the objective function (3.5a) seeks to minimize the total cost of implementing decisions plus the consequent expected loss or overall risk. (Note that the first term in the objective function is equivalent to the polynomial term $\sum_{k\in K}\sum_{d\in J_k} c_d\phi_d\big[\prod_{d'\in B_k}\phi_{d'}\big]$ because of Constraints (3.5g) and (3.5j)). Whereas we adopt the objective function (3.5a) in this paper, we could alternatively consider a mean-variance model following Markowitz [50, 51] by accommodating a variance term within (3.5a), where the variance of loss is given by:

$$\text{Var(loss)} = \sum_{j\in\tau} l_j^2 \prod_{i\in S_{1j}} p_i \prod_{i\in S_{2j}} (1-p_i) \prod_{d\in S_{3j}} \phi_d - \left(\sum_{j\in\tau} l_j \prod_{i\in S_{1j}} p_i \prod_{i\in S_{2j}} (1-p_i) \prod_{d\in S_{3j}} \phi_d\right)^2. \tag{3.6}$$

Although this complicates the model by doubling the degree of the polynomial objective function, the reformulation mechanism and polyhedral analysis presented below can readily handle the variance term (3.6) by introducing $|\tau|+2$ new variables along with suitable linearization constructs. However, we shall focus on (3.5a) in this study and we propose using the mean-variance risk function as a future extension. Constraints (3.5b) and (3.5c) impose the resource availability restrictions; Constraint (3.5d) enforces the budgetary limitation; Constraints (3.5e), (3.5f), and

(3.5g) follow from (3.3), (3.4), and (3.2), respectively; (3.5h) requires the (p,l)-variables to satisfy the specified bounding restrictions; and $\mathscr{P}$ in (3.5i) is a polyhedron (embedded in the nonnegative orthant) that includes any additional suitable constraints and variables in order to further restrict the (q,r)-variables, or to relate them to other influencing technological or operational decisions that are not explicitly stated in the above model. In the simplest case, $\mathscr{P} \equiv \{(q,r) : (q,r) \geq 0\}$. Observe that the nonconvexity in Problem DTO arises due to the polynomial function in (3.5a) and (3.5g), the logarithmic (factorable) term in (3.5e), as well as binary decision variables in (3.5j).

Remark 3.3. There is another related model that can be formulated to study a sensitivity analysis issue in the context of decision trees, which is also of interest and can be handled by the algorithmic process discussed below. Consider a decision tree in which the event probabilities and loss values are not known with certainty, but might vary within intervals as designated in (3.5h) (while not being strategically controllable). Given such variabilities in the probabilities and the consequences, we might be interested in ascertaining the least expected consequence or risk value, along with the maximum possible deviation from this value that could occur due to the inherent uncertainties in the problem. Hence, we could first minimize (3.5a) subject to (3.5g), (3.5h), and (3.5j), and then fixing the resulting ϕ-variable values, we could next determine the maximum of (3.5a) subject to (3.5h) in order to resolve these two respective issues. Furthermore, as an alternative approach for making decisions that hedge against this uncertainty, we could formulate and solve a corresponding robust optimization problem in the spirit of Mulvey et al. [55], or minimize the maximum risk by solving $\min\limits_{\phi:(3.5g),(3.5j)} \max\limits_{(p,l)\in\Omega} \{(3.5a)\}$. $\square$

We next define the following auxiliary variables, along with their implied bounds, in order to conveniently reformulate Problem DTO. To begin with, we transform the objective function by denoting

$$\theta_j \equiv l_j \prod_{i\in S_{1j}} p_i \prod_{i\in S_{2j}} (1-p_i),\ \forall j \in \tau. \tag{3.7a}$$

Thus we have,

$$\theta_j^l \leq \theta_j \leq \theta_j^u, \text{ where } \left\{ \begin{array}{l} \theta_j^l = l_j^l \prod\limits_{i\in S_{1j}} p_i^l \prod\limits_{i\in S_{2j}} (1-p_i^u) \\ \theta_j^u = l_j^u \prod\limits_{i\in S_{1j}} p_i^u \prod\limits_{i\in S_{2j}} (1-p_i^l) \end{array} \right\},\ \forall j \in \tau. \tag{3.7b}$$

Furthermore, noting (3.1), (3.5a), and (3.7a), we denote

$$\psi_j \equiv \theta_j w_{S_{3j}}, \forall j \in \tau, \tag{3.7c}$$

where

$$w_{S_{3j}} \equiv \prod_{d \in S_{3j}} \phi_d, \forall j \in \tau. \tag{3.7d}$$

Noting that the binariness of the ϕ_d-variables implies that of the $w_{S_{3j}}$-variables, $\forall j \in \tau$, we can linearize (3.7c) by replacing it with the restrictions:

$$\theta_j^l w_{S_{3j}} \leq \psi_j \leq \theta_j^u w_{S_{3j}} \text{ and } \theta_j^l (1 - w_{S_{3j}}) \leq \theta_j - \psi_j \leq \theta_j^u (1 - w_{S_{3j}}), \tag{3.7e}$$

by which it is readily verified that (3.7c) holds true whenever $w_{S_{3j}}$ takes on binary values, $\forall j \in \tau$. In fact, because of the positive (unit) objective coefficients on ψ_j, this relationship will hold true even if we retain just the inequalities $\psi_j \geq \theta_j^l w_{S_{3j}}$ and $\psi_j \geq \theta_j - \theta_j^u (1 - w_{S_{3j}})$ from (3.7e), $\forall j \in \tau$.

In the same spirit as $w_{S_{3j}}, j \in \tau$, we define a new variable w_J to represent $\prod_{d \in J} \phi_d$ for some specific sets $J \subset \{1, \ldots, D\}$ as described below, which will be used in the sequel for linearizing (3.5g). Here, whenever $|J| = 1$ with $J = \{d\}$, we will take the corresponding $w_J \equiv \phi_d$ itself, and whenever $J = \varnothing$, we will take $w_J \equiv 1$. Now, consider any set S_{3j} for $j \in \tau$ (in general, $w_{S_{3j}}$ might be replicated for different $j \in \tau$), and assume that the indices in S_{3j} are arranged according to the order in which the corresponding decisions d (with associated variables ϕ_d) occur along the path from the root node to node j. If $|S_{3j}| \geq 2$, this will generate w_J-variables via sets J defined by taking the first two indices from S_{3j}, the first three indices from S_{3j}, and so on, up to all the indices from S_{3j} (finally yielding $w_{S_{3j}}$ as denoted above). Letting $\mathscr{J}$ denote the resulting distinct sets J generated, we define the following identities:

$$w_J = \prod_{d \in J} \phi_d, \forall J \in \mathscr{J}. \tag{3.8}$$

Note that (3.8) subsumes (3.7d) since $S_{3j} \in \mathscr{J}$, $\forall j \in \tau$.

Next, to linearize (3.5e), we introduce the variables

$$y_{1i} = ln(p_i) \text{ and } y_{2i} = ln(1 - p_i), \ \forall i \in I_e. \tag{3.9a}$$

Note that,

$$y_{1i}^l \le y_{1i} \le y_{1i}^u \text{ and } y_{2i}^l \le y_{2i} \le y_{2i}^u,$$

$$\text{where } y_{1i}^l = ln(p_i^l),\ y_{1i}^u = ln(p_i^u),\ y_{2i}^l = ln(1-p_i^u),\ \text{and } y_{2i}^u = ln(1-p_i^l), \forall i \in I_e. \tag{3.9b}$$

Similarly, to linearize (3.7a) itself, we denote

$$z_j \equiv ln(\theta_j),\ \forall j \in \tau, \tag{3.10a}$$

where, based on (3.7b), we impose

$$z_j^l \le z_j \le z_j^u,\ \text{with } z_j^l = ln(\theta_j^l) \text{ and } z_j^u = ln(\theta_j^u),\ \forall j \in \tau. \tag{3.10b}$$

Likewise, to accommodate the term $ln(l_j)$ generated by taking logarithms in (3.7a), define

$$\xi_j \equiv ln(l_j), \forall j \in \tau, \tag{3.11a}$$

and impose the related bounds:

$$\xi_j^l \le \xi_j \le \xi_j^u,\ \text{where } \xi_j^l = ln(l_j^l) \text{ and } \xi_j^u = ln(l_j^u),\ \forall j \in \tau. \tag{3.11b}$$

Using the foregoing transformations and substitutions, we obtain the following equivalently reformulated problem DTO(Ω), which is predicated on the hyperrectangle Ω.

DTO(Ω): Minimize $\displaystyle\sum_{d=1}^{D} c_d \phi_d + \sum_{j \in \tau} \psi_j$ (3.12a)

subject to

$$\psi_j \ge \theta_j^l w_{S_{3j}} \text{ and } \psi_j \ge \theta_j - \theta_j^u (1 - w_{S_{3j}}),\ \forall j \in \tau \tag{3.12b}$$

$$w_J = \prod_{d \in J} \phi_d, \forall J \in \mathscr{J} \tag{3.12c}$$

$$\sum_{i \in I_e} q_{im} \le s_m, \forall m = 1, \ldots, M \tag{3.12d}$$

$$\sum_{j \in \tau} r_{jn} \le t_n, \forall n = 1, \ldots, N \tag{3.12e}$$

$$\sum_{i \in I_e} \sum_{m=1}^{M} c_{im} q_{im} + \sum_{j \in \tau} \sum_{n=1}^{N} d_{jn} r_{jn} \le \beta, \tag{3.12f}$$

$$y_{1i} - y_{2i} = a_{i0} - \sum_{m=1}^{M} a_{im} q_{im}, \forall i \in I_e \tag{3.12g}$$

$$l_j = b_{0j} - \sum_{n=1}^{N} b_{jn} r_{jn}, \forall j \in \tau \tag{3.12h}$$

$$\sum_{d \in J_k} \phi_d = w_{B_k}, \forall k \in K \tag{3.12i}$$

$$z_j = \xi_j + \sum_{i \in S_{1j}} y_{1i} + \sum_{i \in S_{2j}} y_{2i}, \forall j \in \tau \tag{3.12j}$$

$$y_{1i} = ln(p_i), \forall i \in I_e \tag{3.12k}$$

$$y_{2i} = ln(1 - p_i), \forall i \in I_e \tag{3.12l}$$

$$z_j = ln(\theta_j), \forall j \in \tau \tag{3.12m}$$

$$\xi_j = ln(l_j), \forall j \in \tau \tag{3.12n}$$

$$(p, l) \in \Omega \tag{3.12o}$$

$$(q, r) \in \mathscr{P} \tag{3.12p}$$

$$\left\{ \begin{array}{l} \theta_j^l \leq \theta_j \leq \theta_j^u, z_j^l \leq z_j \leq z_j^u, \text{ and } \xi_j^l \leq \xi_j \leq \xi_j^u, \forall j \in \tau \\ y_{1i}^l \leq y_{1i} \leq y_{1i}^u \text{ and } y_{2i}^l \leq y_{2i} \leq y_{2i}^u, \ \forall i \in I_e \end{array} \right\} \tag{3.12q}$$

$$\phi_d \text{ binary}, \forall d = 1, \ldots, D, \text{ and } 0 \leq w_J \leq 1, \forall J \in \mathscr{J}. \tag{3.12r}$$

Here, Ω is as specified in (3.5h), and we note that the bounds in (3.12q) depend on Ω (even as Ω will be modified via a partitioning process in our proposed algorithmic approach below) and are given by (3.7b), (3.9b), (3.10b), and (3.11b), respectively. For the sake of convenience in discussion, we shall denote the set of variables in Problem DTO(Ω), with obvious vector notation, as:

$$x \equiv (p, l, q, r, \phi, w, \psi, \theta, y_1, y_2, z, \xi),$$

where, in particular, $y_1 \equiv (y_{1i}, \forall i \in I_e)$ and $y_2 \equiv (y_{2i}, \forall i \in I_e)$. Observe that DTO($\Omega$) is, in general, a mixed-integer 0-1 factorable program (see Sherali and Wang [75] for continuous factorable programs). However, in our case, DTO(Ω) is linear except for the complicating identities (3.12c) and (3.12k) - (3.12n).

We next discuss some suitable polyhedral outer-approximation mechanisms to handle the univariate, monotone logarithmic functions in (3.12k) - (3.12n). Toward this end, generically denote

any identity in (3.12k) - (3.12n) as:

$$f = ln(\gamma), \text{ where } 0 < \gamma^l \leq \gamma \leq \gamma^\mu < \infty. \tag{3.13}$$

We then replace each such constraint (3.13) by the following affine convex envelope:

$$f \geq ln(\gamma^l) + \frac{(\gamma - \gamma^l)}{(\gamma^\mu - \gamma^l)}[ln(\gamma^\mu) - ln(\gamma^l)], \tag{3.14a}$$

along with some $H \geq 2$ tangential supports:

$$f \leq ln(\gamma_h) + \frac{(\gamma - \gamma_h)}{\gamma_h}, \text{ for } h = 1, \ldots H, \text{ where } \gamma^l \equiv \gamma_1 < \gamma_2 < \ldots < \gamma_H \equiv \gamma^\mu. \tag{3.14b}$$

In order to prescribe some judicious alternatives for selecting the points of tangency $\{\gamma_2, \ldots, \gamma_{H-1}\}$ (other than the interval end-points) in (3.14b), consider the following result:

Proposition 3.1. Consider any $0 < \bar{\gamma} < \hat{\gamma}$ such that $ln(\hat{\gamma}) - ln(\bar{\gamma}) = \Delta$. Then, for $\gamma \in [\bar{\gamma}, \hat{\gamma}]$, the maximum error, E, between the function $ln(\gamma)$ and its piecewise linear approximation defined by the tangential supports at $\bar{\gamma}$ and $\hat{\gamma}$ depends on Δ alone and is given by

$$E = ln[e^\Delta - 1] - ln(\Delta) + \frac{\Delta}{e^\Delta - 1} - 1. \tag{3.15}$$

Proof. By the monotonicity of both the logarithmic and the affine tangential supporting functions, the stated maximum error, E, occurs at the point of intersection γ^* of the tangential supports as given by $\gamma^* = \bar{\gamma}\hat{\gamma}\Delta/(\hat{\gamma} - \bar{\gamma})$, with

$$E = \frac{\hat{\gamma} ln(\hat{\gamma}) - \bar{\gamma} ln(\bar{\gamma})}{(\hat{\gamma} - \bar{\gamma})} - 1 - ln(\gamma^*). \tag{3.16}$$

Substituting for γ^* in (3.16), and writing $ln(\hat{\gamma}) = ln(\bar{\gamma}) + \Delta$, we get

$$E = ln\left(\frac{\hat{\gamma}}{\bar{\gamma}} - 1\right) - ln(\Delta) + \frac{\Delta}{[(\hat{\gamma}/\bar{\gamma}) - 1]} - 1,$$

which, upon using $\hat{\gamma}/\bar{\gamma} = e^\Delta$, yields (3.15). □

Corollary 3.1. Given any $\varepsilon > 0$, let $\Delta = \Delta_\varepsilon$ be the solution to (3.15) when we set $E = \varepsilon$. Now, suppose that we approximate the function $f \equiv ln(\gamma)$ on $[\gamma^l, \gamma^\mu] \subseteq (0, \infty)$ by H tangential supports

constructed at points uniformly distributed along the f-axis including the end-points, where

$$H = \left\lceil \frac{ln(\gamma^u) - ln(\gamma^l)}{\Delta_\varepsilon} \right\rceil + 1. \tag{3.17}$$

Then, the maximum approximation error will be bounded above by ε.

Proof. By Proposition 3.1 and since $\partial E/\partial \Delta > 0$ for $\Delta > 0$ in (3.15), the maximum approximation error will be bounded above by ε provided $[ln(\gamma^u) - ln(\gamma^l)]/(H-1) \leq \Delta_\varepsilon$, which yields (3.17). □

Accordingly, we define the *Bounded Error Strategy* **(BES)** for selecting points for generating the tangential supports (3.14b) as that prescribed by Corollary 3.1, given any error tolerance $\varepsilon > 0$. The number of tangential supports generated would then further depend on the bounding interval $[\gamma^l, \gamma^u]$ as given by (3.17). Alternatively, in order to control the number of tangential supports generated, we shall also apply BES with a prespecified value of H. In this case, we generate (3.14b) at some H uniformly distributed points along the f-axis, including the interval end-points. This yields

$$\gamma_h = exp\left\{ ln(\gamma^l) + \left(\frac{h-1}{H-1}\right) ln\left[\frac{\gamma^u}{\gamma^l}\right] \right\}, \text{ for } h = 1, \ldots, H.$$

We shall refer to this strategy as **BES(H)**. We recommend the value $H = 4$ based on our computational results reported in Section 3.4, where we investigated using $H = 4, \ldots, 20$.

Next, in order to generate a linear programming relaxation LP(Ω) for computing lower bounds in a branch-and-bound framework, we additionally adopt the following alternative linearization strategies for (3.12c) as identified in Section 3.2.1 below:

3.2.1 Linearization Techniques

In this section, we shall discuss two alternative schemes for linearizing (3.12c) that differ in their size or complexity and the relative tightness of the resulting LP relaxation. These are respectively denoted as linearization methods LM1 and LM2, and, together with the polyhedral outer-approximation (3.14) applied to (3.12k) - (3.12n), produce corresponding LP relaxations **LP1(Ω)** and **LP2(Ω)**, respectively. We shall also use the terminology **LP(Ω)** to refer generically to either of the foregoing relaxations LP1(Ω) and LP2(Ω).

Linearization Method LM1:

This is a standard linearization technique that utilizes the following constraints for each $J \in \mathscr{J}$ such that $|J| \geq 2$ (noting (3.12r)):

$$w_J \leq \phi_d, \forall d \in J, \text{ and } w_J \geq \sum_{d \in J} \phi_d - |J| + 1. \tag{3.18}$$

Note that as portended by (3.12c), and using (3.12r), when $\phi_d = 0$ for any $d \in J$, then (3.18) implies that $w_J = 0$, whereas when $\phi_d = 1,\ \forall d \in J$, then (3.18) implies that $w_J = 1$ as well.

Linearization Method LM2:

In this method, consider any decision-point node $k \in K$. Recall that B_k denotes the ordered set of decision indices that occur on the path from node 0 to node k. If $B_k = \varnothing$, then no additional constraints are generated for this node k. Else, suppose that $|B_k| \geq 1$. Then, we generate the restrictions:

$$w_{B_k+d} = \phi_d, \forall d \in J_k \text{ (for each } k \in K : B_k \neq \varnothing), \tag{3.19}$$

where we denote $w_{B_k+d} \equiv w_{B_k \cup \{d\}}$, with the index d appearing last in the resulting set $B_k \cup \{d\}, \forall d \in J_k$.

Proposition 3.2. The constraints (3.19) are valid and together with (3.12i) and (3.12r), imply that (3.12c) holds true.

Proof. First of all, note that (3.19) is valid since if $\phi_d = 0$ for any $d \in J_k$, then we must have $w_{B_k+d} = 0$ by its interpretation, and if $\phi_d = 1$ for any $d \in J_k$, then by (3.12i) and (3.12r), we must have $w_{B_k} = 1$, so that again the interpretation of w_{B_k+d} implies that we must have $w_{B_k+d} = 1$.

Next, let us establish that (3.12c) holds true by induction on $|J|$. Consider any $k \in K$ such that $B_k = \{d'\}$ and $d \in J_k$ so that $|J| = 2$ with $J \equiv \{d', d\}$ (the case $|J| = 1$ is trivial by definition). Now, if $\phi_{d'} = 0$, then (3.12i) implies that $\phi_d = 0$, so that (3.19) yields $w_{d'd} = 0$. On the other hand, if $\phi_{d'} = 1$, then (3.12i) implies that $\phi_d = 0$ or 1, which respectively yields $w_{d'd} = 0$ or 1 via (3.19).

Inductively, to complete the proof, consider $J = \{d_1, \ldots, d_h\}$, where $h \geq 3$, and assume that

(3.12c) holds true for any strict subset of this set. Hence, there exists $k \in K$ such that $B_k = \{d_1, \ldots, d_{h-1}\}$, with $d_h \in J_k$. Let us show that (3.12c) is satisfied for any such J. If $\phi_d = 0$ for any $d \in \{d_1, \ldots, d_{h-1}\}$, then by the induction hypothesis, we have that $w_{B_k} = 0$, which implies by (3.12i) that $\phi_d = 0$, $\forall d \in J_k$. Therefore, $\phi_{d_h} = 0$, which implies by (3.19) that $w_J = 0$. Else, if $\phi_{d_1} = \ldots = \phi_{d_{h-1}} = 1$, then $w_{B_k} = 1$ by the induction hypothesis, and so by (3.12i), $\phi_{d_h} = 0$ or 1, which respectively yields $w_J = 0$ or 1 via (3.19), and so (3.12c) is again satisfied in either case. □

The next result demonstrates that LM2 yields a *tighter representation* than LM1, i.e., its defining constraints imply those of LM1 in the continuous (LP) sense.

Proposition 3.3. The constraints represented in (3.19), (3.12i), and $\phi \geq 0$ of LM2 imply those in (3.18) of LM1.

Proof. Consider any $J' \in \mathscr{J}$ and assume without loss of generality that $J' = \{1, \ldots, h\}$, with $\phi_1, \ldots, \phi_h$ occurring in this order along the path from node 0 to some node j. Hence, by (3.18), LM1 generates the constraints

$$w_{J'} \leq \phi_d, \ \forall d = 1, \ldots, h, \tag{3.20a}$$

$$w_{J'} \geq \sum_{d=1}^{h} \phi_d - (h-1). \tag{3.20b}$$

Now, by (3.19) applied to the node from which the arc having ϕ_h emanates, LM2 directly produces $w_{J'} = \phi_h$ and (3.12i) yields $\phi_h \leq w_{J'-\{h\}}$, which, by (3.19) applied at the node from which the arc having the variable ϕ_{h-1} emanates, produces $w_{J'-\{h\}} = \phi_{h-1}$. Hence, $w_{J'} \leq \phi_{h-1}$ as well. Continuing in this fashion along the chain from node j to node 0 yields that (3.20a) is implied.

To complete the proof, we next show that (3.20b) is implied as well. Note that (3.19) yields for any $j_1 \in J_k$, $k \in K$, that

$$w_{B_k + j_1} = \phi_{j_1} \geq w_{B_k} + \phi_{j_1} - 1, \tag{3.21}$$

where the last inequality follows from the fact that $w_{B_k} \leq 1$ in (3.12r). Hence, applying (3.21) at the node from which the arc having the variable ϕ_h emanates with $B_k + j_1 \equiv J'$ and $j_1 \equiv h$ we get,

$$w_{J'} \geq w_{J'-\{h\}} + \phi_h - 1. \tag{3.22}$$

Repeating this at the node from which the arc having the variable ϕ_{h-1} emanates yields $w_{J'-\{h\}} \geq w_{J'-\{h,h-1\}} + \phi_{h-1} - 1$, which, together with (3.22), implies that

$$w_{J'} \geq w_{J'-\{h,h-1\}} + \phi_h + \phi_{h-1} - 2. \tag{3.23}$$

Continuing (3.23) along the chain from node j to node 0, we get

$$w_{J'} \geq w_{J'-\{h,h-1,\ldots,2\}} + \phi_h + \phi_{h-1} + \ldots + \phi_2 - (h-1) = \sum_{d=1}^{h} \phi_d - (h-1),$$

where the last equality follows by noting that $w_{J'-\{h,h-1,\ldots,2\}} = w_1 \equiv \phi_1$. Hence, (3.20b) is also implied. □

Now, for any $j \in K \cup I_e \cup \tau$, let $d(j)$ be the first decision index that is encountered in the (reverse) path from node j to the root node (whenever $d(j)$ does not exist, we define $\phi_{d(j)} \equiv 1$). Then, we can simplify LM2 by eliminating the w_J-variables upon using (3.19), which effectively equates $w_{S_{3j}} = \phi_{d(j)}, \forall j \in \tau$ in (3.12b) and $w_{B_k} = \phi_{d(k)}, \forall k \in K$ in (3.12i). Hence, by Proposition 3.2, *LM2 can be equivalently written by using these substitutions in* (3.12b) *and* (3.12i)*, and eliminating* (3.12c) *and the* w_J*-inequalities in* (3.12r). The following result reveals a partial convex hull representation inherent within LM2.

Proposition 3.4. Consider the following polyhedral set defined by the constraints of LM2.

$$\Lambda \equiv \{\phi \geq 0 : \sum_{d \in J_k} \phi_d = \phi_{d(k)}, \forall k \in K\}. \tag{3.24}$$

Then, Λ is nonempty and compact with binary-valued extreme points.

Proof. The compactness of Λ follows readily by noting that the constraints in Λ imply that $0 \leq \phi_d \leq 1, \forall d = 1, \ldots, D$, recalling that $\phi_{d(k)} \equiv 1$ whenever $d(k)$ is null. Now, to complete the proof, let us show that for any arbitrary objective vector $(C_d, d = 1, \ldots, D)$, the linear program

$$\text{Minimize } \left\{ \sum_{d=1}^{D} C_d \phi_d : \phi \in \Lambda \right\} \tag{3.25}$$

has an optimal solution for which ϕ_d is binary-valued.

We solve (3.25) sequentially as follows. Consider any decision node $k \in \arg\max\{|B_k|\}$, and notice that the variables ϕ_d for $d \in J_k$ appear only in the single corresponding constraint for k in (3.24). Hence, there exists an optimal solution in which $\phi_{\hat{d}} = \phi_{d(k)}$, where $\hat{d} \in \arg\min_{d \in J_k}\{C_d\}$, and $\phi_d = 0,\ \forall d \in J_k - \{\hat{d}\}$. This eliminates the variables ϕ_d for $d \in J_k$ from (3.25) along with the associated constraint for this k in (3.24), where, whenever $B_k \neq \varnothing$ (so that $d(k)$ exists), we also update the objective coefficient $C_{d(k)}$ for the upstream (toward the root node) variable $\phi_{d(k)}$ according to $C_{d(k)} \leftarrow C_{d(k)} + C_{\hat{d}}$. Repeating this step, we will finally solve separable problems for decision nodes $k \in K$ having $B_k = \varnothing$, for which the reduced linear program is given as follows for some transformed objective coefficients $(\hat{C}_d, d \in J_k)$:

$$\text{Minimize}\left\{\sum_{d \in J_k} \hat{C}_d \phi_d : \sum_{d \in J_k} \phi_d = 1, \phi_d \geq 0, \forall d \in J_k\right\},$$

and for which there exists an optimal binary solution for $\phi_d,\ d \in J_k$. By back-substituting these binary values for the recorded solutions for the downstream decision nodes, we will get a binary optimal solution for $\phi_d,\ d \in J_k, \forall k \in K$. □

Illustrative Example

Consider the illustration of the decision tree in Figure 3.1 that contains the restrictions (3.5g):

$$\phi_1 + \phi_2 + \phi_3 = 1,$$

$$\phi_4 + \phi_5 = \phi_1,$$

$$\phi_6 + \phi_7 = \phi_3,$$

$$\phi_8 + \phi_9 = \phi_3\phi_7,$$

along with product terms of the type (3.8) for $J \in \mathscr{J}$ as given by:

$$\phi_1\phi_4,\ \phi_1\phi_5,\ \phi_3\phi_6,\ \phi_3\phi_7,\ \phi_3\phi_7\phi_8, \text{ and } \phi_3\phi_7\phi_9.$$

For this situation, the methods LM1 and LM2 will produce the following additional constraints and variables to replace (3.12c) in Problem DTO:

LM1 (Equation (3.18)):

$$d=4:\ \{w_{14}\le\phi_1,\ w_{14}\le\phi_4,\ w_{14}\ge\phi_1+\phi_4-1\}$$
$$d=5:\ \{w_{15}\le\phi_1,\ w_{15}\le\phi_5,\ w_{15}\ge\phi_1+\phi_5-1\}$$
$$d=6:\ \{w_{36}\le\phi_3,\ w_{36}\le\phi_6,\ w_{36}\ge\phi_3+\phi_6-1\}$$
$$d=7:\ \{w_{37}\le\phi_3,\ w_{37}\le\phi_7,\ w_{37}\ge\phi_3+\phi_7-1\}$$
$$d=8:\ \{w_{378}\le\phi_3,\ w_{378}\le\phi_7,\ w_{378}\le\phi_8, w_{378}\ge\phi_3+\phi_7+\phi_8-2\}$$
$$d=9:\ \{w_{379}\le\phi_3,\ w_{379}\le\phi_7,\ w_{379}\le\phi_9,\ w_{379}\ge\phi_3+\phi_7+\phi_9-2\}.$$

LM2 (Equation (3.19), where these identities are substituted into (3.12b) and (3.12i)):

$$k=11:\ \{w_{14}=\phi_4,\ w_{15}=\phi_5\}$$
$$k=12:\ \{w_{36}=\phi_6,\ w_{37}=\phi_7\}$$
$$k=13:\ \{w_{378}=\phi_8,\ w_{379}=\phi_9\}.$$

Remark 3.4. Observe that the size of the decision tree can be reduced whenever there exist consecutive decision-point nodes. If a decision-point node k_1, having the variable ϕ_{d_1} associated with an emanating arc is immediately followed by another decision-point node k_2 having $J_{k_2}\equiv\{d_2,d_3,\ldots,d_n\}$, we can collapse node k_2 into k_1 and define new variables $\phi_{d_1d_2},\phi_{d_1d_3},\ldots,\phi_{d_1d_n}$ associated with corresponding arcs emanating from k_1 to replace the variables $\phi_{d_1},\phi_{d_2},\ldots,\phi_{d_n}$. Compared to the original decision tree, we reduce the number of decision-point nodes and the number of decision alternatives by one for each such step. To illustrate this for the decision tree of Figure 3.1, note that we can collapse both nodes 12 and 13 (sequentially adopting the foregoing step) into node 10, and generate path-based arcs connecting node 10 to nodes 21, 22, and 23, having respective associated variables ϕ_{36},ϕ_{378}, and ϕ_{379}. The constraints (3.5g) in this case would then be written as follows:

$$\phi_1+\phi_2+\phi_{36}+\phi_{378}+\phi_{379}=1,$$

$$\phi_4+\phi_5=\phi_1,$$

and the product terms of the type (3.8) for $J\in\mathscr{J}$ would be given by

$$\phi_1\phi_4 \text{ and } \phi_1\phi_5.\ \square$$

3.2.2 Structure of Optimal Solutions

Let $\bar{x} = (\bar{p}, \bar{l}, \bar{q}, \bar{r}, \bar{\phi}, \bar{w}, \bar{\psi}, \bar{\theta}, \bar{y_1}, \bar{y_2}, \bar{z}, \bar{\xi})$ represent an optimal solution to LP(Ω). To reduce the size of this relaxation, we *a priori* identify constraints that would be inactive at optimality by analyzing the structural behavior of optimal solutions. In this spirit, Proposition 3.5 below establishes that the affine convex envelope for the constructed outer-approximation of the functional form $z_j = ln(\theta_j), j \in \tau$ can be omitted without affecting optimality.

Proposition 3.5. In LP(Ω), the affine convex envelope of $ln(\theta_j)$, for any $j \in \tau$, will not be active at optimality unless if the optimal θ_j-value is at its lower or upper bound. Moreover, at least one of the tangential supports will be active.

Proof. Given a feasible solution x', let the variables $(p, l, q, r, \phi, w, y_1, y_2, \xi, z)$ be fixed in LP(Ω) according to x'. The resulting linear program in ψ and θ effectively bounds θ_j as $\underline{\theta_j} \leq \theta_j \leq \overline{\theta_j}, \forall j \in \tau$, where $\underline{\theta_j}$ and $\overline{\theta_j}$ are respectively determined by some tangential support (3.14b) and the affine convex envelope (3.14a), corresponding to the left-hand side in (3.14a) and (3.14b) fixed at z'_j. By the nature of the objective function (3.12a) and the constraints (3.12b), we would therefore have $\theta_j = \underline{\theta_j}, \ \forall j \in \tau$, at an optimal solution. Hence, for each $j \in \tau$, some tangential support is active at optimality and the affine convex envelope is inactive at optimality unless z'_j, and therefore θ_j, equals its original lower or upper bound. □

Remark 3.5. By the nature of the objective function (3.12a) and the constraint relationships (3.12b), (3.12g), (3.12h), and (3.12j) in LP(Ω), the y_1-, y_2-, and ξ-variables tend to be at their lower/upper bounds at optimality. Although removing the tangential supports associated with the corresponding functional forms (3.12k), (3.12l), and (3.12n) other than those at the interval end-points might worsen the lower bound obtained via the relaxed problem, the total computational time may improve as a result of the decrease in the size of LP(Ω) that is solved at each node of the branch-and-bound tree. Hence, in our computations, we shall experiment with this reduced modeling strategy. □

3.2.3 Further Properties of the LP Relaxation

The next set of results lay the groundwork for composing our proposed global optimization strategy for solving Problem DTO. Henceforth, for any Problem P, we shall denote its optimal value as $v[P]$.

Also, let $\bar{x}$ is an optimal solution to LP(Ω).

Proposition 3.6. v[LP(Ω)] gives a lower bound for v[DTO(Ω)]. Moreover, if $\bar{x}$ solves LP(Ω) and satisfies (3.12k) - (3.12n) and (3.12r), then $\bar{x}$ also solves DTO(Ω) with the same objective value.

Proof. Follows from the construction of LP(Ω), noting that under the hypothesis of the proposition and the validity of LMr, r=1,2, we also then have that (3.12c) holds true. □

Proposition 3.7. Let $\bar{x}$ solve LP(Ω). Let $(\hat{l},\hat{q},\hat{r}) \equiv (\bar{l},\bar{q},\bar{r})$ and let $\hat{p}$ be computed by (3.5e), i.e., $\hat{p}_i = \bar{g}_i/(1+\bar{g}_i)$ where $\bar{g}_i = exp\{a_{i0} - \sum_{m=1}^{M} a_{im}\bar{q}_{im}\}$, $\forall i \in I_e$. Furthermore, set $\hat{\theta}_j \equiv \hat{l}_j \prod_{i\in S_{1j}} \hat{p}_i \prod_{i\in S_{2j}} (1-\hat{p}_i), \forall j \in \tau$, and let $\hat{\phi}$ be a binary optimal solution to the linear program composed in Proposition 3.4 as given by:

$$\hat{LP}: \text{ Minimize } \left\{ \sum_{d=1}^{D} c_d\phi_d + \sum_{j\in\tau} \hat{\theta}_j \phi_{d(j)} : \phi \in \Lambda \right\}. \tag{3.28}$$

Then, $(\hat{p},\hat{l},\hat{q},\hat{r},\hat{\phi})$ is a feasible solution to Problem DTO with objective value $v[\hat{LP}]$.

Proof. From (3.12q) (as in (3.9b)), we have $ln(p_i^l) \le \bar{y}_{1i} \le ln(p_i^u)$ and $ln(1-p_i^u) \le \bar{y}_{2i} \le ln(1-p_i^l)$. Thus, from (3.12g), $ln(p_i^l) - ln(1-p_i^l) \le \bar{y}_{1i} - \bar{y}_{2i} = a_{i0} - \sum_{m=1}^{M} a_{im}\bar{q}_{im} \le ln(p_i^u) - ln(1-p_i^u)$. Hence, since $\hat{p}$ has been computed by using the constraint (3.5e), we have

$$ln\Big(\frac{p_i^l}{1-p_i^l}\Big) \le ln\Big(\frac{\hat{p}_i}{1-\hat{p}_i}\Big) = a_{i0} - \sum_{m=1}^{M} a_{im}\bar{q}_{im} \le ln\Big(\frac{p_i^u}{1-p_i^u}\Big),$$

which results in $p_i^l \le \hat{p}_i \le p_i^u$. Thus, from the constraints of LP(Ω), we have that $(\hat{p},\hat{l},\hat{q},\hat{r})$ is feasible to (3.5b) - (3.5f), (3.5h), and (3.5i). Furthermore, by the structure of LM2 and Propositions 3.2 and 3.4, $\hat{LP}$ defined by (3.28) then yields a binary optimal solution $\hat{\phi}$ that represents the best optimal completion $(\hat{p},\hat{l},\hat{q},\hat{r},\hat{\phi})$ to the foregoing partial solution. □

Proposition 3.8. Let $\bar{x}$ solve LP(Ω) with objective value v[LP(Ω)]. If each of the variable values $\bar{p}_i$, $\bar{l}_j$, and $\bar{\theta}_j$ equals either its corresponding lower or upper bound and $\bar{\phi}$ is binary-valued, then $\bar{x}$ solves DTO(Ω) with objective value v[DTO(Ω)] = v[LP(Ω)].

Proof. From the construction of LP(Ω), it is sufficient to show that $\bar{x}$ satisfies (3.12c) and (3.12k) - (3.12n). For the generic case (3.13), if γ equals either of its bounds, then Constraints (3.14a) and

(3.14b) imply that $f = ln(\gamma)$. Likewise, if each of the variable values $\bar{p}_i$, $\bar{l}_j$, and $\bar{\theta}_i$ equals either of its corresponding bounds, then (3.12k) - (3.12n) are satisfied. Moreover, as established for LMr, r = 1, 2, in Section 3.2.1 and Proposition 3.2, if $\bar{\phi}$ is binary-valued, then, (3.12c) is satisfied. Thus, $\bar{x}$ is feasible to DTO(Ω). □

Proposition 3.9. Let Ω be such that $p_i^l = p_i^u, \forall i \in I_e$, and $l_j^l = l_j^u, \forall j \in \tau$. If $\bar{x}$ solves LP(Ω) with $\bar{\phi}$ being binary-valued, then $\bar{x}$ also solves DTO(Ω).

Proof. If $p_i^l = p_i^u$ and $l_j^l = l_j^u$, we have that $\theta_j^l = \theta_j^u$ by (3.7b). The proof now follows from Proposition 3.8. □

3.2.4 Optimality-Induced Valid Inequalities

In order to further tighten the model representation by legitimately reducing the search region as well as improving the underlying continuous LP relaxation, we introduce in this section certain valid inequalities (denoted **VIs**) that are implied by optimality (rather than feasibility) considerations.

Proposition 3.10. There exists an optimal solution to Problem DTO satisfying the following inequalities:

$$q_{im} \leq s_m \phi_{d(i)}, \ \forall i \in I_e, \ m = 1, \ldots, M, \tag{3.29a}$$

$$r_{jn} \leq t_n \phi_{d(j)}, \ \forall j \in \tau, \ n = 1, \ldots, N. \tag{3.29b}$$

Proof. First of all, note that Problem DTO has an optimal solution since it is bounded and feasible (the solution $(q, r) = (0, 0)$, with $p_i = p_i^u \equiv p_i^{u(3.3)}, \ \forall i \in I_e, \ l_j = l_j^u \equiv l_j^{u(3.4)}, \ \forall j \in \tau$, and any ϕ satisfying (3.5g) and (3.5j) gives a feasible solution). Now, for any $i \in I_e$, if $\phi_{d(i)} = 0$, then event i is inconsequential to the problem, and so, we need not allocate any event-related resources to reduce p_i below p_i^u in (3.5e). Hence, we can set $q_{im} = 0, \ \forall m = 1, \ldots, M$, i.e., (3.29a) is valid. On the other hand, if $\phi_{d(i)} = 1$, then (3.29a) is again valid because it is implied by (3.5b). Likewise, (3.29b) is satisfied at an optimal solution because for any $j \in \tau$, if $\phi_{d(j)} = 0$, then consequence $j \in \tau$ does not arise (or does not impact the objective function), and we can therefore set $r_{jn} = 0, \ \forall n = 1, \ldots, N$, while if $\phi_{d(j)} = 1$, then (3.29b) is implied by (3.5c). □

Henceforth, we shall assume that the inequalities (3.29a) and (3.29b) are incorporated within Problem DTO, and hence, within DTO(Ω) and LP(Ω), $\forall\,\Omega$. In addition, we shall perform the following variable fixings in Problem DTO(Ω) and LP(Ω) as prompted by Proposition 3.11 below. This strategy will become relevant in the sequel when we revise the bounds on the p- and l-variables in a branch-and-bound framework. Naturally, if any such fixings render a particular node subproblem infeasible, then we can fathom this node.

Proposition 3.11. Consider any (node subproblem) DTO(Ω) predicated on a set of imposed bounds Ω (and incorporating the VIs (3.29a) and (3.29b)). Then,

$$p_i^u < p_i^{u(3.3)} \Rightarrow \phi_d = 1,\ \forall d \in B_i,\ \text{for each } i \in I_e, \tag{3.30a}$$

$$l_j^u < l_j^{u(3.4)} \Rightarrow \phi_d = 1,\ \forall d \in B_j,\ \text{for each } j \in \tau. \tag{3.30b}$$

Proof. Consider any $i \in I_e$, and suppose that the currently imposed upper bound p_i^u on p_i satisfies $p_i^u < p_i^{u(3.3)}$. Then, by (3.12g), (3.12k), and (3.12l), we have that $q_{im} > 0$ for some $m \in \{1,\ldots,M\}$, and so, (3.29a) and (3.12r) imply that $\phi_{d(i)} = 1$. But since the constraints (3.12c, 3.12i, 3.12r) imply (3.19), we hence have $\phi_d = 1, \forall d \in B_i$. Similarly, for any $j \in \tau$, if the currently imposed upper bound l_j^u on l_j satisfies $l_j^u < l_j^{u(3.4)}$, then (3.12h) implies that $r_{jn} > 0$ for some $n \in \{1,\ldots,N\}$, and then (3.29b) and (3.19) yield $\phi_d = 1,\ \forall d \in B_j$. □

3.2.5 Upper Bounding Scheme: Procedure UB

In this subsection, we use Proposition 3.7 to develop a method for computing upper bounds on Problem DTO based on the solution of any lower bounding relaxation LP(Ω). (If any optimization problem is detected to be infeasible in this process, we abort the procedure.)

Procedure UB:

Step 1: Solve LP(Ω) (including Constraints (3.29)) and obtain an optimal solution $\bar{x}$. Apply Proposition 3.7 to obtain the solution $(\hat{p}, \hat{l}, \hat{q}, \hat{r}, \hat{\phi})$.

Step 2: Fix $\phi = \hat{\phi}$, resolve LP(Ω) to obtain an optimum $\bar{x}$, and apply Proposition 3.7 to derive a possibly revised solution $(\hat{p}, \hat{l}, \hat{q}, \hat{r}, \hat{\phi})$. Repeat this step until no further improvement is obtained in the objective function value for Problem DTO.

Step 3: Fix $\phi = \hat{\phi}$, and use a nonlinear programming (local search) solver to optimize DTO(Ω), starting with the solution obtained at Step 2 as an initial solution. (We used SNOPT Version 7 (Gill et al. [26]) for this purpose.) Output the resulting solution value as an upper bound on Problem DTO, and update the incumbent solution x^* and its objective value v^*, if necessary.

3.2.6 Range Reduction

The imposed range for each ϕ-, p-, l-, and θ-variable can be tightened by sequentially solving a pair of linear programs that minimize and maximize each variable in turn over the feasible region of LP(Ω), while additionally restricting the original objective function to take on values lesser than or equal to the best known upper bound v^* obtained by Procedure UB.

Starting with the ϕ-variables, note that if the minimum value of ϕ_d is positive for any $d \in \{1,\ldots,D\}$, then we can fix $\phi_d \equiv 1$, and likewise, if the maximum value is less than one, we can fix $\phi_d \equiv 0$. Next, considering the p-variables, the foregoing pair of associated linear programs is used to update the lower and upper bounds on each p_i-variable in turn, where we also update the bounds on the affected θ_j-variables using Equation (3.32), for each j such that $i \in S_{1j} \cup S_{2j}$. Moreover, noting that the bounds on the p- and θ-variables define the bounds on the y_1-, y_2-, and z-variables according to (3.9b) and (3.10b), if any of the bounds on the former variables are revised, then the bounds on the corresponding latter variables are also updated. The same procedure is followed for l_j, $j \in \tau$, during which the bounds on θ_j along with the bounds on the corresponding z- and ξ-variables are also updated using (3.32), (3.10b), and (3.11b). Finally, a pair of linear programs is solved to directly tighten the bounds on $\theta_j, \forall j \in \tau$.

Following this range reduction process, the polyhedral outer approximations for the logarithmic relationships are constructed based on the revised bounds on the variables. Next, LP(Ω) is (re-) solved and the lower and upper bounds on Problem DTO are updated as possible. Note that at each node of the branch-and-bound tree, we invoke range reduction only once.

3.3 Global Optimization Branch-and-Bound Algorithms

We now design three alternative approaches to solve Problem DTO via DTO(Ω) and its relaxation LP(Ω). In **Algorithm A**, we utilize a specialized branch-and-bound process based on partitioning the hyperrectangle Ω, where the bounds on the variables $(\psi,\ \theta,\ y_1,\ y_2,\ z,\ \xi)$ are accordingly computed by (3.7e), (3.7b), (3.9b), (3.9b), (3.10b), and (3.11b), respectively. For any node subproblem DTO(Ω) that is associated with a particular Ω, we construct the relaxation LP(Ω) and solve it to compute a lower bound. Let $\bar{x}$ solve LP(Ω). If the conditions of Proposition 3.6 hold, we will have also solved subproblem DTO(Ω). Otherwise, we apply Procedure UB to find a feasible solution to Problem DTO and update the incumbent solution and the associated upper bound if possible, perform range reductions to update Ω and (3.12q), and as necessary, we branch at this node by partitioning Ω as follows:

Branching Rule A: While selecting the branching variable, priority is given to the ϕ_d-variables. Thus, we first check if $\bar{\phi}$ is binary-valued. If not, then we define the set $K' = \{k \in K : (\bar{\phi}_d,\ d \in J_k)$ is not binary-valued$\}$ and find $\hat{k} \in \arg\operatorname{lexmin}_{k\in K'}\{|B_k|, \min_{d\in J_k}|\bar{\phi}_d - 0.5|\}$. We then branch on $\phi_{\hat{d}}$, where $\hat{d} \in \arg\min_{d\in J_{\hat{k}}}\{|\bar{\phi}_d - 0.5|\}$, by using the dichotomy that $\phi_{\hat{d}} = 1 \vee \phi_{\hat{d}} = 0$. Note that Constraints (3.12b) and (3.12i) together with Equations (3.18) and (3.19) are invoked to fix additional (ϕ, w)-variables as possible whenever we branch on a ϕ-variable. On the other hand, if $\bar{\phi}$ is binary-valued, then we find a variable p_i or l_j having the largest bounding interval according to:

$$\max\{(p_i^u - p_i^l), (l_j^u - l_j^l), \forall i, j\}, \tag{3.31a}$$

where ties are broken by favoring the variable that gives the largest discrepancy in

$$\max\{|\bar{y}_{1i} - ln(\bar{p}_i)|, |\bar{y}_{2i} - ln(1-\bar{p}_i)|, |\bar{\bar{\xi}}_j - ln(\bar{l}_j)|, \forall i, j\}, \tag{3.31b}$$

where the first two terms in (3.31b) relate to p_i and the third term to l_j. If the identified term in (3.31) relates to p_i, then we branch on this variable by partitioning its interval according to the dichotomy that $p_i \in [p_i^l, (p_i^l + p_i^u)/2] \ \vee \ p_i \in [(p_i^l + p_i^u)/2, p_i^u]$. If the identified term corresponds to l_j, then we partition its interval according to the dichotomy that $l_j \in [l_j^l, (l_j^l + l_j^u)/2] \ \vee \ l_j \in [(l_j^l + l_j^u)/2, l_j^u]$. $\square$

Algorithm B is the same as Algorithm A, except that we also include θ_j in the partitioning process, as motivated by Proposition 3.8. In this case, given any imposed bounds $[\theta_j^l, \theta_j^u]$ on the variable θ_j, $\forall j \in \tau$, we update these bounds based on the implied bounds derived via (3.7b) according to:

$$\left[\begin{array}{l} \theta_j^l \leftarrow max\{\theta_j^l,\ l_j^l \prod_{i \in S_{1j}} p_i^l \prod_{i \in S_{2j}} (1-p_i^u)\} \\ \theta_j^u \leftarrow min\{\theta_j^u,\ l_j^u \prod_{i \in S_{1j}} p_i^u \prod_{i \in S_{2j}} (1-p_i^l)\} \end{array} \right], \forall j \in \tau. \tag{3.32}$$

Now, define Ω' as

$$\Omega' \equiv \{(p,l,\theta) : p_i^l \leq p_i \leq p_i^u, \forall i,\ l_j^l \leq l_j \leq l_j^u, \forall j,\ \theta_j^l \leq \theta_j \leq \theta_j^u, \forall j\}, \tag{3.33}$$

and let DTO(Ω') be identical to DTO(Ω), except that we primarily impose the bounds $(p,l,\theta) \in \Omega'$, and then compute the bounds on the variables $(\psi,\ y_1,\ y_2,\ z,\ \xi)$ using (3.7e), (3.9b), (3.9b), (3.10b), and (3.11b), respectively. Whenever range reduction is performed, all these bounds are updated accordingly.

At each node of the branch-and-bound tree for Algorithm B, we proceed exactly as in Algorithm A except that we now solve the relaxation problem LP(Ω') for computing lower bounds, and also, for the partitioning scheme, the Branching Rule A is substituted by the following, where $\bar{x}$ solves LP(Ω'):

Branching Rule B: We apply the same scheme as in Branching Rule A in case $\bar{\phi}$ is not binary-valued. Else, we find a variable that yields the maximum discrepancy according to:

$$\max\{|\bar{y}_{1i} - ln(\bar{p}_i)|, |\bar{y}_{2i} - ln(1-\bar{p}_i)|, |\bar{\xi}_j - ln(\bar{l}_j)|, |\bar{z}_j - ln(\bar{\theta}_j)|, \forall i,j\}, \tag{3.34}$$

where ties are broken by favoring the variable having the largest bounding interval. If the maximum is attained by one of the first two terms, then we branch on the corresponding variable p_i by partitioning its interval according to $[p_i^l, \bar{p}_i] \vee [\bar{p}_i, p_i^u]$. Similarly, if the maximum is attained by the third or fourth term, we branch with respect to l_j or θ_j, partitioning their intervals as $[l_j^l, \bar{l}_j] \vee [\bar{l}_j, l_j^u]$, or $[\theta_j^l, \bar{\theta}_j] \vee [\bar{\theta}_j, \theta_j^u]$, respectively. □

The third alternative, called **Algorithm C**, can be viewed as a combination of Algorithms A and B. Motivated by Propositions 3.8 and 3.9, we adopt the following partitioning scheme:

Branching Rule C: If $\bar{\phi}$ is not binary-valued, then we apply Branching Rule A as before. Else, we select a variable having the largest bounding interval among the set of variables having at least a δ-deviation from their associated logarithmic functions, where $0 \leq \delta \leq 10^{-1}$. The value of δ changes through the optimization process depending on the number of variables that violate the exact corresponding logarithmic relations. Specifically, we select a variable p_i or l_j having the largest bounding interval according to:

$$\max\{(p_i^u - p_i^l), (l_j^u - l_j^l), \forall i \in I', \forall j \in \tau'\}, \tag{3.35}$$

where the index sets I' and τ' are defined by

$$I' \equiv \{i : |\bar{y}_{1i} - ln(\bar{p}_i)| > \delta\} \cup \{i : |\bar{y}_{2i} - ln(1-\bar{p}_i)| > \delta\} \text{ and } \tau' \equiv \{j : |\bar{\xi}_j - ln(\bar{l}_j)| > \delta\}.$$

If both I' and τ' are empty, we set $\delta \leftarrow \delta/10^v$ for the smallest integer $v \geq 1$ such that at least one of these index sets becomes nonempty, and then apply (3.35). For the selected branching variable, we split its interval at the geometric mean, i.e., if the identified term in (3.35) corresponds to some p_i-variable, then we partition its interval according to $[p_i^l, \sqrt{p_i^l p_i^u}] \vee [\sqrt{p_i^l p_i^u}, p_i^u]$, and if it corresponds to some l_j-variable, we partition its interval according to $[l_j^l, \sqrt{l_j^l l_j^u}] \vee [\sqrt{l_j^l l_j^u}, l_j^u]$. □

Remark 3.6. Note that, for a function $f = ln(\gamma), 0 < \gamma^l < \gamma < \gamma^u < \infty$, splitting the γ-interval at its geometric mean corresponds to bisecting the implied bounds on the f-variables at its arithmetic mean, i.e., at $[ln(\gamma^l) + ln(\gamma^u)]/2$. Figure 3.2 displays this feature, and exhibits the evident potential for generating improved bounds via the child-nodes by using the geometric mean splitting technique. For the sake of comparison, we shall also implement the interval partitioning technique predicated by Branching Rules A and B within Branching Rule C. □

Now, we formally describe the branch-and-bound procedure Algorithm A for solving Problem DTO. At each stage s of this algorithm, $s = 0, 1, \ldots$, we define A_s as the set of non-fathomed, or *active nodes*. Each active node $a \in A_s$ is associated with a hyperrectangle Ω^a. A lower bound LB_a on an active node is obtained by solving the linear programming relaxation LP(Ω^a) to yield $LB_a = v[\text{LP}(\Omega^a)]$. At each stage s, we define the global lower bound on Problem DTO by

$$LB(s) \equiv \min\{LB_a : a \in A_s\}.$$

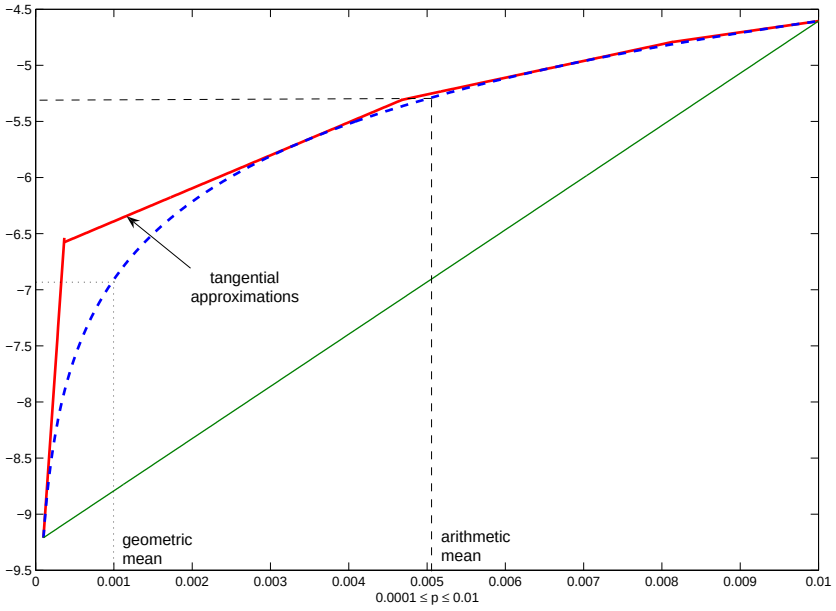


Figure 3.2: Partitioning at the arithmetic mean versus the geometric mean.

Whenever LP(Ω^a) is solved for a node $a \in A_s$, we apply Procedure UB to possibly find a feasible solution for updating the upper bound for Problem DTO, and perform range reductions as necessary. These bounds are used in concert with a least lower bound node selection rule and the aforementioned partitioning schemes as detailed below.

3.3.1 Branch-and-Bound Algorithm A for Problem DTO

Step 0: Initialization. Set $s = 0$, $A_s = \{0\}$, $a(s) = 0, a = 0$, and $\Omega^0 = \Omega$. Solve LP(Ω^0) and let x^0 be the optimal solution obtained. Set $LB_0 = v[\text{LP}(\Omega^0)]$. Apply Procedure UB of Section 3.2.5 with $\bar{x} \equiv x^0$ to derive an incumbent solution x^* with objective value v^*. Invoke the range reduction scheme of Section 3.2.6 to revise LB_0 and (x^*, v^*) as possible. If $LB_0 \geq v^*(1-\varepsilon)$ for some optimality tolerance $\varepsilon \geq 0$, then stop with the incumbent solution as (ε)-optimal to Problem DTO. Otherwise, proceed to Step 1.

Step 1: Partitioning Step. Invoke the Branching Rule A to partition the selected node $a(s)$ into two subnodes $a+1$ and $a+2$ with associated hyperrectangles Ω^{a+1} and Ω^{a+2}, respectively. Replace $A_s \leftarrow A_s \cup \{a+1, a+2\} - \{a(s)\}$.

Step 2: Bounding Step. Solve the relaxed problems LP(Ω^{a+1}) and LP(Ω^{a+2}) after fixing ϕ-variables as possible using Proposition 3.11. Apply Procedure UB to the solutions found at each node and update the incumbent solution if possible, and perform range reductions as necessary.

Step 3: Fathoming Step. Fathom any non-improving node and update $A_{s+1} \leftarrow A_s - \{a \in A_s : LB_a \geq v^*(1-\varepsilon)\}$. Increment s by 1.

Step 4: Termination Check and Node Selection Step. If $A_s = \varnothing$, stop with the incumbent solution as an $\varepsilon-$optimum. Otherwise, select a node $a(s) \in \text{argmin}\{LB_a : a \in A_s\}$ and return to Step 1.

Proposition 3.12. (Main Convergence Result)
Algorithm A (with $\varepsilon = 0$) either terminates finitely with the incumbent solution as an optimum to DTO, or else, an infinite sequence of stages is generated such that along any infinite branch of the tree, any accumulation point of the (p,l,q,r,ϕ)-variable part of the sequence of linear programming relaxation solutions solves Problem DTO.

Proof. The case of finite termination is clear. Hence, suppose that an infinite sequence of stages is generated. Consider any infinite branch of the tree having a nested hyperrectangle sequence $\Omega^{a(s)}$ for s belonging to some index set S. For each stage s, we have $LB(s) = LB_{a(s)} = v[\text{LP}(\Omega^{a(s)})], \forall s \in S$. For each node $a(s)$, let $x^{a(s)}$ solve LP($\Omega^{a(s)}$). By the compactness of the feasible region, there exists a convergent subsequence for $\{x^{a(s)}, \Omega^{a(s)}\}_{s\in S}$. Without loss of generality, assume that $\{x^{a(s)}, \Omega^{a(s)}\}_{s\in S} \rightarrow (x^*, \Omega^*)$. We must show that the $(p^*, l^*, q^*, r^*, \phi^*)$-variable part of x^* solves Problem DTO.

Note that $LB_{a(s)} = \min\{LB_a : a \in A_s\} \leq v[\text{DTO}], \forall s \in S$, which is also preserved in the limit:

$$v^* \equiv \lim_{s\rightarrow\infty, s\in S} LB_{a(s)} \leq v[\text{DTO}]. \tag{3.36}$$

Since we can only branch on the ϕ-variables finitely often, we have that $\phi^{a(s)}$ is binary-valued for all $s \in S$ sufficiently large. Hence, along the infinite branch, at least one of the p_i- or l_j-variables is chosen as the branching variable infinitely often. Since we bisect the corresponding interval at each step, the length of the interval for a such variable converges to 0. Therefore,

according to Branching Rule A, we have in the limit as $s \to \infty, s \in S$ that $p_i^{*l} = p_i^{*u}, \forall i$, and $l_j^{*l} = l_j^{*u}, \forall j$. By Proposition 3.9, we thus have that x^* is feasible to DTO with objective function value $v^* \equiv \lim_{s\to\infty, s\in S} LB_{a(s)}$, and so, $v^* \geq v[\text{DTO}]$. Together with (3.36), we have that x^* solves DTO with objective function value v^*. □

3.3.2 Branch-and-Bound Algorithm B for Problem DTO

This alternative branch-and-bound procedure is the same as Algorithm A, with the exception of the branching variable selection at each stage s along with the partitioning of the hyperrectangle Ω' associated with each active node $a \in A_s$, where Ω' is defined by (3.33). The following result establishes the global convergence of Algorithm B.

Proposition 3.13. Similar to the main convergence result of Algorithm A stated in Proposition 3.12, Algorithm B (with $\varepsilon = 0$) solves Problem DTO.

Proof. The case of convergence in a finite number of steps is clear. For the case of infinite stages, taking any infinite branch of the tree where the stages s are indexed by a set S, we have as in the proof of Proposition 3.12 that $\{x^{a(s)}, \Omega'^{a(s)}\}_{s\in S} \to \{x^*, \Omega'^*\}$, and that $LB(s) = LB_{a(s)} = v[\text{LP}(\Omega'^{a(s)})] \leq v[\text{DTO}], \forall s \in S$, which also holds true in the limit:

$$v^* = \lim_{s\to\infty, s\in S} LB_{a(s)} \leq v[\text{DTO}]. \tag{3.37}$$

We now show that x^* is feasible to Problem DTO. Along the convergent subsequence, for s large enough, $\phi^{a(s)} = \phi^*$ is binary-valued, and so some p_i-, l_j-, or θ_j-variable is partitioned infinitely often at stages $s \in S_1$, say, where $S_1 \subseteq S$. From Branching Rule B, this variable equals one of its bounds in the limit, and therefore, the discrepancy related to it in (3.34) approaches zero. Since this variable has the maximum discrepancy in (3.34), $\forall s \in S_1$, the discrepancies related to the other p-, l-, and θ-variables also approach zero as $s \to \infty, s \in S_1$. Consequently, by Proposition 3.8, x^* satisfies Constraints (3.12c), and (3.12k) - (3.12n) with $\phi_d^* \in \{0,1\}, \forall d \in D$. Hence, x^* is feasible to DTO with objective value v^*, so that $v^* \geq v[\text{DTO}]$. Together with (3.37), we have that x^* solves DTO with objective value v^*. □

3.3.3 Branch-and-Bound Algorithm C for Problem DTO

As described earlier in the section, this procedure follows the same scheme as that of Algorithms A and B, while adopting a combination of these two methods for selecting a branching variable and partitioning its interval, and including the alternative option of splitting the interval of the selected variable at its geometric mean. As such, its global convergence proof follows identically to that in Propositions 3.12 and 3.13 above, and is omitted for the sake of brevity.

3.4 Computational Results

In this section, we study the effectiveness of the proposed branch-and-bound procedures for solving Problem DTO. We begin by considering the gas-line rupture application illustrated in Figure 3.1, and solve it using the proposed algorithms as well as using the commercial global optimization software BARON, Version 8.1.5 ([84]). We further analyze the sensitivity of the solution with respect to the budget and resource availability restrictions, and also with respect to variations in the logit and loss function parameters.

Next, we explore the best combination of algorithmic features including the various branching variable selection schemes and partitioning strategies. We also investigate the performance of the different proposed linearization methods and techniques for generating tangential supports, and assess the effect of incorporating range reductions and VIs on CPU time. In addition, we examine the direct implementation of the software BARON to solve the original problem formulation DTO (given by (3.5)) as well as its transformed version DTO(Ω) (given by (3.12)), in comparison with our best proposed procedure. Runs with BARON have been made on a remote 1.6 GHz Intel Pentium M processor running Linux (courtesy of N. V. Sahinidis), whereas the proposed procedures have been implemented on a local 2.33 Ghz Intel Pentium Xeon processor running Windows (all runs with BARON in this paper have been made with default settings (with increased memory workspace) and by pre-generating the GAMS problem file prior to inputting this into the BARON solver).

3.4.1 Gas-Line Rupture: Illustrative Example

Consider the scenario in which, in the aftermath of a gas leak (represented by node 0), cascading sequences of probabilistic events and decisions result in consequences as depicted in Figure 3.1. For this specific gas-line rupture application, we assume that five event-related and five consequence-related resources are available to prevent failures and to alleviate consequences. The model coefficients a_{im} and b_{jn} were generated uniformly in the respective intervals [0.05, 0.1] and [100, 500]. Additionally, we set $a_{i0} = ln(0.05/(1-0.05)), \forall i$, and generated $b_{j0}, \forall j$, randomly on $[10000 + l^l_j, 10000 + 2l^l_j]$. The total available event-related and consequence-related resources were set to $s_m = 10, \forall m$ and $t_n = 20, \forall n$, respectively, where the corresponding per-unit costs of allocating resources, c_{im} and d_{jn}, were generated uniformly on [20, 40]. The total available budget for resource allocation was initially taken as $\beta = 3500$. The lower bound on $l_j, \forall j \in \tau$, was determined depending on the scenario produced by the unique path connecting the root node to the leaf node j. More specifically, a contribution to l^l_j of $5000(2.5)^k$ was set to be incurred for each failure event on the path from node 0 to the leaf node, where k is the nodal-distance between the failed event and the leaf node. The lower bound on l_j for any leaf node that has no failure event along the path to it from node 0 was set to 500. The upper bound was determined as $l^{u(3.4)}_j, \forall j \in \tau$, whereas the lower and upper bounds on p_i were set to 0.0001 and $p^{u(3.3)}_i$, respectively, $\forall i \in I_e$. Moreover, the direct cost for each selected decision alternative $d = 1, \ldots, D = 9$ was generated randomly on [100, 200].

Using the above data, we first explored the effect of uncertainty in logit and loss function parameters on the failure probabilities and loss amounts, and hence on the total expected loss in the system. Toward this end, suppose that a policy maker estimates function parameters, $a_{im} = \bar{a}_{im}, \forall i, m$, and $b_{jn} = \bar{b}_{jn}, \forall j, n$, with incomplete information and, accordingly, determines optimal decision alternatives, ϕ^*_{est}, and resource allocations, q^*_{est} and r^*_{est}. Now, assume that there exists up to a $\pm 5\%$ error, say, in estimating the logit and loss function parameters, so that their actual values (uniformly generated) are given by $\bar{a}_{im} \pm 0.05\bar{a}_{im}$ and $\bar{b}_{jn} \pm 0.05\bar{b}_{jn}$, respectively. Consequently, by substituting the implemented policy decisions $(\phi^*_{est}, q^*_{est}, r^*_{est})$ within the model functions represented by the actual parameter values, we can compute the corresponding failure probabilities and loss amounts, yielding an actual realized total expected loss of z_{act}. On the other hand, if these actual parameters had been known *a priori*, we could have obtained the optimal expected loss, z^*_{act}, by applying the proposed algorithm. The resulting percentage optimality gap, $100(z_{act} - z^*_{act})/z^*_{act}$, provides an indication of the sensitivity of the quality of the derived solution

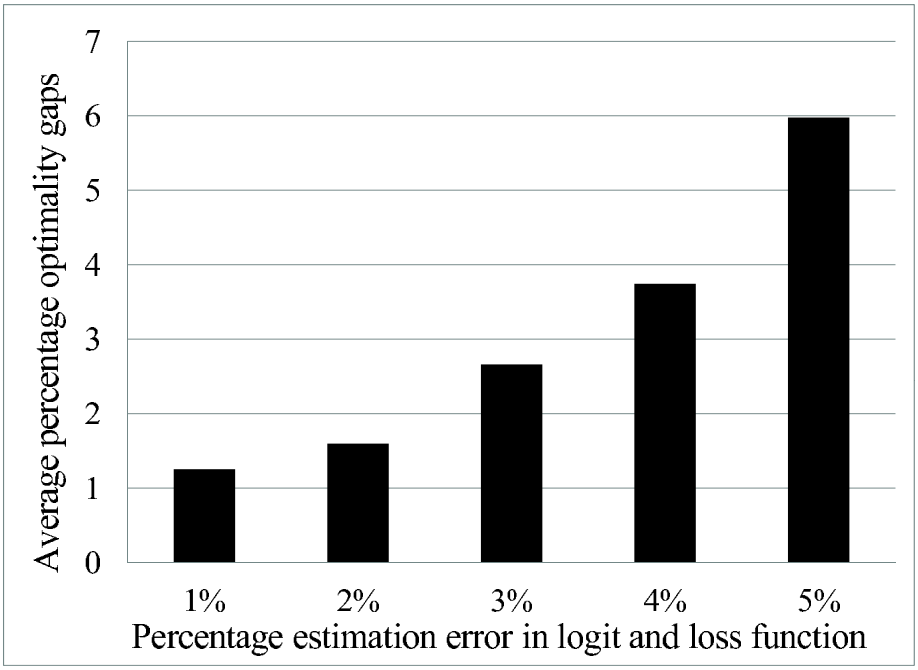


Figure 3.3: Effect of estimation errors in logit and loss function parameters on the expected loss.

to the function parameters, and also reflects on the value of perfect information. We utilized parameter estimation errors of $1\%,\ldots,5\%$, and replicated each case 10 times. Figure 3.3 displays the average percentage optimality gaps obtained for the different uncertainty levels, where the expected loss is in the order of 1775. For error levels of $1\%,\ldots,5\%$ in estimates, the average percentage optimality gaps are $1.3\%, 1.6\%, 2.7\%, 3.7\%$, and 6.0%, respectively. Note that the lower and upper bounds on failure probabilities and the losses at leaf nodes, as well as the magnitudes of the logit and loss function parameters influence this computed percentage optimality gap, and the results shown here are based on reasonable parameter ranges and are meant to serve only as an illustration of the types of studies a decision-maker might undertake to assess the sensitivity of the quality of solutions being considered for implementation with respect to possible errors in parameter estimations.

Next, using five levels of the available budget, $\beta \in \{2000,\ldots,6000\}$, and four levels of event-related resources, $s \in \{20,\ldots,50\}$, we computed the optimal expected loss (objective function value) and the variance of loss (as given by Equation (3.6)). The results are displayed in Figure 3.4. As a point of interest, the minimum expected loss and the associated variance for the case of unlimited resources are depicted by horizontal lines in the corresponding graphs. As expected, the objective function value and the variance of loss improve significantly with initial increments in

budgetary and event-related resources, but further resource increases result in diminishing marginal returns. Note that the exceptional variance increase at (β, s) = (6000, 40) is due to the discrete restructuring of the tree by virtue of ϕ_2 being activated, where prior to this, the decisions ϕ_3 and ϕ_6 were active. Hence, the decline in variance to its limiting value as resources increase need not be monotone due to the discrete nature of the problem.

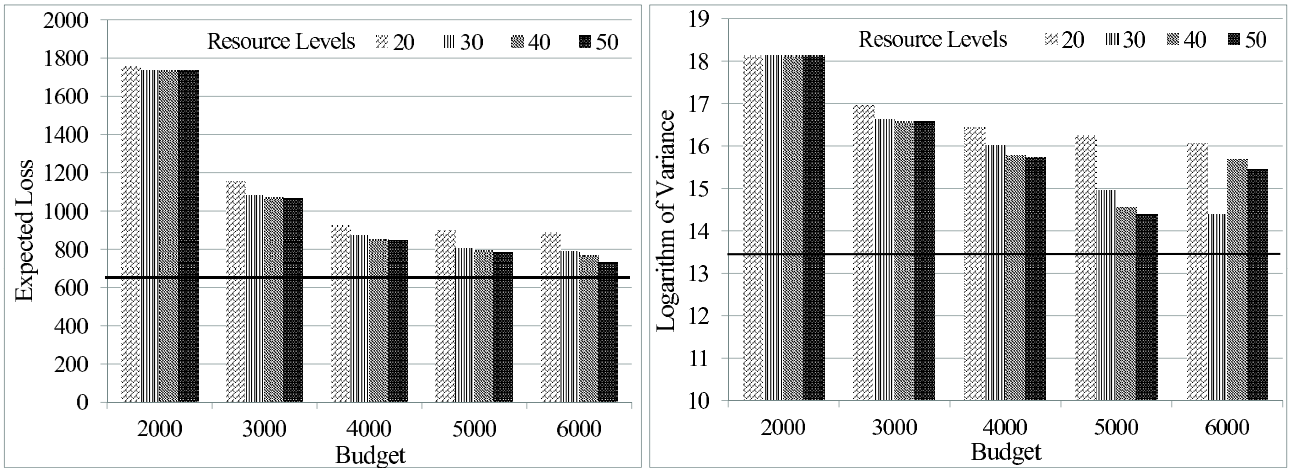


Figure 3.4: Effect of budget and event-related resources on expected loss and variance.

As a point of interest, all of the foregoing problems were solved in about 2 CPU seconds using either the proposed algorithm or BARON.

3.4.2 Random Test Cases

We next generated random decision tree problem instances to assess the effectiveness of the proposed solution techniques as well as that of the global optimization software BARON (Version 8.1.5). The two inputs provided to the random tree generation process are the desired *decision node density*, as defined by the ratio of the number of decision nodes to the total number of nodes, and the size of the tree as measured by its maximal depth. Each tree was accordingly generated by initially constructing nodes 0 and 1 as in Figure 3.1. Then, while sequentially generating additional nodes (including the case of node 1), we first checked whether the node was at the maximum depth, in which case it was labeled as a leaf node. Else, we determined the type (event or decision) of the node randomly, depending on the desired and current ratios of the number of event and decision nodes. If the node turned out to be a decision node, we randomly determined the number of

decision alternatives emanating from it. Finally, the leaf node representing the direct consequence of a failure at node 1 was created. To complete the instance specification, we randomly generated the logit coefficients a_{im} and b_{jn}, the cost coefficients c_{im} and d_{jn}, and the decision costs c_d uniformly over the respective intervals [0.5, 1] and [1000, 5000], [20, 40] and [20, 40], and [2, 4]. We set b_{j0} equal to the sum of the lower bound on l_j and a uniformly generated random variable over the interval [100000, 150000] for the first 10 instances. For the next 10 instances, b_{j0} were generated over the interval [100000, 200000]. The lower bound on l_j was determined as described in Section 3.4.1, using a contribution of $1000(2.5)^k$ for each failed event on the path from the root node to the leaf node, where k is the nodal-distance between the failed event and the leaf node. We set the lower bound on l_j to 10 for any node that has no failure event along the path to it from the root node. The budget and the decision node density were respectively selected as 3000 and 0.18-0.38 (specified desired value = 0.3) for the instances numbered 1-5 and 11-15, and as 1500 and 0.56-0.74 (specified desired value = 0.7) for the instances numbered 6-10 and 16-20. The first 10 instances have a depth of five and are preliminarily used for evaluating the performance of eight algorithmic strategy combinations, whereas the depth of the remaining instances is six. The resulting number of leaf nodes range from 22-42 and 36-106 for the test instances having a maximum depth of five and six, respectively.

In the following runs, we define the **base case approach** as the one that incorporates the linearization scheme LM2 (Section 3.2.1), the valid inequalities VIs (Section 3.2.4), and invokes range reduction (Section 3.2.6), while generating four tangential supports for each functional form $f = ln(\gamma)$ using the BES ($H = 4$) strategy, and utilizes an optimality gap tolerance of $\varepsilon = 0.001$.

Using the base case approach, we first studied the efficiency of the three branching variable selection rules (A, B, and C) together with the three splitting rules (arithmetic mean, geometric mean, and the current optimal value), for a total of eight combinations as displayed in Table 3.1. (Note that Rule A combined with the current optimal value partitioning strategy is not included because the current optimal value could be at the lower or upper bound of the selected branching variable, which would consequently not produce valid child-nodes.) Observe also that Algorithm A is defined by the branching variable selection rule A and the arithmetic mean splitting rule (Combination 1), whereas Algorithm B is defined by the branching variable selection rule B and the current optimal value splitting rule (Combination 5). Algorithm C is (principally) defined by the branching variable selection rule C and the geometric mean splitting rule (Combination 7).

Table 3.1: Performance of eight algorithmic strategy combinations.

	Branching Variable		Problem Instance										
	Selection	Splitting	1	2	3	4	5	6	7	8	9	10	**Average**
1 (Alg. A)	Rule A	Arith.	10.8	150	8.6	31.4	5.2	16.3	11.9	8.8	22.9	15.3	**28.1**
2	Rule A	Geom.	10.4	139.8	8.2	27.4	5.1	16.4	10.7	9	22.6	14.8	**26.4**
3	Rule B	Arith.	11.6	132.8	9.8	71.3	11.3	16.6	11.8	8.8	22.9	14.8	**31.2**
4	Rule B	Geom.	10.9	187.7	12.1	142.8	9.8	16.3	11.4	8.9	22.8	15	**43.8**
5 (Alg. B)	Rule B	Current	14.4	215.2	12.3	124.2	10.6	15.9	11.7	9.1	22.6	14.9	**45.1**
6 (Alg. D)	Rule C	Arith.	10.1	115.1	8.4	26.4	5.1	16.3	11.3	8.8	23	14.8	**23.9**
7 (Alg. C)	Rule C	Geom.	10.4	125	8	25.7	5	16.3	10.7	8.9	22.8	14.6	**24.7**
8	Rule C	Current	10.5	157.5	8.2	27.5	4.9	16.6	11.4	8.8	22.6	15	**28.3**

The branching variable selection rule C and the arithmetic mean splitting rule outperformed the other combinations, and therefore this composite strategy was used for further evaluating various algorithmic variants as discussed next. (Algorithm C, defined by Combination 7, comes a close second, and is also further evaluated in subsequent runs. In fact, with BES($H = 5$) used in lieu of BES($H = 4$), the mean CPU times for Combinations 6 and 7 were 26.4 and 26.0 CPU seconds, respectively.)

We next performed a series of preliminary experiments in which one of the features in the base case was varied to ascertain the effects of the different implementationary strategies. Tables 3.2-3.6 provide results for the following experimental studies:

- Table 3.2: Comparison of linearization methods LM1 versus LM2.
- Table 3.3: Effect of the number of tangential supports: BES with $\varepsilon = 0.01, 0.05, 0.1$, and 0.2 versus BES($H$) with $H = 4, 5, 10$, and 20.
- Table 3.4: As in Remark 3.5 in Section 3.2.2, the y_1-, y_2-, and z-variables tend to be at their lower/upper bounds at optimality. Hence, we tested generating only H=2 tangential supports at the two interval end-points for these variables, but used BES(H=4) for the remaining variables.
- Table 3.5: Effect of implementing range reductions.
- Table 3.6: Effect of incorporating the valid inequalities VIs (3.29) of Proposition 3.10 and conducting the related tests (3.30) of Proposition 3.11.

Table 3.2: Performance of the two proposed linearization methods LM1 and LM2.

	LM1		LM2	
Problem Instance	CPU time	# of nodes explored	CPU time	# of nodes explored
1	9.6	57	10.1	57
2	119.5	587	115.1	587
3	8.3	63	8.4	65
4	26.8	201	26.4	201
5	5.3	45	5.1	41
Average (1-5)	**33.9**	**191**	**33**	**190**
6	15.6	145	16.3	151
7	10.4	107	11.3	105
8	17.5	113	8.8	57
9	27.4	175	23	155
10	14.7	97	14.8	97
Average (6-10)	**17.1**	**127**	**14.8**	**113**
Overall Avg.	**25.5**	**159**	**23.9**	**152**

Table 3.3: Effect of the number and placement of tangential supports on CPU time.

Problem	BES							
Instance	(H=4)	(H=5)	(H=10)	(H=20)	(ε=0.01)	(ε=0.05)	(ε=0.1)	(ε=0.2)
1	10.1	9.7	11.8	18.5	21.9	13.6	12.2	11.0
2	115.1	143.1	125.8	192.5	161.2	145.7	155.5	289.6
3	8.4	7.5	9.5	14.6	15.9	10.8	9.4	9.6
4	26.4	28.5	34.3	49.2	44.8	35.0	35.2	39.4
5	5.1	5.3	6.2	9.1	8.9	6.0	6.6	7.1
6	16.3	11.0	20.2	30.5	27.5	16.9	16.7	15.9
7	11.3	11.1	13.9	21.1	20.7	14.1	13.2	12.4
8	8.8	6.5	9.8	11.6	15.7	9.3	9.9	10.7
9	23.0	25.0	24.5	40.9	35.2	26.6	22.8	25.6
10	14.8	15.8	21.8	31.7	33.5	18.7	17.5	18.6
Average	**23.9**	**26.4**	**27.8**	**42**	**38.5**	**29.7**	**29.9**	**44**

The results displayed in Tables 3.2-3.6 indicated that utilizing the linearization method LM2, generating $H = 4$ tangential supports via BES($H = 4$), and invoking range reduction and valid inequalities outperformed other alternatives (the use of VIs reduced the CPU time by 13% over the same strategies without VIs). Note that the relative efficiency of the linearization methods depends on the decision node density. Whereas both linearization schemes were comparable for the first five instances having a low decision node density (LM2 consumed 2.5% lesser effort), we observed a 13% improvement in the CPU time for the next five instances having a high decision

Table 3.4: Two versus four tangential supports for the y_1-, y_2-, and ξ-variables.

	Two tangential supports		BES (H=4)	
Problem instance	CPU time	# of nodes explored	CPU time	# of nodes explored
1	9.6	57	10.1	57
2	108.1	587	115.1	587
3	8.3	65	8.4	65
4	26.0	201	26.4	201
5	4.8	41	5.1	41
6	15.6	145	16.3	151
7	10.8	103	11.3	105
8	9.7	61	8.8	57
9	21.4	155	23.0	155
10	15.2	97	14.8	97
Average	**23**	**151**	**23.9**	**152**

Table 3.5: Effect of the range reduction strategy.

	With range reduction		Without [a] range reduction		
Problem instance	CPU time	# of Nodes explored	CPU time	# of Nodes explored	Optimality [b] Gap (%)
1	9.6	57	226.4	2000	3.2
2	108.1	587	209.2	2000	10
3	8.3	65	215.8	2000	6.1
4	26.0	201	194.8	2000	4.1
5	4.8	41	212	2000	6.3
6	15.6	145	169.9	2000	1.1
7	10.8	103	169.5	2000	2.7
8	9.7	61	223	2000	0.6
9	21.4	155	202.9	2000	11.8
10	15.2	97	262.5	2000	0.6
Average	**23**	**151**	**208.6**	**2000**	**4.7**

[a] Algorithm terminated with the maximum limiting number of explored nodes without satisfying the specified optimality gap restriction.

[b] We formulated the optimality gap as $\frac{(UB-LB)}{UB}$ where UB and LB are upper and lower bounds on objective function value.

node density when using LM2 versus LM1. This is to be expected because higher decision node density instances involve more complex binary sub-structures in the model, which therefore benefit more substantially by using the improved representation LM2 over LM1. We use these settings in further runs with our proposed algorithm, hereinafter referred to as **Algorithm D** (essentially, Combination 6 in Table 3.1). We also explore the efficiency of Combination 7 (**Algorithm C**) along with Algorithm D since the difference in performance between these two procedures is not

Table 3.6: Effect of implementing the proposed valid inequalities VIs.

	With VIs		Without VIs	
Problem instance	CPU time	# of Nodes explored	CPU time	# of Nodes explored
1	9.6	57	7.9	43
2	108.1	587	153.2	587
3	8.3	65	5.7	47
4	26.0	201	14.6	89
5	4.8	41	2.8	21
6	15.6	145	7.5	45
7	10.8	103	11.3	81
8	9.7	61	10.7	53
9	21.4	155	29.3	155
10	15.2	97	21.5	77
Average	**23**	**151**	**26.4**	**120**

significant as displayed in Table 3.1.

The decision tree optimization problem can alternatively be directly solved by applying off-the-shelf commercial software to either the original model (3.5) or to the better-structured equivalently transformed formulation (3.12). The global optimization software BARON was therefore utilized to solve problem instances modeled alternatively as in (3.5) or (3.12) using LM2, as well as to solve problems modeled via (3.12) along with both LM2 and the proposed VIs (3.29), in order to assess the direct solvability of the enhanced models using standard software. Table 3.7 displays the results obtained in comparison with Algorithm D. The optimality gap for each case is computed as $(UB-LB)/UB$, where UB and LB are the respective upper and lower bounds obtained on the optimal objective function value at termination. Using the original DTO formulation (3.5), BARON terminated prematurely for two of the 20 test problems (with an average optimality gap of 29.8%), whereas the proposed algorithm optimized all the problems within the specified CPU time limit of 1800 seconds. Furthermore, BARON terminated with suboptimal solutions within the time limit for Instances 1, 3, and 17, with respective optimality gaps of 2.4%, 1.8%, and 1.4%. On average, it took 293 CPU seconds for BARON to solve Problem DTO (including the early termination cases), while the same set of problems were optimized in 29.7 CPU seconds with the proposed algorithm (yielding an 89.9% savings in effort). Excluding the early termination cases, the average CPU times were 122.4 seconds for BARON versus 27.2 seconds for the proposed algorithm (yielding a 77.8% savings in effort). As a point of interest, Algorithm C solved these instances in 30.6 CPU seconds on average, where this value decreased to 27.6 CPU seconds when the premature

Table 3.7: Results for BARON using the original and the transformed DTO formulations, versus Algorithm D.

	BARON: Original DTO (3.5)		BARON:Transformed DTO (3.12) with LM2		BARON:Transformed DTO (3.12) with LM2 and VIs (3.29)		Proposed Algorithm D	
Problem instance	CPU time	Optimality Gap (%)	CPU time	Optimality Gap (%)	CPU time	Optimality Gap (%)	CPU time	Optimality Gap (%)
1	**15.9**	**2.4***	**1800**	**50.2**	99.5	0	9.6	0
2	**1800**	**30.2**	**1800**	**59.7**	**1800**	**61.1**	108.1	0
3	**2.6**	**1.8***	**1800**	**29.4**	481.7	0	8.3	0
4	25.8	0	**1800**	**59.1**	512.2	0	26.0	0
5	4.3	0	**1800**	**19.7**	56.2	0	4.8	0
Avg.	**369.7**		**1800**		**589.9**		**31.4**	
6	29.5	0	**1800**	**29.5**	238.7	0	15.6	0
7	4.3	0	**1800**	**36.3**	166.6	0	10.8	0
8	293.0	0	**1800**	**27.9**	125.7	0	9.7	0
9	130.9	0	**1800**	**50.0**	**1800**	**48.6**	21.4	0
10	71	0	**1800**	**38.5**	172.4	0	15.2	0
Avg.	**105.8**		**1800**		**500.7**		**14.5**	
11	28	0	**1800**	**50**	483.4	0	22.1	0
12	17.4	0	1366	0	429.7	0	16.2	0
13	395.4	0	**1800**	**46.8**	**1800**	**22.8**	30.1	0
14	24	0	**1800**	**62.3**	**1800**	**24.1**	109.7	0
15	**1800**	**29.4**	**1800**	**62.2**	**1800**	**n/a**	23.4	0
Avg.	**453**		**1713.2**		**1262.6**		**40.3**	
16	541.6	0	**1800**	**26.6**	**1800**	**17.5**	81	0
17	**406.1**	**1.4***	**1800**	**44.9**	1454.9	0	37.3	0
18	78.5	0	1381	0	474.9	0	20.9	0
19	180.2	0	**1800**	**100**	340	0	17.2	0
20	12.1	0	403.6	0	58.4	0	6.7	0
Avg.	**243.7**		**1436.9**		**825.7**		**32.6**	

*Terminated with a suboptimal solution having the specified optimality gap with respect to the optimum found by Algorithm D.

termination cases are excluded. Moreover, BARON solved only 3 out of the 20 problem instances to optimality using the transformed DTO formulation (3.12) with LM2, which may be due to the increase in the number of variables as well as the number of nonlinear constraints. However, when we further invoked the valid inequalities VIs within the formulation (3.12), BARON managed to solve 14 out of the 20 problem instances using this enhanced model representation formulation.

Chapter 4

Enhancing RLT-based Relaxations for Polynomial Programming Problems via a New Class of v-Semidefinite Cuts

In this chapter, we propose to enhance Reformulation-Linearization Technique (RLT)-based linear programming (LP) relaxations for polynomial programming problems by developing cutting plane strategies using concepts derived from semidefinite programming. Given an RLT relaxation, we impose positive semidefiniteness on suitable dyadic variable-product matrices, and correspondingly derive implied semidefinite cuts. In the case of polynomial programs, there are several possible variants for selecting such particular variable-product matrices for imposing the positive semidefiniteness restrictions in order to derive implied valid inequalities, which leads to a new class of cutting planes that we call *v-semidefinite cuts*. We explore various strategies for generating such cuts, and exhibit their relative effectiveness towards tightening the RLT relaxations and solving the underlying polynomial programming problems in conjunction with an RLT-based branch-and-cut scheme, using a test-bed of randomly generated instances as well as standard problems from the literature. Our results demonstrate that these cutting planes achieve a significant tightening of the lower bound in contrast with using RLT as a stand-alone approach, thereby enabling an appreciable reduction in the overall computational effort, even in comparison with the commercial software BARON.

4.1 Introduction

The *Reformulation-Linearization Technique* (**RLT**) is a unifying approach for solving wide classes of discrete as well as continuous factorable nonconvex optimization problems via a process of deriving tight effective relaxations through the generation of suitable valid inequalities in a higher dimensional space. In keeping with this underlying spirit, the RLT methodology can be further enhanced by incorporating additional valid inequalities that are obtained via *semidefinite programming* **(SDP)** constructs (see Anstreicher [5], Burer and Vandenbussche [13, 14], Helmberg [29], Kojima and Tunçel [38], Lasserre [41–43], Sherali and Fraticelli [70], and Vandenberghe and Boyd [86]). In this chapter, we propose a new class of cuts called *v-semidefinite* **(*v*-SDP)** cuts that are generated by incorporating suitable SDP restrictions within the RLT framework. For the sake of illustration, we describe the mechanism for deriving such cuts and apply them to solve the challenging class of polynomial programming problems, although the procedure can be readily extended to solve the more general class of factorable programs by using the augmented approach discussed in Sherali and Wang [75].

Hence, consider the generic formulation of a polynomial program, **PP**(Ω), predicated on a bounding hyperrectangle Ω:

$$\begin{aligned}
\textbf{PP}(\Omega)\textbf{:}\quad & \text{Minimize} \quad \{\phi_0(x) : x \in Z \cap \Omega\} \\
& \text{where} \quad Z = \{x : \phi_r(x) \geq \beta_r, \text{ for } r = 1,\ldots,R_1,\ \ \phi_r(x) = \beta_r, \text{ for } r = R_1+1,\ldots,R\}, \\
& \qquad\quad \Omega = \{x : 0 \leq l_j \leq x_j \leq u_j < \infty,\ \forall j = 1,\ldots,n\},
\end{aligned}$$

where

$$\phi_r(x) \equiv \sum_{t \in T_r} \alpha_{rt} \left[\prod_{j \in J_{rt}} x_j \right], \text{ for } r = 0,\ldots,R.$$

Here, T_r is an index set for the terms defining $\phi_r(.)$, and α_{rt} are real coefficients for the monomial terms $\prod_{j \in J_{rt}} x_j$, $t \in T_r$, $r = 0,\ldots,R$, where each J_{rt} is a multi-set that might contain a repetition of indices from $\mathcal{N} \equiv \{1,\ldots,n\}$. For example, if $J_{rt} = \{1,2,2,3\}$, then the corresponding monomial term is $x_1 x_2^2 x_3$. Denoting $\mathcal{N}^\delta \equiv \{\mathcal{N},\ldots,\mathcal{N}\}$ to be composed of δ replicates of $\mathcal{N}$, where δ is the maximum degree of the polynomial functions defining PP(Ω), we have that each $J_{rt} \subseteq \mathcal{N}^\delta$, with $1 \leq |J_{rt}| \leq \delta$, $\forall t \in T_r$, $r = 0,\ldots,R$. For future reference, we also define $\Delta \equiv \lfloor \delta/2 \rfloor$.

Sherali and Tuncbilek [71] describe an RLT methodology for constructing tight linear pro-

gramming relaxations that are embedded within a suitable branch-and-bound algorithm for solving PP(Ω) to global optimality. This procedure essentially involves the partitioning of the original set Ω into sub-hyperrectangles, each of which is associated with a node of the branch-and-bound tree. Let $\Omega' \equiv \{x : l'_j \leq x_j \leq u'_j, \forall j \in \mathscr{N}\} \subseteq \Omega$ be any such partition, and denote as LP(Ω') the corresponding linear programming RLT relaxation that is generated for this associated node subproblem. Given Ω', in order to construct the linear programming bounding problem LP(Ω') using RLT, implied *bound-factor product constraints* are generated by using products of the *bound-factors*, $(x_j - l'_j) \geq 0$ and $(u'_j - x_j) \geq 0$, $\forall j \in \mathscr{N}$, taken δ at a time. These constraints can be expressed as follows:

$$F_\delta(J_1, J_2) \equiv \prod_{j \in J_1} (x_j - l'_j) \prod_{j \in J_2} (u'_j - x_j) \geq 0, \forall (J_1 \cup J_2) \subseteq \mathscr{N}^\delta, |J_1 \cup J_2| = \delta. \tag{4.1}$$

After including the constraints (4.1) in the problem PP(Ω'), the following substitution is applied to linearize the resulting problem:

$$X_J = \prod_{j \in J} x_j, \forall J \subseteq \mathscr{N}^\delta, \tag{4.2}$$

where the indices in J are assumed to be sequenced in nondecreasing order, and where $X_{\{j\}} \equiv x_j$, $\forall j \in \mathscr{N}$, and $X_\emptyset \equiv 1$. Note that each distinct multi-set J produces a distinct variable X_J. For notational convenience, define the lifted-space RLT variable vector $X \equiv \{X_J, J \subseteq \mathscr{N}^\delta, 2 \leq |J| \leq \delta\}$, and let $[.]_L$ represent the linearization of any expression or constraint set $[.]$ under the substitution (4.2). This yields LP(Ω') as specified below:

$$\begin{aligned}
\textbf{LP}(\Omega')\textbf{:} \quad & \text{Minimize} \quad \{[\phi_0(x)]_L : (x,X) \in Z_L \cap \Omega'\} \\
& \text{where} \quad Z_L = \{(x,X) : [\phi_r(x)]_L \geq \beta_r, \text{ for } r = 1, \ldots, R_1, \\
& \qquad\qquad [\phi_r(x)]_L = \beta_r, \text{ for } r = R_1+1, \ldots, R, \ [(4.1)]_L\}, \\
& \qquad \Omega' = \{x : 0 \leq l_j \leq l'_j \leq x_j \leq u'_j \leq u_j < \infty, \ \forall j = 1, \ldots, n\}.
\end{aligned} \tag{4.3}$$

Note that as discussed by Sherali and Tuncbilek [71], LP(Ω') can be made to advantageously include additional *bound-constraint-factor product constraints* composed by deriving valid polynomial inequalities of order δ through products of bound-factors and similar more general constraint-factors, and then linearizing using (4.2). Notwithstanding such augmentations of LP(Ω'), we endeavor to tighten this linear programming relaxation in the present research effort by generating valid inequalities or cutting planes that are prompted by SDP considerations.

To illustrate the underlying concept, consider a quadratic polynomial programming problem. Note that the new RLT variables in this context are represented by the $n \times n$ matrix $M_0 \equiv [xx^T]_L$, where $[.]_L$ represents the linearization of the expression $[.]$ under the substitution:

$$X_{ij} = x_i x_j, \forall i, j = 1, \ldots, n, \; i \leq j. \tag{4.4}$$

Observe that, since $[xx^T]$ is *symmetric and positive semidefinite* (denoted $\succeq 0$), we can require that $M_0 \succeq 0$, as opposed to simply enforcing nonnegativity on this matrix. Then, we have that:

$$M_0 \succeq 0 \Leftrightarrow \alpha^T M_0 \alpha = [(\alpha^T x)^2]_L \geq 0, \forall \alpha \in \mathbb{R}^n, \|\alpha\| = 1. \tag{4.5}$$

In fact, a stronger implication in this same vein is obtained by considering

$$x_{(1)} = \begin{bmatrix} 1 \\ x \end{bmatrix}, \text{ and defining the matrix } M_1 \equiv [x_{(1)} x_{(1)}^T]_L = \begin{bmatrix} 1 & x^T \\ x & M_0 \end{bmatrix}, \tag{4.6}$$

and requiring that $M_1 \succeq 0$.

In lieu of solving the resulting semidefinite programming relaxations, which would detract from the robustness and efficiency of the algorithm (especially as the size of the problem increases), Sherali and Fraticelli [70] recommended utilizing linear programming relaxations enhanced with a class of RLT constraints known as *semidefinite cut*. Accordingly, given a solution $(\bar{x}, \bar{X})$ to the RLT-based LP relaxation, Sherali and Fraticelli [70] developed an algorithm based on superdiagonalization process (see Bazaraa et al. [10]) that examines whether $\bar{M} \succeq 0$, where $\bar{M}$ evaluates M at $(\bar{x}, \bar{X})$. This algorithm automatically generates an $\bar{\alpha}$, in case $\bar{M}$ is not positive semidefinite, and yields the following semidefinite cut:

$$\bar{\alpha}^T M \bar{\alpha} = [(\bar{\alpha}^T x)^2]_L \geq 0. \tag{4.7}$$

Motivated by the approach and experience reported in Sherali and Fraticelli [70], and based on the theoretical results embodied by the semidefinite programming relaxations of Lasserre [41, 43] as presented in (2.4), (2.5), and (2.6), the idea we adopt in the present study is to replace the linear matrix inequalities in these problems by associated generalizations of the semidefinite cuts of the type (4.7). In cases where even the dimension of the matrices in these LMIs might be too large to be computationally manageable, we could conceivably generate semidefinite cuts based on

the relationship $[vv^T]_L \geq 0$, where v is a vector of suitable monomial terms involving certain key x-variables alone. This concept constitutes the underlying theme of the present study, and gives rise to several proposed variants of the resulting class of cuts that we refer to as *v-semidefinite cuts*.

The remainder of this chapter is organized as follows. Section 4.2 begins with a description of the SDP cut derived in Sherali and Fraticelli [70] as well as proposes a new cut generation routine using restricted eigenvalue-eigenvector computations, and then proceeds to motivate the class of v-semidefinite cuts via an illustrative example. Next, Section 4.3 explores the generation of several such cuts based on appropriately composed vectors v, and Section 4.4 illustrates these cuts using an example. Extensive computational experience is reported in Section 4.5 using randomly generated test cases, as well as standard polynomial programming problems from the literature that arise in various applications.

4.2 Cut Generation Routines

We begin this section by summarizing the semidefinite cut generation algorithm proposed by Sherali and Fraticelli [70]. Next, in Section 4.2.2, we present a modified restricted cut generation routine that utilizes eigenvalues and eigenvectors. We shall utilize these routines for deriving the proposed v-semidefinite cuts later in Section 4.3 for solving polynomial programs. For generality in discussion, we state these procedures in the context of deriving valid inequalities implied by $M \succeq 0$, where M is some $\bar{n} \times \bar{n}$ symmetric matrix associated with an LMI under consideration. Essentially, we are interested in determining a suitable $\bar{\alpha} \in \mathbb{R}^{\bar{n}}$ such that a semidefinite cut of the type (4.7) given by $\bar{\alpha}^T M \bar{\alpha} \geq 0$ can be imposed at the solution to the current relaxation.

4.2.1 Cut Generation Routine of Sherali and Fraticelli

Given a solution $(\bar{x}, \bar{X})$ to the underlying RLT relaxation for any nonconvex polynomial program, consider the application of the *superdiagonalization algorithm* (see Bazaraa et al. [10]) for checking the positive semidefiniteness of the symmetric matrix $\bar{M}$, where $\bar{M}$ evaluates M at the solution $(\bar{x}, \bar{X})$. In this process, starting with $G^1 \equiv \bar{M}$, suppose that we are currently examining the reduced submatrix, $G^i \in \mathbb{R}^{(\bar{n}-i+1)\times(\bar{n}-i+1)}$, $1 \leq i \leq \bar{n}$, appearing in rows and columns $i, \ldots, \bar{n}$ of the partially

diagonalized matrix, where G^i has the following partitioned form with the first row and column explicitly shown:

$$G^i \equiv \begin{bmatrix} G^i_{11} & (g^i)^T \\ g^i & G \end{bmatrix}. \tag{4.8}$$

The cutting plane generation strategy (abbreviated **SF**) then proceeds as follows:

Cut generation routine SF:

Initialization: Put i=1.

Step 1: Given G^i as in (4.8) if $G^i_{11} < 0$, then set

$$\alpha^i = (\alpha_i, \ldots, \alpha_{\bar{n}})^T = (1, 0, \ldots, 0)^T \in \mathbb{R}^{\bar{n}-i+1}$$

and go to Step 8. Else, go to Step 2.

Step 2: If $i = \bar{n}$, then $\bar{M} \succeq 0$; stop. Else, go to Step 3.

Step 3: If $G^i_{11} = 0$, go to Step 4. Otherwise, go to Step 6.

Step 4: If $G^i_{1j} = 0, \forall j$, go to Step 5. Else, select that $j \in \{2, \ldots, \bar{n}-i+1\}$ for which $G^i_{1j} = G^i_{j1} = \theta \neq 0$, and which is such that with $\phi \equiv G^i_{jj}$, we obtain the smallest resulting value of $\lambda = \frac{\phi - \sqrt{\phi^2+4\theta^2}}{2}$. Accordingly, then, set

$$\alpha^i = (\alpha_i, 0, 0, \ldots, \alpha_{i+j-1}, 0, \ldots, 0)^T \in \mathbb{R}^{\bar{n}-i+1}$$

where $\alpha_i = \frac{1}{\sqrt{1+\frac{\lambda^2}{\theta^2}}}$ and $\alpha_{i+j-1} = \frac{\alpha_i \lambda}{\theta}$. Go to Step 8.

Step 5: Store $G^i_{11} = 0$, and let G^{i+1} be obtained by deleting the first row and column of G^i. Go to Step 7.

Step 6: Store $\begin{pmatrix} G^i_{11} \\ g^i \end{pmatrix}$ and compute $G^{i+1} = G - \frac{g^i (g^i)^T}{G^i_{11}}$. Go to Step 7.

Step 7: Increment i by 1 and return to Step 1.

Step 8: Check if $i > 1$. If yes, go to Step 9. Else, go to Step 11.

Step 9: Recursively for $\gamma = i-1$ to 1, compute α_γ and α^γ as follows: If $G^\gamma_{11} = 0$, then set $\alpha_\gamma = 0$; otherwise, compute $\alpha_\gamma = \frac{-(\alpha^{\gamma+1})^T g^\gamma}{G^\gamma_{11}}$, where $\alpha^{\gamma+1} \equiv (\alpha_{\gamma+1}, \ldots, \alpha_{\bar{n}})^T$, and set $\alpha^\gamma \equiv$

$\begin{bmatrix} \alpha_\gamma \\ \alpha^{\gamma+1} \end{bmatrix}$. Go to Step 10.

Step 10: Normalize α^1 and go to Step 11.

Step 11: Generate a new SDP cut $\alpha^T M\alpha \geq 0$ using $\alpha \equiv \alpha^1$, and append this to the current relaxation.

The foregoing approach establishes an inductive polynomial-time process (of worst-case complexity $O(\bar{n}^3)$) for generating an SDP cut.

Remark 4.1. Note that, in the above cut generation routine, the matrix $M \equiv [vv^T]_L$ can be initially permuted based on a suitable re-arrangement of the elements of v in order to take advantage of the sequential nature of the routine. For example, in the quadratic case, Sherali and Fraticelli [70] analyze look-ahead, diagonal-sort, and full permutation re-arrangement strategies of this type. In the *look-ahead* scheme, instead of examining just the leading diagonal element of G^i at Step 1 to check if $G^i_{11} < 0$ for any given $i \in \{1,\ldots,\bar{n}\}$, all of the diagonal elements are scanned and a cut is generated for each violation of positive semidefiniteness. In another variant, called *diagonal-sort*, the diagonal elements of the matrix $\bar{M}$ are pre-sorted in nonincreasing order, with $\bar{M}$ rearranged accordingly, and the cut generation routine SF is then applied to this permuted matrix. In a third variant called *full permutation*, whenever the leading diagonal element of G^i violates semidefiniteness (is negative) at Step 1, the cut generation routine pauses to permute the rows and columns of G^i so that the most positive diagonal element of G^i becomes the leading element, and the algorithmic process is then continued. The performances of these three variants are extensively tested for different matrices M in Section 4.5. □

4.2.2 Cut Generation Routine Using Restricted Eigenvalue and Eigenvector Computations

As an alternative, we now propose a cut generation routine using restricted eigenvalue and eigenvector computations. Our motivation is that whereas SDP cuts $\alpha^T M\alpha \geq 0$ derived using α-vectors given by the (normalized) eigenvectors corresponding to the most negative eigenvalues tend to be tight, the eigenvalue-eigenvector computations become arduous as the size of the matrix increases.

To benefit from the tightness of these cuts while curtailing the accompanying computational burden, we examine suitable submatrices G^i of $\bar{M}$ that are limited in size to 10×10 and compute eigenvalues and eigenvectors of such matrices in lieu of utilizing the full original $\bar{n} \times \bar{n}$ matrix $\bar{M}$. (We assume here that $\bar{n} > 10$; else, we directly employ $\bar{M}$ itself.) Starting with $\bar{M}$ (which is perhaps appropriately permuted as discussed in Remark 4.1), we examine positive semidefiniteness on a sequence of overlapping block diagonal submatrices G^i of $\bar{M}$ of size 10×10 by shifting the diagonal position *five elements* at a time, where the final shift involves up to five elements. Given a certain maximum number, K, of cuts to be thus generated, the following routine (abbreviated **EV-10**) derives the required α-vectors.

Cut generation routine EV-10:

Initialization: Determine the number of submatrices to be considered, given by $m = \lceil (\bar{n})/5 \rceil - 1$, and the maximum number of cuts to be generated for each submatrix, given by $K' = \lceil K/m \rceil$. Set $i = 1$ and $j = 1$.

Step 1: Extract the submatrix G^i appearing in rows and columns $j, \ldots, j+9$, of $\bar{M}$ and go to Step 2.

Step 2: Calculate the eigenvalues and eigenvectors of G^i. Set $k_i = 1$ and go to Step 3.

Step 3: If $k_i \leq K'$, go to Step 4. Else, increment i by 1 and go to Step 6.

Step 4: If the k_i^{th} smallest eigenvalue of G^i is negative, go to Step 5. Else, increment i by 1 and go to Step 6.

Step 5: Generate $\alpha \in \mathbb{R}^{\bar{n}}$ having elements $j, \ldots, j+9$ set equal to the corresponding normalized eigenvector and with the remaining elements set equal to zero. Insert this α-vector into the current list of cut-generation vectors in nondecreasing order of its associated eigenvalue. Increment k_i by 1 and go to Step 3.

Step 6: If $i \leq m$, set $j = \min\{5i-4, \bar{n}-9\}$ and go to Step 1. Else, go to Step 7.

Step 7: Generate the SDP cuts $\alpha^T M \alpha \geq 0$ using (up to) the first K α-vectors in the cut-generation list and append these cuts to the current relaxation.

For benchmark purposes, in a routine abbreviated **EV**, we compute the normalized eigenvectors of the original $\bar{n} \times \bar{n}$ matrix $\bar{M}$, and generate up to K SDP cuts $\alpha^T M \alpha \geq 0$, where the vectors α are the normalized eigenvectors corresponding to the (up to) K most negative eigenvalues of $\bar{M}$.

There are potentially several cuts that can be generated based on different matrix compositions and α-vector computations of this type. To motivate further discussion, consider the following illustrative example.

4.2.3 Motivation for the Class of Semidefinite Cuts: An Illustration

Consider the following pair of problems (Lasserre [43]):

P1: Minimize $\{x - x^2 : 0 \leq x \leq 1\}$ and **P2:** Minimize $\{x^2 - x : 0 \leq x \leq 1\}$.

For each of P1 and P2, the RLT relaxation (4.3) using δ=2 and $y \equiv [x^2]_L$ is given by LP1 and LP2, respectively, as stated below:

$$\begin{array}{lll} \textbf{LP1:} & \text{Minimize} & z = x - y \\ & \text{subject to} & [x^2]_L = y \geq 0 \\ & & [(1-x)^2]_L = 1 + y - 2x \geq 0 \\ & & [x(1-x)]_L = x - y \geq 0 \\ & & 0 \leq x \leq 1. \end{array} \tag{4.9a}$$

$$\begin{array}{lll} \textbf{LP2:} & \text{Minimize} & z = y - x \\ & \text{subject to} & [x^2]_L = y \geq 0 \\ & & [(1-x)^2]_L = 1 + y - 2x \geq 0 \\ & & [x(1-x)]_L = x - y \geq 0 \\ & & 0 \leq x \leq 1. \end{array} \tag{4.9b}$$

Note that LP1 yields an optimum of $z = 0$ at $x = y = 0$, which is therefore an optimum to P1 (an alternative optimum has $x = y = 1$). However, the convex program P2 has an interior optimum given by $x = 1/2$ with an objective value of $-1/4$. The relaxation LP2 yields the solution

$x = 1/2, y = 0$, with an objective value of $z = -1/2 < -1/4$.

Now, suppose that in the spirit of (4.6), we impose

$$M_1 \equiv \begin{bmatrix} 1 & x \\ x & y \end{bmatrix} \succeq 0, \tag{4.10a}$$

i.e.,

$$\alpha_1^2 + \alpha_2^2 y + 2\alpha_1\alpha_2 x \geq 0, \forall(\alpha_1, \alpha_2). \tag{4.10b}$$

Graphically, considering the case of $\alpha_1 > 0$ and $\alpha_2 < 0$ for the sake of illustration, we have the situation depicted in Figure 4.1. So long as

$$\frac{-\alpha_1}{2\alpha_2} \geq \frac{\alpha_1^2}{\alpha_2^2}, \tag{4.11}$$

the revised optimum to LP2 after incorporating the cut is characterized by the intersection of $1 + y - 2x = 0$ and the cut. The coordinates of this solution are given by

$$x = \frac{\alpha_2^2 - \alpha_1^2}{2\alpha_2(\alpha_2 + \alpha_1)} = \frac{\alpha_2 - \alpha_1}{2\alpha_2} \text{ and } y = 2x - 1. \tag{4.12}$$

The corresponding revised lower bounding LP objective value is then given by

$$y - x = x - 1 = \frac{-(\alpha_1 + \alpha_2)}{2\alpha_2}. \tag{4.13}$$

Note that if we normalize the cut by taking $\alpha_1 \equiv 1$, the condition (4.11) and $\alpha_2 < 0$, along with (4.13) yields

$$\alpha_2 \leq -2, \text{ with the LP objective value } = \frac{-(1 + \alpha_2)}{2\alpha_2}. \tag{4.14}$$

The greatest LP bound (of value $-1/4$) is achieved when $\alpha_2 = -2$. In this case, the cut is given by:

$$1 + 4y - 4x \geq 0, \tag{4.15}$$

and the revised LP optimum is given by $(x, y) = (3/4, 1/2)$ according to (4.12), but also by the (desired) non-extremal solution $(1/2, 1/4)$, with objective value equal to $-1/4$. In particular, the solution $x = 1/2$ thus solves P2 with objective value equal to $-1/4$. Note that as $\alpha_2 \to -\infty$, the LP bound value in (4.14) approaches $-1/2$, which was the lower bound obtained by the original

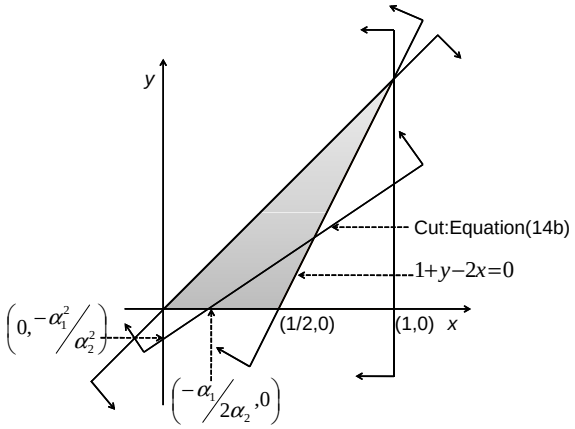


Figure 4.1: The feasible region for Problem LP2 and the associated semidefinite cut.

relaxation, LP2.

Eigenvalue-Eigenvector strategy applied to M_1

The SDP cut generation strategy of Konno et al. [40], or using Proposition 4 of Sherali and Fraticelli [70], determines the cut (4.10b), where α is derived via the problem:

$$\text{Minimize } \{\alpha^T \bar{M}_1 \alpha : \|\alpha^2\| = 1\}, \text{ where } \bar{M}_1 = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 0 \end{bmatrix}. \tag{4.16}$$

The solution to (4.16) yields

$$(\alpha_1, \alpha_2) = (0.38268, -0.92388), \text{ with objective function value } = -0.207107. \tag{4.17}$$

Note that the obtained objective function value is simply the most negative eigenvalue of $\bar{M}_1$, and the α-vector is the corresponding normalized eigenvector. The associated cut (4.10b) is given by:

$$0.126444 + 0.85355y - 0.7071x \geq 0. \tag{4.18}$$

Since the condition (4.11) holds true, the revised LP value after adding this cut is given by (4.13)

as

$$\frac{-(\alpha_1-\alpha_2)}{2\alpha_2} = -0.2929 < -0.25. \tag{4.19}$$

Hence, the optimal value is not recovered after adding this single cut.

Alternative Strategy of Sherali and Fraticelli [70]

Using Proposition 2 in Sherali and Fraticelli [70], or applying the cut generation routine SF of Section 2.1 on $\bar{M}_1$, yields $\alpha = (-1/2, 1)$ prior to normalization (or equivalently $\alpha = (1/2, -1)$), which leads to the cut (4.10b) given by:

$$\frac{1}{4} + y - x \geq 0.$$

This is the same as (4.15), thereby producing an optimum via the subsequent relaxation.

Remark 4.2. Note that since

$$\begin{bmatrix} 1 & x \\ x & y \end{bmatrix} \succeq 0 \Rightarrow y - x^2 \geq 0,$$

we can *directly* incorporate this convex constraint in the model, and by its simplicity, exploit its structure in the Lagrangian dual scheme as discussed in Sherali and Tuncbilek [72, 73]. In our example, as depicted in Figure 4.2, the optimization of the relaxation LP2 with this additional constraint would produce the optimal solution (x, y)=(1/2, 1/4). Although we employ only linear RLT relaxations in the sequel in order to avail of standard LP software, it is worthwhile to consider such convex variable bounding inequalities in a Lagrangian dual scheme for deriving tighter bounds. □

4.3 Class of v-semidefinite Cuts for Polynomial Programs

In this section, we devise several vectors v for which we can impose positive semidefiniteness on $[vv^T]$ in order to derive corresponding semidefinite cuts. We refer to the semidefinite cuts obtained by imposing $[v^{(k)}(v^{(k)})^T]_L \succeq 0$ as $v^{(k)}$-SDP cuts, where certain judicious choices of $v^{(k)}$ for different k are proposed below.

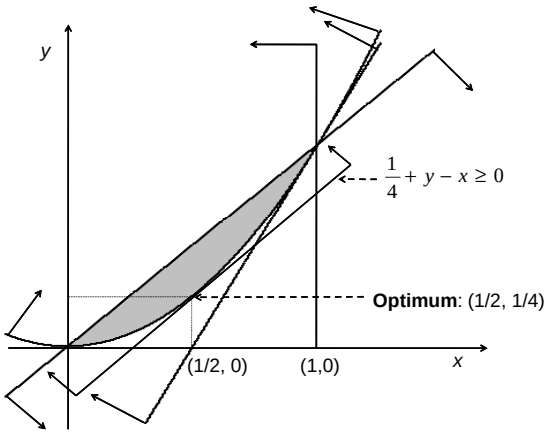


Figure 4.2: Including a convex SDP bounding constraint to tighten the relaxation.

For $k = 1$, a first particular choice of v is given as follows:

$$v^{(1)} = [1, \text{all monomials of order } \Delta \text{ using } x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{1+\binom{n+\Delta-1}{\Delta}}, \tag{4.20}$$

where recall that $\Delta \equiv \lfloor \delta/2 \rfloor$. Observe that for the special case of quadratic polynomial programs ($\delta = 2$), we get

$$v^{(1)} = [1, x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{n+1},$$

where Δ=1, which is precisely what is used in Sherali and Fraticelli [70].

Another choice of v, which expands on (4.20) and is motivated by the moment matrices of Lasserre [41, 43] is recommended as follows:

$$v^{(2)} = \begin{bmatrix} 1, \{x_1, x_2, \ldots, x_n\}, \{\text{all quadratic monomials using } x_1, x_2, \ldots, x_n\}, \ldots, \\ \{\text{all monomials of order } \Delta \text{ using } x_1, x_2, \ldots, x_n\} \end{bmatrix}^T \in \mathbb{R}^{\binom{n+\Delta}{\Delta}}. \tag{4.21}$$

Note that imposing positive semidefiniteness on $[v^{(2)}(v^{(2)})^T]$ would subsume that on $[v^{(1)}(v^{(1)})^T]$, since $v^{(1)}$ is a part of $v^{(2)}$. However, noting that $v^{(2)}$ can have a high dimensionality, we achieve a compromise by first identifying the *most discrepant term* X_{J^*}, where J^* is identified alternatively

by either (4.22a) or (4.22b) below based on the solution $(\bar{x}, \bar{X})$ to an initial RLT relaxation:

$$J^* \in \arg\max \left\{ \left| \bar{X}_J - \prod_{j \in J} \bar{x}_j \right| : J \subseteq \mathscr{N}^\delta \right\}; \tag{4.22a}$$

or

$$J^* \in \arg\max \left\{ \left| \bar{X}_J - \prod_{j \in J} \bar{x}_j \right| : \begin{array}{l} J \subseteq \mathscr{N}^\delta, \text{ and where } \prod_{j \in J} x_j \text{ is a monomial} \\ \text{term appearing in the original problem} \end{array} \right\}, \tag{4.22b}$$

and where X_J and $\mathscr{N}^\delta$ are as defined in Section 4.1. Then, using just the variables appearing in J^* (in lieu of $\{x_1, x_2, \ldots, x_n\}$), we apply $\{$(4.20) or (4.21)$\}$ to derive corresponding v-vectors $\{v^{(3)}$ or $v^{(4)}\}$, and $\{v^{(5)}$ or $v^{(6)}\}$, with J^* given by (4.22a) or (4.22b), respectively.

Likewise, motivated by the partitioning strategy and computations in Sherali and Tuncbilek [73], as a further augmentation to (4.22b), we select

$$J^* \in \arg\max \left\{ D_J : \begin{array}{l} J \subseteq \mathscr{N}^\delta, \text{ and where } \prod_{\in J} x_j \text{ is a monomial} \\ \text{term appearing in the original problem} \end{array} \right\}, \tag{4.22c}$$

where $D_J \equiv \sum_{\substack{r \in \{0,1,\ldots,R\}: \\ J_{rt} = J \text{ for some } t \in T_r}} \left| \alpha_{rt} \left[\bar{X}_{J_{rt}} - \prod_{j \in J_{rt}} \bar{x}_j, \right] \right|, \forall J \subseteq \mathscr{N}^\delta$. Once again, we apply $\{$(4.20) or (4.21)$\}$ while using just the variables appearing in J^* given by (4.22c), to derive the corresponding v-vectors $\{v^{(7)}$ or $v^{(8)}\}$.

In addition, we define

$$\Delta' = \lfloor |J^*|/2 \rfloor, \tag{4.23}$$

and apply (4.20) and (4.21) using just the variables in J^*, where J^* is given by (4.22a), (4.22b), or (4.22c), while *also replacing* Δ by Δ' in these equations. Let this produce the vectors $\{v^{(9)}$ or $v^{(10)}\}$, $\{v^{(11)}$ or $v^{(12)}\}$, and $\{v^{(13)}$ or $v^{(14)}\}$ using (4.22a), (4.22b), and (4.22c), respectively.

Furthermore, for any polynomial constraint $\phi_r(x) \geq \beta_r$ of order δ_r, define $\Delta_r \equiv \lfloor \frac{\delta}{2} - \frac{\delta_r}{2} \rfloor$. In the case when $\Delta_r \geq 1$, let $v^{(15)}$ and $v^{(16)}$ be as defined in (4.20) and (4.21), respectively, with Δ being

replaced by Δ_r. Then, motivated by (2.6), for k = 15 and 16, we impose

$$\left\{[\phi_r(x)-\beta_r]v^{(k)}(v^{(k)})^T\right\}_L \succeq 0, \tag{4.24}$$

and accordingly derive a corresponding α-vector for a semidefinite cut. This produces a *constraint-factor $v^{(k)}$-SDP* cut of the form:

$$\left\{[\phi_r(x)-\beta_r](\alpha^T v^{(k)})^2\right\}_L \geq 0. \tag{4.25}$$

The choice of the particular inequality $\phi_r(x) \geq \beta_r$ is made based on the extent of discrepancy in the original constraint satisfaction in contrast with the linearized situation as delineated by

$$\underset{\substack{r\in\{1,\ldots,R_1\}:\\ \phi_r(\bar{x})<\beta_r}}{\arg\max} \left\{[\phi_r(x)]_L|_{(\bar{x},\bar{X})} - \phi_r(\bar{x})\right\}. \tag{4.26}$$

If no original constraint is violated at the solution $(\bar{x},\bar{X})$, we select an active constraint having the smallest value of δ_r. Likewise, similar to the case of J^* above, by confining the variables considered in $v^{(k)}$ to those that appear in the selected inequality $\phi_r(x) \geq \beta_r$, we derive other cuts of the type (4.25) as follows. Let

$$V_r \equiv \{x_j : x_j \text{ is a variable appearing in } \phi_r(x) \geq \beta_r\},$$

and let

$$v^{(17)} = [1, \{\text{all monomials of order } \Delta_r \text{ using } x_j \in V_r\}]^T \in \mathbb{R}^{1+\binom{|V_r|+\Delta_r-1}{\Delta_r}}, \tag{4.27}$$

and

$$v^{(18)} = \begin{bmatrix} 1, \{x_j \in V_r\}, \{\text{all quadratic monomials using } x_j \in V_r\}, \ldots, \\ \{\text{all monomials of order } \Delta_r \text{ using } x_j \in V_r\} \end{bmatrix}^T \in \mathbb{R}^{\binom{|V_r|+\Delta_r}{\Delta_r}}. \tag{4.28}$$

Then, for k=17 and 18, we impose (4.24), where r is given by (4.26), in order to derive the corresponding constraint-factor $v^{(k)}$-SDP cuts of the type (4.25).

In the following section, we examine all the foregoing 18 choices of v-vectors to derive different classes of semidefinite cuts and test their performance, both individually, and also with the option

of generating multiple cuts utilizing particular combinations of these types.

Remark 4.3. In the above discussion, we have defined the semidefinite cuts obtained by imposing $[v^{(k)}(v^{(k)})^T]_L \succeq 0$ and $[[\phi_r(x) - \beta_r]v^{(k)}(v^{(k)})^T]_L \succeq 0$ as $v^{(k)}$-*SDP cuts* and *constraint-factor* $v^{(k)}$-*SDP cuts*. For ease in reference in our computational study of the different cut generation strategies, we will alternatively identify any such cut as $v_{i_1 i_2 i_3 i_4}$, where the indices i_1, i_2, i_3, and i_4 are defined as follows:

The first index i_1 identifies the type of SDP cut according to:

- $i_1 = 1$: $v^{(k)}$-SDP cut
- $i_1 = 2$: constraint factor $v^{(k)}$-SDP cut.

The highest degree of the monomial terms defining v is bounded by $\Delta \equiv \lfloor \delta/2 \rfloor$, where δ is the degree of the given polynomial program. Alternatively, we have defined the maximum degree of a monomial term in v according to $\Delta' \equiv \lfloor |J^*|/2 \rfloor$, where $J^* \subseteq \mathscr{N}^\delta$ yields the maximum violation with respect to some criterion. Note that maximum degree designation for the constraint factor v-vectors is given by $\Delta_r \equiv \lfloor \frac{\delta}{2} - \frac{\delta_r}{2} \rfloor$, where δ_r is the degree of the corresponding polynomial constraint. Accordingly, the second index i_2 represents the designation of the (maximum) degree of a monomial term defining v as follows:

- $i_2 = 0$: $\Delta_r \equiv \lfloor \delta/2 - \delta_r/2 \rfloor$(constraint-factor $v^{(k)}$-SDP cut)
- $i_2 = 1$: $\Delta \equiv \lfloor \delta/2 \rfloor$
- $i_2 = 2$: $\Delta' \equiv \lfloor |J^*|/2 \rfloor$.

The index i_3 represents the criterion used for selecting the set of variables for generating the v-vectors, where $\{x_1, x_2, \ldots, x_n\}$ is taken as the default set ($i_3 \equiv 1$). Alternatively, we have suggested utilizing the variables appearing in a set $J^* \subseteq \mathscr{N}^\delta$ that maximizes some measure of discrepancy in the RLT variable values. This yields the following variable set designations, where $i_3 = 5$ is applicable only to constraint-factor $v^{(k)}$-SDP cuts:

- $i_3 = 1$: $\{x_1, x_2, \ldots, x_n\}$

- $i_3 = 2$: $\{x_j : x_j \in J^*\}$, where J^* is given by (4.22a)
- $i_3 = 3$: $\{x_j : x_j \in J^*\}$, where J^* is given by (4.22b)
- $i_3 = 4$: $\{x_j : x_j \in J^*\}$, where J^* is given by (4.22c)
- $i_3 = 5$: $\{x_j : x_j$ appears in $\phi_{r^*}(x) \geq \beta_{r^*}\}$, where r^* is given by (4.26).

Finally, the v-vectors may have monomial terms only of the highest admissible degree as in (4.20), or monomial terms of degree up to the maximum admissible degree as in (4.21). Accordingly, the index i_4 designates the following:

- $i_4 = 1$: Monomial terms only of the highest admissible degree
- $i_4 = 2$: Monomial terms up to the highest admissible degree.

For convenience in reference, Figure 4.3 matches the vectors $v^{(k)}, k = 1, \ldots, 18$, identified earlier with the combinations $v_{i_1 i_2 i_3 i_4}$ defined above. □

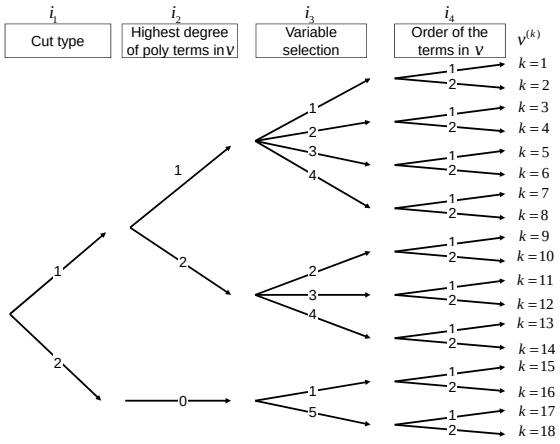


Figure 4.3: Classification of v-vectors as $v^{(k)}$ or $v_{i_1 i_2 i_3 i_4}$.

4.4 Illustrative Example

As an illustration, consider the following polynomial program of degree $\delta = 3$, from Sherali and Tuncbilek [71]:

$$\begin{array}{lll} \text{PP}(\Omega): & \text{Minimize} & \phi_0(x) = x_1x_2x_3 + x_1^2 - 2x_1x_2 - 3x_1x_3 + 5x_2x_3 - x_3^2 + 5x_2 + x_3 \\ & \text{subject to} & 4x_1 + 3x_2 + x_3 \leq 20, \\ & & x_1 + 2x_2 + x_3 \geq 1, \\ & & x \in \Omega \equiv \{2 \leq x_1 \leq 5, 0 \leq x_2 \leq 10, \text{ and } 4 \leq x_3 \leq 8\}. \end{array}$$

The RLT-based linear programming relaxation LP(Ω) involves the following set of variables:

$$(x,X) = \left\{ \begin{array}{c} x_1, x_2, x_3, X_{11}, X_{12}, X_{13}, X_{22}, X_{23}, X_{33}, X_{111}, X_{112} \\ X_{113}, X_{122}, X_{123}, X_{133}, X_{222}, X_{223}, X_{233}, X_{333} \end{array} \right\}.$$

The solution to LP(Ω) is given by:

$$(\bar{x},\bar{X}) = (3,0,8,8,0,24,0,0,64,20,0,64,0,0,192,0,0,0,512), \tag{4.29}$$

with objective value -120.

We now generate appropriate semidefinite cuts in order to tighten the linear programming relaxation LP(Ω). From (4.20), since $\Delta = 1$, the vector $v^{(1)}$(which coincides with $v^{(2)}$) is given by

$$v^{(1)} \equiv [1, x_1, x_2, x_3]^T.$$

Imposing $[v^{(1)}(v^{(1)})^T]_L \succeq 0$ at the current solution $(\bar{x},\bar{X})$, and employing the cut generation scheme SF described in Section 4.2.1, we derive the normalized α-vector, denoted $\alpha^{(1)}$, as,

$$\alpha^{(1)} = [-0.9487, 0.3162, 0, 0]^T,$$

with $\left\{(\alpha^{(1)})^T[v^{(1)}(v^{(1)})^T])\alpha^{(1)}\right\}_L|_{(\bar{x},\bar{X})} = -0.1 < 0$. Therefore, this yields the corresponding semidefinite cut,

$$0.90 - 0.60x_1 + 0.1X_{11} \geq 0. \tag{4.30}$$

Appending (4.30) to LP(Ω) and resolving the resulting LP relaxation yields an optimal solution to

the underlying polynomial program PP(Ω) given by,

$$(x^*, X^*) = (3,0,8,9,0,24,0,0,64,27,0,72,0,0,192,0,0,0,512), \tag{4.31}$$

with objective value -119.

Next, for the sake of further illustration, consider the vector $v^{(3)}$ (which coincides with $v^{(4)}$). Note that at the initial LP relaxation solution $(\bar{x}, \bar{X})$ given in (4.29), the maximum discrepant RLT variable is X_{113}, where the related discrepancy is $|X_{113} - x_1^2 x_3| = 8$. Hence, using (4.22a), we get $J^* = \{1,1,3\}$, thereby yielding,

$$v^{(3)} = v^{(4)} \equiv [1, x_1, x_3]^T. \tag{4.32}$$

The corresponding α-vector for the $v^{(k)}$-SDP cut, where $k = 3$ or 4, is given by $\alpha = [-0.9487, 0.3162, 0]^T$, which produces the semidefinite cut

$$0.90 - 0.60x_1 + 0.1X_{11} \geq 0.$$

This cut coincides with the previous cut (4.30), thereby producing an optimal solution to PP(Ω) via the next relaxation.

Likewise, considering only the discrepant terms that appear in the original problem as in either (4.22b) and (4.22c), we get $J^* = \{1,1\}$, and thus, the corresponding v-vector for all the cases $\{v^{(5)}, v^{(6)}, v^{(7)}, v^{(8)}\}$ is composed as follows:

$$v \equiv [1, x_1]^T. \tag{4.33}$$

Looking ahead at $\{v^{(9)}, \ldots, v^{(14)}\}$, and recognizing that $\Delta' = \lfloor |J^*|/2 \rfloor = 1$ corresponding to both $J^* = \{1,1,3\}$ and $J^* = \{1,1\}$, we get $\{v^{(9)}, v^{(10)}\}$ and $\{v^{(11)}, v^{(12)}, v^{(13)},$ and $v^{(14)}\}$ as respectively specified by (4.32) and (4.33). All these v-vectors generate the identical semidefinite cut

$$0.90 - 0.60x_1 + 0.1X_{11} \geq 0,$$

which once again coincides with the cutting plane given in (4.30), leading to an optimal solution to PP(Ω) upon solving the augmented relaxation.

Finally, consider the key constraint, $4x_1 + 3x_2 + x_3 \leq 20$, which is active at the solution $(\bar{x}, \bar{X})$.

Since $\Delta_r = \lfloor \frac{\delta}{2} - \frac{\delta_r}{2} \rfloor = \lfloor \frac{3}{2} - \frac{1}{2} \rfloor = 1$, this leads to $v^{(15)}$ and $v^{(16)}$ as given by: $v^{(15)} = v^{(16)} \equiv [1, x_1, x_2, x_3]^T$. Utilizing (4.24), we therefore assert that,

$$\begin{bmatrix} 20-4x_1-3x_2-x_3 & 20x_1-4X_{11}-3X_{12}-X_{13} & 20x_2-4X_{12}-3X_{22}-X_{23} & 20x_3-4X_{13}-3X_{23}-X_{33} \\ 20x_1-4X_{11}-3X_{12}-X_{13} & 20X_{11}-4X_{111}-3X_{112}-X_{113} & 20X_{12}-4X_{112}-3X_{122}-X_{123} & 20X_{13}-4X_{113}-3X_{123}-X_{133} \\ 20x_2-4X_{12}-3X_{22}-X_{23} & 20X_{12}-4X_{112}-3X_{122}-X_{123} & 20X_{22}-4X_{122}-3X_{222}-X_{223} & 20X_{23}-4X_{123}-3X_{223}-X_{233} \\ 20x_3-4X_{13}-3X_{23}-X_{33} & 20X_{13}-4X_{113}-3X_{123}-X_{133} & 20X_{23}-4X_{123}-3X_{223}-X_{233} & 20X_{33}-4X_{133}-3X_{233}-X_{333} \end{bmatrix} \succeq 0.$$

This generates a corresponding α-vector as $\alpha = [0.9732, -0.2298, 0, 0]^T$, leading to the constraint-factor SDP cut (4.25) as given by:

$$18.944-12.733x_1-2.842x_2-0.947x_3+2.845X_{11}+1.342X_{12}+0.447X_{13}-0.211X_{111}-0.158X_{112}-0.053X_{113} \geq 0.$$

Appending this cut to LP(Ω) and resolving again produces an optimal solution to PP(Ω) via a single cut.

Finally, note that since the constraints are linear in this example, we have that $[\phi_r(x)]_L \equiv \phi_r(x), \forall r$. Hence, neither constraint of PP(Ω) exhibits a discrepancy within (4.26). Moreover, both these constraints result in $\{v^{(17)}$ or $v^{(18)}\}$ being the same as $v^{(15)} \equiv v^{(16)}$, yielding the identical cut of the type (4.25) as derived above.

As demonstrated in Sherali and Tuncbilek [71], solving PP(Ω) by using RLT as a stand-alone approach would require an enumeration of three nodes in the branch-and-bound tree. However, as demonstrated above, the branch-and-cut process using the proposed SDP-cuts leads to an optimal solution being identified at the root node itself.

4.5 Computational Experience

In this section, we study the relative effectiveness of the different proposed v-SDP cuts and cut generation schemes, along with the performance of the branch-and-cut methodology based on RLT relaxations augmented with SDP cuts (abbreviated RLT+SDP). We begin with a root node analysis to select a competitive set of SDP cuts among the proposed alternatives, and to ascertain the number of cuts to generate per iteration, the relative performance of different cut generation routines, and the effect of problem degree and density. Thereafter, we explore the effect of SDP cut inheritance along the branch-and-bound tree and that of different cut generation strategies on the overall

CPU time for solving the polynomial programming test problems to global optimality. We also compare RLT+SDP to RLT implemented as a stand-alone approach, as well as to the commercial global optimization software BARON, Version 8.1.5 (Tawarmalani and Sahinidis [84]), and to the polynomial programming software GloptiPoly3 (Henrion and Lasserre [30]). In order to provide a suitable control test environment, we perform our initial screening and comparative computational analysis with randomly generated test instances as described below, and we finally illustrate the relative performances of the competitive procedures using some standard test instances from the literature.

Runs with BARON have been made on a remote 1.6 GHz Intel Pentium M processor running Linux (courtesy of Professor N. V. Sahinidis), whereas the RLT and RLT+SDP algorithms, as well as GloptiPoly3 have been implemented on a local 2.33 Ghz. Intel Xeon processor computer running Windows XP. The branch-and-bound algorithms utilize SNOPT Version 7 (Gill et al. [26]) as the nonlinear programming (local search) solver for updating upper bounds, and CPLEX 11.1 for solving the RLT-based LP relaxations.

4.5.1 Root Node Analysis

To test the proposed *v*-SDP cuts and cut generation routines, we randomly generated two sets of problems, namely sparse and dense, each of which is comprised of 20 test problems for each degree from two to five, thus yielding 160 instances. Table 4.1 provides the number of variables and constraints in the polynomial programs generated for each particular degree. For sparse problems, the percentage of nonzero objective and constraint coefficients was set to 5%. The percentage of nonzero objective and constraint coefficients in the dense problems was taken as 100% and 50%, respectively. The set of sparse problems were initially utilized to filter the 18 *v*-SDP cutting plane strategies, and then both the problem sets were used for selecting among some four particular competitive alternatives identified as discussed below.

For each methodology tested, we recorded the average CPU time and the average % *optimality gap closed* over the 20 instances of each of the four types of (sparse) test problems delineated in Table 4.1, where the *optimality gap closed* is computed as $\frac{\text{LB}_{RLT+SDP}-\text{LB}_{RLT}}{\text{UB}_{BARON}-\text{LB}_{RLT}}$ where LB_{RLT} and $\text{LB}_{RLT+SDP}$ are the lower bounds obtained at the root node by RLT and RLT+SDP, respectively, and UB_{BARON} is the upper bound produced by BARON after 1000 CPU seconds. To enable a

uniform application of this statistic, we tentatively eliminated from the present node zero analysis one quadratic, four degree-four, and four degree-five problems for which BARON could not find a feasible solution within the maximum imposed CPU time limit of 1000 seconds.

Table 4.1: Size of the random problems for the root node analysis.

Problem	Number of			
Degree	Variables	Constraints	RLT variables	RLT constraints
2	100	50	5151	20100
3	30	24	5456	37820
4	12	16	1820	17550
5	8	10	1287	15504

To begin with, we utilized a look-ahead version of the cut generation scheme SF (see Section 4.2.1 and Remark 4.1) with 10 *cycles* or *rounds* of cut generations, and with a maximum of $\bar{n}$ cuts per round, where $\bar{n}$ is the size of the matrix $\bar{M}$ used in Routine SF for which we impose positive semidefiniteness. (Later, we shall examine other alternatives to this *base-case strategy* using a filtered set of competitive v-SDP cuts.)

Table 4.2 presents the results obtained for the proposed 18 types of v-SDP cuts as identified by the first column (see Figure 4.3). The cutting planes v_{1111} and v_{1112} closed 11.4% and 13.5% of the optimality gap in 8.6 and 8.5 CPU seconds, respectively, whereas v_{2011}-v_{2052} closed $7.2-8.2\%$ of the optimality gap in 2.9-3.2 CPU seconds. The remainder of the v-vectors reduced the gap by 2.8-5.5% in 3.8-7 CPU seconds, where their performance relative to v_{1111}, v_{1112}, and v_{2011}-v_{2052} was most pronounced for quadratic (sparse) problems. We hence selected v_{1111} and v_{1112} along with v_{1142} and v_{2011} for further experimentations, in order to increase the diversity in the chosen strategies.

Next, we studied the effect of varying the maximum number of cuts derived per cut generation cycle. Table 4.3 presents results for generating (up to) 5, 10, 20, $\bar{n}/2$, or $\bar{n}$ cuts per cycle for each selected v-SDP cut over the set of test problems. Based on the composite statistic of the *average % optimality gap closed per unit CPU time*, as computed in the last column of Table 4.3, the strategy of generating $\bar{n}$ cuts per cycle is somewhat dominant. Although setting the number of cuts per cycle to $\bar{n}$ deteriorates this measure for some cases, the accompanying improvement in the % optimality gap closed is significant enough to warrant the selection of generating $\bar{n}$ cuts per cycle, which serves as our chosen option for further runs. We also examined the performance of

Table 4.2: Performance of the eighteen v-vectors.

	Quadratic		Cubic		Degree-four		Degree-five		**Mean**		
v-vector	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	**CPU Time (sec)**	**% opt. gap closed**	**% opt. gap closed per CPU time**
v_{1111}	4.1	6.7	11.7	8.4	8.7	11.7	10.0	18.7	**8.6**	**11.4**	**1.3**
v_{1112}	4.0	6.7	11.7	8.4	8.6	14.9	9.8	24.1	**8.5**	**13.5**	**1.6**
v_{1121}	2.3	4.2	5.7	2.4	3.8	1.3	3.4	3.1	**3.8**	**2.8**	**0.7**
v_{1122}	2.3	4.2	5.7	2.4	5.8	2.5	4.1	3.5	**4.5**	**3.2**	**0.7**
v_{1131}	2.4	4.4	9.3	3.6	6.5	3.0	4.6	5.2	**5.7**	**4.0**	**0.7**
v_{1132}	2.4	4.4	9.2	3.6	7.7	4.5	6.5	5.7	**6.5**	**4.6**	**0.7**
v_{1141}	2.8	4.4	9.8	4.5	7.1	4.1	5.2	6.9	**6.2**	**5.0**	**0.8**
v_{1142}	2.8	4.4	9.7	4.5	8.1	5.5	7.6	7.4	**7.1**	**5.4**	**0.8**
v_{1221}	2.3	4.2	5.6	2.4	3.9	1.3	3.4	3.1	**3.8**	**2.8**	**0.7**
v_{1222}	2.3	4.2	5.7	2.4	5.8	2.0	4.2	3.5	**4.5**	**3.0**	**0.7**
v_{1231}	2.4	4.4	9.2	3.6	6.5	3.0	4.6	5.2	**5.7**	**4.0**	**0.7**
v_{1232}	2.4	4.4	9.2	3.6	7.6	4.5	6.5	5.7	**6.4**	**4.5**	**0.7**
v_{1241}	2.8	4.4	9.7	4.5	7.0	4.2	4.9	6.9	**6.1**	**5.0**	**0.8**
v_{1242}	2.8	4.4	9.9	4.5	8.2	5.5	7.3	7.4	**7.0**	**5.4**	**0.8**
v_{2011}	0	0	1.0	0.4	6.0	14.3	4.8	18.2	**3.0**	**8.2**	**2.8**
v_{2012}	0	0	1.0	0.4	5.8	14.3	4.7	18.2	**2.9**	**8.2**	**2.8**
v_{2051}	0	0	1.0	0.2	6.9	11.3	5.0	17.5	**3.2**	**7.2**	**2.3**
v_{2052}	0	0	1.0	0.2	6.9	11.3	5.0	17.5	**3.2**	**7.2**	**2.2**

using a combination of the competitive v-SDP and constraint-factor v-SDP cuts v_{1112} and v_{2011}; we refer to this composite strategy as $v_{1112}\&v_{2011}$. The results in Table 4.3 indicate that these cuts reinforce each other and synergistically enhance the overall performance beyond their individual levels. We therefore continue our analysis with the v-SDP cuts v_{1112}, v_{2011}, and the composite cut $v_{1112}\&v_{2011}$.

We now turn our attention to comparing the different cut generation routines: SF with look-ahead, diagonal-sort, and full permutation (Section 4.2.1 and Remark 4.1); and EV and EV-10 (Section 4.2.2). Tables 4.4 and 4.5 present the results obtained for sparse and dense problems, respectively. For most of the selected v-SDP cuts and problem instances, the EV strategy produced the largest % optimality gap closed. However, the CPU time consumed by EV was much too high to make this suitable for implementation within a branch-and-cut algorithm. On the other hand, the look-ahead, diagonal-sort, full-permutation, and EV-10 strategies performed similarly and reasonably well, with EV-10 being selected as cut generation scheme for the base-case strategy. For the sake of interest, we also test all the other cut generation routines in the context of utilizing them within a branch-and-cut algorithm in Section 4.5.2. As far as the effect of sparsity is concerned,

Table 4.3: Effect of the maximum number of cuts per cut generation cycle.

		Quadratic		Cubic		Degree-four		Degree-five		**Mean**		
v-vector	max # of cuts per iter.	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	**CPU Time (sec)**	**% opt. gap closed**	**% opt. gap closed per CPU time**
v_{1111}	5	3.8	5.8	11.8	6.3	7.9	9.6	9.8	17.0	**8.3**	**9.7**	**1.16**
	10	3.9	6.1	11.8	7.5	7.7	10.4	9.4	17.4	**8.2**	**10.4**	**1.26**
	20	3.8	6.4	11.7	8.5	8.2	11.1	9.8	18.4	**8.4**	**11.1**	**1.33**
	$\bar{n}/2$	4.0	6.6	11.5	7.9	8.4	11.4	9.8	19.0	**8.4**	**11.3**	**1.34**
	$\bar{n}$	4.1	6.7	11.7	8.4	8.7	11.7	10.0	18.7	**8.6**	**11.4**	**1.33**
v_{1112}	5	3.8	5.8	11.8	6.3	8.0	11.5	9.4	20.3	**8.3**	**11.0**	**1.33**
	10	3.9	6.1	11.8	7.5	8.1	12.7	10.2	21.8	**8.5**	**12.0**	**1.41**
	20	4.0	6.4	11.7	8.5	8.3	14.1	9.8	22.8	**8.4**	**13.0**	**1.54**
	$\bar{n}/2$	4.3	6.6	11.8	7.9	8.6	14.5	10.0	23.1	**8.7**	**13.0**	**1.51**
	$\bar{n}$	4.0	6.7	11.7	8.4	8.6	14.9	9.8	24.1	**8.5**	**13.5**	**1.58**
v_{1142}	5	4.1	4.3	11.0	4.0	8.2	3.5	7.5	6.4	**7.7**	**4.5**	**0.59**
	10	3.3	4.3	10.7	4.4	7.8	4.3	7.4	7.3	**7.3**	**5.1**	**0.70**
	20	2.7	4.9	9.6	4.4	8.0	5.0	7.5	7.4	**6.9**	**5.4**	**0.78**
	$\bar{n}/2$	2.8	4.4	10.0	4.3	8.1	5.6	7.2	7.1	**7.0**	**5.3**	**0.76**
	$\bar{n}$	2.8	4.4	9.7	4.5	8.1	5.5	7.6	7.4	**7.1**	**5.4**	**0.77**
v_{2011}	5	0	0	1.1	0.3	6.3	13.1	5.0	17.7	**3.1**	**7.8**	**2.52**
	10	0	0	1.0	0.4	5.7	14.3	4.7	18.1	**2.9**	**8.2**	**2.87**
	20	0	0	1.1	0.4	5.9	14.3	5.2	18.3	**3.0**	**8.2**	**2.71**
	$\bar{n}/2$	0	0	1.0	0.4	6.1	13.9	5.0	17.9	**3.0**	**8.0**	**2.63**
	$\bar{n}$	0	0	1.0	0.4	6.0	14.3	4.8	18.2	**3.0**	**8.2**	**2.77**
v_{1112} & v_{2011}	5	3.8	5.8	11.8	6.7	9.0	28.2	10.8	38.0	**8.9**	**19.7**	**2.22**
	10	3.9	6.1	11.7	8.3	9.3	30.8	11.3	39.5	**9.1**	**21.2**	**2.34**
	20	3.8	6.4	11.8	9.0	10.0	33.0	11.3	40.2	**9.2**	**22.1**	**2.40**
	$\bar{n}/2$	3.9	6.6	11.5	8.4	9.5	33.8	11.0	40.7	**9.0**	**22.4**	**2.49**
	$\bar{n}$	4.1	6.7	11.8	8.8	10.3	34.0	10.8	41.8	**9.2**	**22.8**	**2.48**

dense and sparse problems yielded similar values for the optimality gap closed, with the CPU times being relatively higher for dense problems. The relative performances of the different v-SDP cuts were similar. Examining the overall cut generation effort and % optimality gap closed metrics, we selected the composite cut $v_{1112}\&v_{2011}$ for further experimentation.

Table 4.4: Performance of cut generation schemes at the root node for sparse problems.

		Quadratic		Cubic		Degree-four		Degree-five		**Mean**	
v-vector	Cut Generation Schemes	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	**CPU Time (sec)**	**% opt. gap closed**
v_{1112}	Look-ahead	4.0	6.7	11.7	8.4	8.6	14.9	9.8	24.1	**8.5**	**13.5**
	Diagonal-sort	3.3	6.5	12.2	8.5	7.2	9.6	8.0	16.1	**7.7**	**10.2**
	Full Permutation	12.1	6.4	12.6	10.4	14.1	19.3	11.2	27.3	**12.5**	**15.9**
	EV	191.8	0.6	24.3	35.4	197.8	26.5	20.4	31.7	**108.6**	**23.6**
	EV-10	11.3	10.9	15.2	15.0	12.0	29.1	11.6	17.4	**12.5**	**18.1**
v_{2011}	Look-ahead	0	0	1.0	0.4	6.0	14.3	4.8	18.2	**3**	**8.2**
	Diagonal-sort	0	0	1.2	0.4	7.7	14.1	5.5	19.1	**3.6**	**8.4**
	Full Permutation	0	0	1.1	0.4	7.5	14.4	5.5	18.5	**3.5**	**8.3**
	EV	0	0	17.5	0.2	14.2	13.6	6.2	18.0	**9.5**	**8**
	EV-10	0	0	1.7	0.2	6.2	13.4	6.2	17.6	**3.5**	**7.8**
v_{1112} & v_{2011}	Look-ahead	4.1	6.7	11.8	8.8	10.3	34.0	10.8	41.8	**9.2**	**22.8**
	Diagonal-sort	3.3	6.5	12.6	8.5	11.0	26.6	11.1	34.6	**9.5**	**19.1**
	Full Permutation	12.1	6.4	12.8	10.6	15.2	39.3	13.1	47.0	**13.3**	**25.8**
	EV	191.5	0.6	35.8	35.2	205.7	49.8	23.9	50.9	**114.2**	**34.1**
	EV-10	6.9	3.3	13.4	13.6	15.9	36.4	13.9	45.6	**12.5**	**24.7**

Table 4.5: Performance of cut generation schemes at the root node for dense problems.

		Quadratic		Cubic		Degree-four		Degree-five		**Mean**	
v-vector	Cut Generation Schemes	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	CPU Time (sec)	% opt. gap closed	**CPU Time (sec)**	**% opt. gap closed**
v_{1112}	Look-ahead	5.6	4.2	16.8	3.1	7.5	8.9	10.0	24.2	**10**	**10.1**
	Diagonal-sort	5.7	4.1	15.5	3.2	7.3	8.0	8.5	19.6	**9.3**	**8.7**
	Full Permutation	5.6	5.2	15.9	3.3	11.7	14.5	11.5	27.5	**11.2**	**12.6**
	EV	160.8	46.6	28.0	6.2	59.2	50.6	24.8	32.3	**68.2**	**33.9**
	EV-10	7.0	8.9	17.4	1.7	11.8	21.3	10.5	27.4	**11.7**	**14.8**
v_{2011}	Look-ahead	0	0	26.7	3.7	8.2	7.7	10.4	28.9	**11.3**	**10.1**
	Diagonal-sort	0	0	26.8	2.0	13.1	6.8	9.9	29.4	**12.5**	**9.5**
	Full Permutation	0	0	30.6	3.1	9.8	7.7	12.9	29.7	**13.3**	**10.1**
	EV	0	0	170.4	7.6	27.9	9.0	27.9	24.7	**56.6**	**10.3**
	EV-10	0	0	18.0	3.2	10.9	7.8	19.7	25.9	**12.2**	**9.2**
v_{1112} & v_{2011}	Look-ahead	5.6	4.2	26.5	6.7	9.8	16.5	12.4	49.8	**13.6**	**19.3**
	Diagonal-sort	5.7	4.1	30.2	5.4	14.1	15.7	12.4	46.6	**15.6**	**18**
	Full Permutation	5.8	5.2	31.3	6.5	15.2	20.9	15.8	53.4	**17**	**21.5**
	EV	158.0	46.6	157.0	15.4	56.4	56.6	44.3	62.7	**103.9**	**45.3**
	EV-10	7.0	8.9	25.6	5.3	18.2	28.8	23.5	53.9	**18.6**	**24.2**

4.5.2 Cut Performance within a Branch-and-Cut Algorithm

In this section, we first conduct experiments to select the best cut generation routine and cut inheritance strategy within a branch-and-cut algorithm for solving the polynomial test problems to global optimality. After composing an effective branch-and-cut methodology of this type (referred to as RLT+SDP), we provide comparisons of this RLT+SDP procedure against using RLT as a stand-alone process (without v-SDP cuts), as well as using the global optimization solver BARON, Version 8.1.5 ([84]), and the specialized polynomial program solver GloptiPoly3 ([30]).

In the implemented branch-and-cut algorithm, the node having the least lower bound was selected for branching. The branching variable, x_{j^*}, was selected according to

$$j^* \in \arg\max_{j=1,\ldots,n}\{\theta_j\}, \text{ where } \theta_j = \max_{t=1,\ldots,\delta-1} \max_{\substack{J \subseteq \mathcal{N}^\delta \\ |J|=t}} \{|\bar{X}_{(J\cup j)} - \bar{x}_j \bar{X}_J|\}, \forall j = 1,\ldots,n, \tag{4.34}$$

and where $(\bar{x}, \bar{X})$ represents the solution to the current LP relaxation. We partitioned the current interval $[l_{j^*}, u_{j^*}]$ of the selected variable according to dichotomy that $x_{j^*} \in [l_{j^*}, \bar{x}_{j^*}] \vee x_{j^*} \in [\bar{x}_{j^*}, u_{j^*}]$. This guarantees convergence to a global optimum (Sherali and Tuncbilek [71]). We fathomed nodes for which the lower bound satisfies the inequality $(v^* - \text{LB}) \leq \varepsilon |v^*|$, where v^* is the incumbent solution value, LB is the lower bound computed for this node, and ε is the optimality gap tolerance (which was set to 0.01 in our computational analysis). We also limited the computational time for each run to 3600 CPU seconds. Furthermore, we utilized three criteria for terminating the cut generation cycles: minimum % optimality gap closed per cycle (set to 0.005); minimum % optimality gap closed over three cycles (set to 0.02); and a specified maximum number of cycles (set to 5).

We generated 20 polynomial programming test problems, five for each degree from two to five, with 100% nonzero objective coefficients. The number of variables and constraints for these randomly generated polynomial programs are displayed in Table 4.6. The % of positive objective coefficients were varied as 10%, 50%, and 90%. For clarity of presentation, we first consider only the instances having 90% positive objective coefficients to compose the proposed branch-and-cut methodology, and then provide computational results over the entire test bed.

First, we compared the cut generation routines SF, EV, and EV-10 of Sections 4.2.1 and 4.2.2, including the look-ahead, diagonal-sort, and full permutation variants of SF, and the EV-10 (sorted)

Table 4.6: Size of the random problems for the branch-and-cut algorithm.

Problem	Number of			
Degree	Variables	Constraints	RLT variables	RLT constraints
2	30	16	496	1830
3	19	12	1540	9880
4	12	6	1820	17550
5	11	4	4368	65780

variant of EV-10 as described in Remark 4.1. Table 4.7 presents the results obtained. Overall, EV-10 (sorted) performed the best, saving on average 8%, 4%, 23%, 25%, and 16% of the CPU effort in comparison with EV, EV-10, look-ahead, diagonal-sort, and full permutation, respectively. This reinforces our previous recommendation of utilizing EV-10 as the prescribed cut generation routine.

Table 4.7: Branch-and-cut CPU times (seconds) for the different cut generation schemes.

Cut Generation Scheme	EV	EV-10	EV-10 (sorted)	Look Ahead	Diagonal Sort	Full Permutation
δ=2 n=30	1152	1454	1317	1966	2261	1820
δ=3 n=19	1323	1396	1349	1682	1722	1638
δ=4 n=12	1844	975	980	1071	1045	858
δ=5 n=11	575	877	865	1156	961	1051
Average	**1223**	**1176**	**1128**	**1469**	**1497**	**1342**

Having selected the cut generation scheme, we next investigated the effect of cut inheritance along the branch-and-cut tree. The results obtained are summarized in Table 4.8. Although the SDP cuts generated at any node are also valid for all other nodes, the effect of node-specific RLT relaxations and the accompanying discrepancies in the corresponding LP solutions makes cuts that are inherited from elsewhere in the branch-and-cut tree less effective. Furthermore, the accumulation of SDP cuts increases the problem complexity as well as the LP solution times significantly. When we implemented cut inheritance from parent nodes, the average CPU time improved from 1317 to 1188, 1349 to 1284, and 865 to 754 CPU seconds for degree two, three, and five problems, respectively, while for degree four problems, the algorithm performed similar in comparison with the no inheritance case, with an overall average improvement of 6.5%. When the cut inheritance was extended one step further to include grandparent nodes as well, the average CPU time for cubic problems improved somewhat, but the performance for degree four and five problems worsened. On average, cut inheritance from both parent and grandparent nodes, as opposed to from

just parent nodes, deteriorated the overall performance by 3.3%. Thus, cut inheritance only from parent nodes was selected for implementation.

Table 4.8: Effect of cut inheritance.

Problem Size	Cut Inheritance: No Inheritance	Only from Parent Nodes	From Parent and Grandparent Nodes
δ=2 n=30	1317	1188	1183
δ=3 n=19	1349	1284	1250
δ=4 n=12	980	995	1063
δ=5 n=11	865	754	865
Average	**1128**	**1055**	**1090**

Next, we compared the performances of RLT+SDP and RLT over the expanded range of random instances by introducing problems having various percentages (10%, 50%, and 90%) of positive coefficients in the objective function, while still retaining 100% density in the objective coefficients.

Table 4.9: Performance of RLT+SDP and RLT.

			% of positive objective coefficients							
		# of	10%		50%		90%		**Average**	
Problem Size	Algorithm	unsolved instances	CPU Time	% impr. over RLT	CPU Time	% impr. over RLT	CPU Time	% impr. over RLT	**CPU Time**	**% impr. over RLT**
δ=2 n=30	RLT+SDP	1	16	-12	831	55	1188	43	**678**	**48**
	RLT	3	14		1839		2074		**1309**	
δ=3 n=19	RLT+SDP	2	19	64	1265	22	1284	31	**856**	**28**
	RLT	4	54		1626		1863		**1181**	
δ=4 n=12	RLT+SDP	1	6	61	152	39	995	37	**384**	**38**
	RLT	1	15		249		1590		**618**	
δ=5 n=11	RLT+SDP	0	137	71	447	54	754	60	**446**	**60**
	RLT	1	471		977		1872		**1106**	
Overall Averages	**RLT+SDP**	**1**	**45**	**68**	**674**	**43**	**1055**	**43**	**591**	**44**
	RLT	**2.25**	**138**		**1173**		**1850**		**1054**	

The results for RLT and RLT+SDP are displayed in Table 4.9. Observe that, on average, the CPU time required by either method increased significantly with an increase in the percentage of positive coefficients. Also, RLT+SDP outperformed RLT for all cases except for quadratic problems having 10% positive objective coefficients. Moreover, RLT failed to optimize nine out of the 60 problems, whereas RLT+SDP solved five of these nine cases in 1700 CPU seconds on average. For the four problems that both algorithms were unable to solve in the set limit of 3600

CPU seconds, the average optimality gap[1] at termination using RLT+SDP was 42%, versus 66% when implementing RLT. Excluding the nine cases for which it terminated prematurely, RLT optimized the remaining 51 polynomial programs in 604 CPU seconds on average, whereas RLT+SDP reduced the effort required to solve these 51 instances by 59% to only 246 CPU seconds.

We also compared RLT+SDP to the commercial global optimization software BARON, Version 8.1.5 (Tawarmalani and Sahinidis [84]), uniformly using an optimality tolerance of $\varepsilon = 0.01$. Table 4.10 presents the results obtained. BARON terminated prematurely for 40 out of the 60 test problems with an average optimality gap of 490%, whereas RLT+SDP was able to solve 36 of these 40 instances in 561 CPU seconds on average. For the remaining four unsolved problems, the average optimality gaps at termination were 2625% using BARON and 42% implementing RLT+SDP. For the 20 problems that BARON solved to optimality, the average CPU time required was 673 seconds, whereas RLT+SDP solved these same instances in 43 CPU seconds on average, yielding a savings of 94% in computational effort. Overall, over the 60 instances, RLT+SDP saved 44% and 77% in computational effort over RLT and BARON, respectively. In sharp contrast, GloptiPoly 3 was unable to solve any of these 60 test instances due to out-of-memory errors.

Table 4.10: Performance of RLT+SDP and BARON.

Problem Size	Algorithm	# of unsolved instances	% of positive obj. coefficients 10% CPU time	50% CPU time	90% CPU time	**Average CPU Time**
δ=2 n=30	RLT+SDP	1	16	831	1188	**678**
	BARON	7	17	2596	3227	**1947**
δ=3 n=19	RLT+SDP	2	19	1265	1284	**856**
	BARON	13	2921	3600	3600	**3374**
δ=4 n=12	RLT+SDP	1	6	152	995	**384**
	BARON	10	215	3600	3600	**2472**
δ=5 n=11	RLT+SDP	0	137	447	754	**446**
	BARON	10	1623	3600	2893	**2705**
Overall Averages	RLT+SDP	**1**	**45**	**674**	**1055**	**591**
	BARON	**10**	**1194**	**3349**	**3330**	**2624**

Whereas we have used randomly generated instances above to provide insights into the effect of problem and instance structure on computational effort, we conclude our experimental analysis by presenting results for RLT+SDP versus RLT in Table 4.11 pertaining to the solution of some standard test problems from the literature. The problems are taken from [22, 32, 41, 49, 67, 88]

[1]The optimality gap is computed as $\frac{\text{UB}-\text{LB}}{|\text{UB}|}100\%$ where UB and LB are the best upper and lower bounds found by each algorithm.

with the number of variables (n), the degree of the polynomial programs (δ), and the number of constraints (m) displayed in Table 4.11. The test problems from Floudas et al. [22] include nonconvex quadratic programs (Chapter 2), quadratically constrained quadratic programs (Chapter 3), and univariate polynomial programs (Chapter 4). The first three problems from Lasserre [41] are unconstrained problems of degree two, two, and six, respectively, whereas the last problem has three quadratic inequality constraints. The two instances from Visweswaran and Floudas [88], Examples 0 and 5, are unconstrained problems. Problem 71 in Hock and Schittkowski [32] has one fourth degree inequality and one quadratic equality constraint, whereas Problem 343 in Schittkowski [67] has two inequality constraints of degrees three and five, respectively. Finally, Example 3 in Manousiouthakis and Sourlas [49] is a reactor network design problem having one linear inequality and six quadratic equalities. Given the sparsity of the test instances, we utilized the composite cut $v_{1142}\&v_{2011}$ for the RLT+SDP strategy based on its promising performance for sparse problems as previously observed in Table 4.2.

Table 4.11 displays the results obtained. Note that the CPU times for most problems are quite small and the differences are inconsequential. However, in general, the number of nodes enumerated with RLT+SDP exhibited a significant decrease compared to RLT as a stand-alone approach. For most of the quadratic problems from Floudas et al. [22], the RLT relaxations automatically satisfied positive semidefiniteness; hence, no SDP cuts were generated for these test cases. Even so, the computational effort for RLT+SDP was not higher than that for RLT for these cases since the SDP cut generation routines are automatically turned off after a certain number of nodes (10) for which at most one cut is generated. For Problem 9 in Chapter 2 of Floudas et al. [22], the computational effort required by the composite cut $v_{1142}\&v_{2011}$ versus RLT was reduced by 17%. Continuing with quadratically constrained problems in Chapter 3 of Floudas et al. [22], we observe that the computational effort for Problem 1 using RLT+SDP with $v_{1142}\&v_{2011}$ versus RLT decreased by 54%, from 5.78 CPU seconds to 2.67 CPU seconds, while exploring 183 nodes compared to 525 nodes, respectively. For Problems 2, 3, and 4, there was no significant difference between RLT and RLT+SDP. For the univariate polynomial programs in Chapter 4 of Floudas et al. [22], the average number of nodes explored was reduced from 16.3 to 6 when the SDP cuts $v_{1142}\&v_{2011}$ were incorporated within RLT. Furthermore, the polynomial programs from Lasserre [41] were optimized on average within 0.1 CPU seconds and 0.2 CPU seconds while exploring 6 and 21 nodes using RLT+SDP and RLT, respectively. Finally, the performance of RLT and RLT+SDP were quite close for the remaining problems, except for Problem 343 in Schittkowski [67] for which RLT+SDP optimized in 5.55 CPU seconds compared to 196.2 CPU seconds for

Table 4.11: Results for polynomial programs from the literature.

		RLT		RLT+SDP (v_{1142}&v_{2011})	
Problem Source	(n, δ, m)	# of nodes	CPU time	# of nodes (# of cuts)	CPU time
Chapter 2, Problem 1 [22]	(5, 2, 1)	17	0.09	17 (0)	0.09
Chapter 2, Problem 2 [22]	(6, 2, 2)	1	0.02	1 (0)	0.02
Chapter 2, Problem 3 [22]	(13, 2, 9)	1	0.02	1 (0)	0.02
Chapter 2, Problem 4 [22]	(6, 2, 5)	1	0.02	1 (0)	0.02
Chapter 2, Problem 5 [22]	(10, 2, 11)	1	0.02	1 (0)	0.02
Chapter 2, Problem 6 [22]	(10, 2, 5)	11	0.09	11 (0)	0.11
Chapter 2, Problem 7 [22]	(20, 2, 10)	159	6.83	159 (0)	6.98
Chapter 2, Problem 8 [22]	(24, 2, 10)	17	1.06	17 (0)	1.05
Chapter 2, Problem 9 [22]	(10, 2, 1)	9145	104.39	8551 (284)	95.28
Chapter 3, Problem 1 [22]	(8, 2, 6)	525	5.78	183 (353)	2.67
Chapter 3, Problem 2 [22]	(5, 2, 6)	3	0.03	1 (3)	0.02
Chapter 3, Problem 3 [22]	(6, 2, 6)	5	0.05	5 (0)	0.05
Chapter 3, Problem 4 [22]	(3, 2, 3)	49	0.25	49 (0)	0.25
Chapter 4, Problem 1 [22]	(1, 6, 0)	19	0.09	7 (28)	0.09
Chapter 4, Problem 3 [22]	(1, 5, 0)	9	0.03	3 (5)	0.03
Chapter 4, Problem 4 [22]	(1, 4, 0)	9	0.03	5 (12)	0.05
Chapter 4, Problem 5 [22]	(2, 6, 0)	51	0.95	13 (33)	0.38
Chapter 4, Problem 6 [22]	(1, 6, 0)	15	0.08	9 (21)	0.08
Chapter 4, Problem 7 [22]	(1, 4, 0)	5	0.02	3 (5)	0.03
Chapter 4, Problem 8 [22]	(2, 4, 1)	17	0.13	3 (6)	0.05
Chapter 4, Problem 9 [22]	(2, 4, 2)	5	0.19	5 (3)	0.03
Example 1 [41]	(2, 4, 0)	25	0.14	5 (16)	0.08
Example 2 [41]	(2, 4, 0)	23	0.13	5 (13)	0.08
Example 3 [41]	(2, 6, 0)	27	0.5	7 (42)	0.2
Example 5 [41]	(2, 2, 3)	9	0.05	7 (1)	0.05
Example 0 [88]	(1, 3, 0)	3	0.03	3 (6)	0.03
Example 5 [88]	(1, 6, 0)	15	0.06	11 (24)	0.09
Problem 71 [32]	(4, 4, 2)	17	0.5	5 (59)	0.33
Problem 343 [67]	(3, 7, 2)	401	196.17	11 (328)	5.55
Example 3 [49]	(8, 2, 7)	15	0.23	17 (16)	0.34

RLT. Similar to the sparse cut option EV-10, when we employed the SF cut generation strategy with the look-ahead and the full permutation schemes, the respective times for the four cases in Table 4.11 where the computational times exceeded 2 CPU seconds were 6.61, 79.69, 14.48, 0.66 and 6.83, 83.11, 12.16, 0.64 CPU seconds.

Finally, we mention that for the foregoing four cases, the software BARON consumed 0.10, 2.65, 0.18, and 0.10 seconds compared with 6.98, 95.28, 2.67, and 5.55 for RLT+SDP, respectively. Note that, as mentioned in the introduction section, *bound-constraint* and *constraint-factor RLT product constraints*, which are proven to be very effective especially for quadratic programs

[72], are not included in the relaxation LP(Ω), since our primary objective here is to explore the effect of v-SDP cuts on general polynomial programs. Indeed, the study in Sherali and Tuncbilek [74] indicates that by including bound-constraint factor product constraints as well as range reduction strategies, the effort required for Problem 343 in Schittkowski [67], for example, is reduced by 89.8% in comparison with using the simple generic RLT relaxation (4.3). Likewise, the commercial software BARON incorporates sophisticated partitioning and range reduction strategies, which evidently greatly assist in solving the sparse, mainly quadratic problems from the literature. However, on more general challenging polynomial programs tested herein, even the simple RLT implementation augmented with SDP cuts performs significantly better than BARON (see Table 4.10), thereby suggesting that the incorporation of the proposed SDP cuts within commercial software might be a worthwhile endeavor.

On the other hand, GloptiPoly 3 failed to optimize Problem 7 in Chapter 2 and Problem 1 in Chapter 3 due to memory issues, whereas Problem 9 in Chapter 2 was solved in 3.84 seconds. Finally, GloptiPoly 3 terminated in 0.84 seconds with the declaration that optimality was not verified for the Problem 343 in Schittkowski [67].

Chapter 5

Combined Bound-Grid-Factor Constraints for Enhancing RLT Relaxations for Polynomial Programs

This chapter studies the global optimization of polynomial programming problems using Reformulation-Linearization Technique (RLT)-based linear programming (LP) relaxations. We introduce a new class of *bound-grid-factor* constraints that can be judiciously used to augment the basic RLT relaxations in order to improve the quality of lower bounds and enhance the performance of global branch-and-bound algorithms. Certain theoretical properties are established that shed light on the effect of these valid inequalities in driving the discrepancies between RLT variables and their associated nonlinear products to zero. To preserve computational expediency while promoting efficiency, we propose certain concurrent and sequential cut generation routines and various grid-factor selection rules. The results indicate a significant tightening of lower bounds, which yields an overall reduction in computational effort for solving a test-bed of polynomial programming problems to global optimality in comparison with the basic RLT procedure as well as the commercial software BARON.

5.1 Introduction

Consider the following polynomial program **PP** of order δ:

$$\textbf{PP:}\quad \text{Minimize}\quad \phi_0(x) \tag{5.1a}$$

$$\text{subject to}$$

$$\phi_r(x) \geq \beta_r, \forall r = 1,\ldots,R_1 \tag{5.1b}$$

$$\phi_r(x) = \beta_r, \forall r = R_1+1,\ldots,R \tag{5.1c}$$

$$x \in \Omega \equiv \{x : 0 \leq l_j \leq x_j \leq u_j < \infty, \forall j \in \mathscr{N}\}, \tag{5.1d}$$

where

$$\phi_r(x) \equiv \sum_{t\in T_r} \alpha_{rt}\Big[\prod_{j\in J_{rt}} x_j\Big], \text{ for } r = 0,\ldots,R,$$

and where T_r is an index set for the terms defining $\phi_r(.)$, with α_{rt} being a real coefficient associated with the monomial $\prod_{j\in J_{rt}} x_j,\ \forall t \in T_r,\ r = 0,\ldots,R$. Given $\mathscr{N} \equiv \{1,\ldots,n\}$, we let $\mathscr{N}^d$ denote a *multi-set of order d*, which is comprised of distinct combinations of indices (assumed throughout to be arranged in nondecreasing order) that belong to the d-degree Cartesian product $\mathscr{N}\times\ldots\times\mathscr{N}$. Accordingly, we have $J_{rt} \subseteq \bigcup_{d=1}^{\delta} \mathscr{N}^d, \forall t \in T_r, r = 0,\ldots,R$.

Following the RLT procedure described by Sherali and Tuncbilek [71], we utilize linear programming relaxations embedded within a branch-and-bound algorithm to solve Problem PP to global optimality. The main construct in this procedure, which is the focus of the present chapter, is to derive a linear programming (LP) relaxation based on the current node's bounding hyperrectangle, Ω, for the purpose of computing lower bounds. To generate this LP relaxation at the root node, denoted LP(Ω), we proceed as follows (a similar technique is applied to generate the relaxation LP(Ω') for any other node subproblem in the branch-and-bound tree that is predicated on the defining hyperrectangle $\Omega' \subset \Omega$). Define the *bound-factors* $(x_j - l_j) \geq 0$ and $(u_j - x_j) \geq 0,\ \forall j \in \mathscr{N}$. We then generate the corresponding *bound-factor product* RLT constraints *of order* δ [71] by taking the products of the bound-factors δ at a time as specified below:

$$\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j) \geq 0, \forall (J_1 \cup J_2) \subseteq \mathscr{N}^{\delta}, |J_1 \cup J_2| = \delta. \tag{5.2}$$

After appending the bound-factor product constraints (5.2) to Problem PP, we linearize the resulting augmented program by substituting a new RLT variable for each distinct monomial according to:

$$X_J = \prod_{j\in J} x_j, \forall J \subseteq \mathscr{N}^d, d = 2,\ldots,\delta, \tag{5.3}$$

where for convenience, we shall denote $X_{\{j\}} \equiv x_j, \forall j \in \mathscr{N}$, and $X_{\{\emptyset\}} \equiv 1$. Henceforth, whenever such a linearization operation is applied to any polynomial expression $[.]$ under the substitution (5.3), we shall denote the resultant linear function in (x,X) by $[.]_L$. Thus, we obtain LP(Ω) as follows:

LP(Ω): Minimize $[\phi_0(x)]_L$ (5.4a)

subject to

$$[\phi_r(x)]_L \geq \beta_r, \forall r = 1,\ldots,R_1 \tag{5.4b}$$

$$[\phi_r(x)]_L = \beta_r, \forall r = R_1+1,\ldots,R \tag{5.4c}$$

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \forall (J_1 \cup J_2) \subseteq \mathscr{N}^\delta, |J_1 \cup J_2| = \delta \tag{5.4d}$$

$$x \in \Omega. \tag{5.4e}$$

Now, in addition, suppose that we identify grid-points $\bar{x}_{jg}, g = 1,\ldots,G_j$, for each $j \in \mathscr{N}$, such that $l_j < \bar{x}_{j1} < \bar{x}_{j2} < \ldots < \bar{x}_{jG_j} < u_j, \forall j \in \mathscr{N}$. Define $\mathscr{G}$ to be the collection of multi-sets comprised of up to $\lfloor \delta/2 \rfloor$ index combinations (j,g) (with possible repetitions). For each $J_3 \in \mathscr{G}$, let J_3^* be the corresponding multi-set given by

$$J_3^* \equiv \bigcup_{(j,g)\in J_3} \{j,j\}.$$

Then, as a generalization of bound-factor RLT constraints, we examine the combined *bound-grid-factors*:

$$\{(x_j - l_j), (u_j - x_j), (x_j - \bar{x}_{jg})^2 \text{ for } g = 1,\ldots,G_j\}, \forall j \in \mathscr{N},$$

and accordingly, we generate the following *bound-grid-factor constraints of order* δ:

$$\left[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\prod_{(j,g)\in J_3}(x_j - \bar{x}_{jg})^2\right]_L \geq 0, \forall \{J_1,J_2,J_3\} : (J_1 \cup J_2 \cup J_3^*) \subseteq \mathscr{N}^\delta. \tag{5.5}$$

Assuming that we have q grid-points for each variable $j \in \mathcal{N}$, and denoting $\delta' = \lfloor \delta/2 \rfloor$, the total number of restrictions in (5.5) is given by:

$$\sum_{i=0}^{\delta'} \binom{qn+i-1}{i} \binom{2n+\delta-2i-1}{\delta-2i}.$$

The main contributions of this chapter are as follows. First, the simple squared grid-factor constraints introduced by Sherali and Tuncbilek [74] for the box-constrained univariate case are extended to multivariate polynomial programming problems (5.1) as delineated in (5.5). Second, we present related theoretical properties of (5.5) to provide insights and motivate their structure. Third, we explore and compare various choices for locating grid-points while judiciously selecting variables for which such grid-points are generated, and we also examine the utility of employing reduced LP relaxations where the reformulated model is augmented with a specific subset of (5.5) that incorporates particular bound- and grid-factors, which helps curtail the computational effort. Finally, we perform extensive computational analyses at the root node as well as in a branch-and-bound scheme designed to solve Problem PP to global optimality to demonstrate the improvement that accrues with an appropriate inclusion of bound-grid-factor constraints as opposed to using only bound-factor constraints within the RLT-based procedure, and we also provide a benchmark comparison with respect to the commercial global optimization software BARON-Version 9.0.6 (Sahinidis and Tawarmalani [66] and Tawarmalani and Sahinidis [84]).

The remainder of this chapter is organized as follows. In Section 5.2, we present theoretical results related to (5.5) to glean insights into the nature and strength of these constraints. In Section 5.3, we introduce certain concurrent and sequential constraint generation routines as well as various schemes for locating grid-points, and we provide an illustrative example in Section 5.4. The implemented branch-and-bound algorithm that incorporates (5.5) is described in Section 5.5, and detailed computational results for the proposed procedure as well as for the basic RLT method and BARON are presented in Section 5.6.

5.2 Theoretical Results and Insights

Let $\mathscr{Z}$ denote the bound-grid-factor constraints (5.5) defined in terms of the variables (x,X), where recall that $X \equiv \{X_J, \forall J \in \mathscr{N}^d, d = 2,\ldots,\delta\}$. In this section, we present some theoretical results to provide insights into the role played by the grid-factor terms within (5.5), in comparison with the purely bound-factor constraints utilized by Sherali and Tuncbilek [71]. To begin with, the following common property holds true.

Proposition 5.1. The bound-grid-factor constraints (5.5) of order δ imply the identical constraints of lower order d, for $d = 1,\ldots,\delta-1$.

Proof. Similar to the proof of Lemma 2 in Sherali and Tuncbilek [71]. □

In contrast with the main property of bound-factor product constraints, which asserts that whenever $x_j = l_j$ or $x_j = u_j$ for any $j \in \mathscr{N}$, we also get the following nonlinear relationships holding true for any feasible solution to (5.5):

$$X_{J\cup\{j\}} = X_J x_j, \forall J \subseteq \mathscr{N}^d, d = 1,\ldots,\delta-1, \tag{5.6}$$

we now obtain the results given below in case x_j equals a grid-point value within (5.5).

Proposition 5.2. Consider any $j \in \mathscr{N}$, and let $\hat{x}_j \in (l_j,u_j)$ be a particular grid-point in the interval for the variable x_j. Then, denoting $X_{jj} \equiv X_{\{j,j\}}$, we have that,

$$(x,X) \in \mathscr{Z} \text{ with } x_j = \hat{x}_j \Rightarrow \hat{x}_j^2 \le X_{jj} \le \hat{x}_j(l_j+u_j) - l_j u_j, \tag{5.7}$$

with the error of X_{jj} from its desired value $\hat{x}_j^2$ being bounded by $0 \le (X_{jj} - \hat{x}_j^2) \le (\hat{x}_j - l_j)(u_j - \hat{x}_j)$.

Proof. By Proposition 5.1, $\mathscr{Z}$ implies the lower order constraints $\left[(x_j - \hat{x}_j)^2\right]_L \ge 0$ and $\left[(x_j - l_j)(u_j - x_j)\right]_L \ge 0$, which respectively yield the following inequalities:

$$2x_j\hat{x}_j - \hat{x}_j^2 \le X_{jj} \le x_j(l_j+u_j) - l_j u_j. \tag{5.8}$$

Substituting $x_j = \hat{x}_j$ into (5.8) establishes (5.7), with the error bound being given by (5.7) as

$$0 \leq (X_{jj} - \hat{x}_j^2) \leq \hat{x}_j(l_j + u_j) - l_j u_j - \hat{x}_j^2 = (\hat{x}_j - l_j)(u_j - \hat{x}_j).$$

□

Note that the geometric consequence of (5.8) is that the value of X_{jj} in $\mathscr{Z}$ is greater than or equal to the value realized by the tangent plane to the convex function x_j^2 at the point $x_j = \hat{x}_j$, and is less than or equal to the value defined by the chord or the concave envelope of the function x_j^2 over the interval $[l_j, u_j]$.

The next result establishes that if the solution to the RLT-based LP relaxation of some underlying polynomial program in the presence of bound-grid-factor product constraints (5.5) achieves $x_j = \hat{x}_j$ with $X_{jj} = \hat{x}_j^2$ also holding true, then all the RLT variables corresponding to higher order monomials in x_j will faithfully reproduce the respective values of the nonlinear terms they represent, i.e., we will get $X_J = \hat{x}_j^{|J|} = \hat{x}_j^d, \forall J \equiv \{j\}^d, d = 3, \ldots, \delta$.

Proposition 5.3. Consider any $j \in \mathscr{N}$, and let $\hat{x}_j \in (l_j, u_j)$ be a particular grid-point in the interval for the variable x_j. Then, for $\delta \geq 3$,

$$(x, X) \in \mathscr{Z} \text{ with } x_j = \hat{x}_j \text{ and } X_{jj} = \hat{x}_j^2 \Rightarrow X_J = \hat{x}_j^{|J|} = \hat{x}_j^d, \forall J \equiv \{j\}^d, d = 3, \ldots, \delta. \tag{5.9}$$

Proof. Consider any feasible solution $(x, X) \in \mathscr{Z}$ where we additionally have $x_j = \hat{x}_j$ and $X_{jj} = \hat{x}_j^2$. Hence, noting that the given assertion (5.9) is trivially true for $d = 2$, by induction, assume that

$$X_J = \hat{x}_j^{|J|} = \hat{x}_j^d, \forall J \equiv \{j\}^d, \text{ for } d = 2, \ldots, t, \text{ where } 2 \leq t \leq \delta - 1, \tag{5.10}$$

and consider $d = t + 1$. By Proposition 5.1, the following constraints are implied by $\mathscr{Z}$:

$$\left[(x_j - \hat{x}_j)^2 (x_j - l_j)^{t-1}\right]_L \geq 0, \text{ and } \left[(x_j - \hat{x}_j)^2 (u_j - x_j)(x_j - l_j)^{t-2}\right]_L \geq 0. \tag{5.11}$$

The two inequalities in (5.11) can be respectively written as (5.12) and (5.13) below, where $f(x_j)$ is a polynomial in x_j of degree $t - 3$:

$$\left[(x_j - \hat{x}_j)^2 (x_j - l_j)\left[x_j^{t-2} + f(x_j)\right]\right]_L \geq 0, \tag{5.12}$$

$$\Big[(x_j-\hat{x}_j)^2(u_j-x_j)\big[x_j^{t-2}+f(x_j)\big]\Big]_L \geq 0. \tag{5.13}$$

Expanding (5.12) and (5.13) and rearranging terms yields the following, where $J \equiv \{j\}^{t+1}$:

$$X_J \geq \Big[l_j(x_j-\hat{x}_j)^2\big[x_j^{t-2}+f(x_j)\big]+\big[2x_j^2\hat{x}_j-x_j\hat{x}_j^2-x_j^3\big]f(x_j)+\big[2x_j\hat{x}_j-\hat{x}_j^2\big]x_j^{t-1}\Big]_L, \tag{5.14}$$

$$X_J \leq \Big[u_j(x_j-\hat{x}_j)^2\big[x_j^{t-2}+f(x_j)\big]+\big[2x_j^2\hat{x}_j-x_j\hat{x}_j^2-x_j^3\big]f(x_j)+\big[2x_j\hat{x}_j-\hat{x}_j^2\big]x_j^{t-1}\Big]_L. \tag{5.15}$$

Upon linearizing the right-hand sides in (5.14) and (5.15) by using the RLT substitution process (5.3), and applying the induction hypothesis (5.10) when $x_j=\hat{x}_j$, we see that the first two compound terms in (5.14) and (5.15) are zero noting that $f(x_j)$ is a polynomial in x_j of degree $t-3$, while the common third compound term reduces to the following, where $J' \equiv \{j\}^t$ and $J'' \equiv \{j\}^{t-1}$:

$$2\hat{x}_jX_{J'}-\hat{x}_j^2X_{J''}=2\hat{x}_j\hat{x}_j^t-\hat{x}_j^2\hat{x}_j^{t-1}=\hat{x}_j^{t+1}.$$

Hence, (5.14) and (5.15) together imply that $X_J=\hat{x}_j^{t+1}=\hat{x}_j^{|J|}$. □

Observe that the bound-grid-factor product constraints, $(x,X)\in\mathscr{Z}$, tighten the RLT relaxation by requiring (x,X) to satisfy additional inter-relationships beyond the fundamental bound-factor product restrictions, as will be borne out by our computational experiments in Section 5.6. However, unlike the aforementioned result (5.6) for the case when $x_j=l_j$ or $x_j=u_j$, we do not necessarily get that

$$\{(x,X)\in\mathscr{Z} \text{ with } x_j=\hat{x}_j\} \Rightarrow X_{J\cup j}=X_J\hat{x}_j, \forall J\in\mathscr{N}^d, 1\leq d\leq \delta-1, j\in\mathscr{N}, \tag{5.16}$$

or that

$$\{(x,X)\in\mathscr{Z} \text{ with } x_j=\hat{x}_j\} \Rightarrow X_{J\cup\{j,j\}}=X_J\hat{x}_j^2, \forall J\in\mathscr{N}^d, 1\leq d\leq \delta-2, j\in\mathscr{N}, \tag{5.17}$$

where $\hat{x}_j\in(l_j,u_j)$ is a grid-point for $x_j, j\in\mathscr{N}$. In other words, the optimal objective function values of the linear programs (5.18) and (5.19) specified below, when minimized or maximized, do not necessarily equal zero:

$$\min/\max\{X_{J\cup j}-X_J\hat{x}_j : (x,X)\in\mathscr{Z}, x_j=\hat{x}_j\}, \tag{5.18}$$

$$\min/\max\{X_{J\cup\{j,j\}}-X_J\hat{x}_j^2 : (x,X)\in\mathscr{Z}, x_j=\hat{x}_j\}. \tag{5.19}$$

For example, consider $n = 2$ and $\delta = 3$, with $[l_j, u_j] \equiv [0,1], \forall j \in \mathcal{N}$, and suppose that $G_j = 1, \forall j \in \mathcal{N}$, with the single grid-point $\hat{x}_j$, say, being the interval mid-point, 0.5, for each variable $j \in \mathcal{N}$. As displayed in Table 5.1, the optimal values of the linear programs of type (5.18) and (5.19) are not all zero, and in fact, the objective functions exhibit a strictly positive range (maximal minus minimal values) for each relevant J when we impose $x_1 = \hat{x}_1 = 0.5$. However, when we further impose $X_{11} = \hat{x}_1^2 = 0.25$, we see that we obtain zero optimal values in (5.18) and (5.19) precisely when $J \in \{\{1\}, \{1,1\}\}$ and $J \in \{\emptyset, \{1\}\}$, respectively, as propounded by Proposition 5.3.

Table 5.1: Minimal and maximal values of the LPs (5.18) and (5.19) under stated conditions.

Problem type	J	Objective function	$x_1 = \hat{x}_1 = 0.5$		$x_1 = \hat{x}_1 = 0.5$ and $X_{11} = \hat{x}_1^2 = 0.25$	
			min	max	min	max
(5.18)	$\{1\}$	$X_{11} - 0.5x_1$	0	0.25	0	0
	$\{2\}$	$X_{12} - 0.5x_2$	−0.25	0.25	−0.125	0.125
	$\{1,1\}$	$X_{111} - 0.5X_{11}$	0	0.25	0	0
	$\{1,2\}$	$X_{112} - 0.5X_{12}$	−0.083	0.25	−0.0625	0.0625
	$\{2,2\}$	$X_{122} - 0.5X_{22}$	−0.25	0.25	−0.1875	0.1875
(5.19)	$\emptyset$	$X_{11} - 0.25$	0	0.25	0	0
	$\{1\}$	$X_{111} - 0.25x_1$	0	0.375	0	0
	$\{2\}$	$X_{112} - 0.25x_2$	−0.16	0.375	−0.125	0.125

As noted above, this is unlike the case when $x_j = l_j$ or $x_j = u_j$, whence as proven in Sherali and Tuncbilek [71], we indeed obtain relationships of the type (5.16) and (5.17) holding true (see (5.6)), with the optimal values of the LPs of the type (5.18) and (5.19) being necessarily zero. Nonetheless, as we shall see empirically, the bound-grid-factor constraints appreciably assist in tightening the RLT relaxations.

5.3 Alternative Bound-Grid-Factor Constraint Generation Strategies

In this section, we introduce several alternative bound-grid-factor constraint generation routines and grid-point assignment rules in order to derive effective constraints of type (5.5) for solving polynomial programming problems. We begin by describing the proposed concurrent and sequential constraint generation routines. We next present certain equal distance, incumbent, optimal-to-

parent, and optimal-to-self grid-point assignment rules, as well as a particular modified restricted version called the positive-discrepancy rule.

In the *concurrent constraint generation routine*, all the bound-grid-factor constraints (5.5) are generated and directly appended to the model prior to solving the initial RLT-based LP relaxation. On the other hand, in the *sequential constraint generation routine*, we first optimize the LP relaxation that is derived using the bound-factor constraints alone, and then prescribe the grid-points based on the optimal solution obtained for this relaxation (using the optimal-to-self grid-point assignment rule as described in the sequel below), and finally append the corresponding combined bound-grid-factor restrictions to the relaxation. Subsequently, we re-optimize the LP relaxation to obtain a possibly improved lower bound. By thus utilizing the information specific to the current node, the sequential constraint generation routine tends to produce tighter relaxations while, however, requiring additional computational effort because it solves two relaxations at each node.

Next, we discuss three particular choices for locating grid-points in concert with the concurrent constraint generation routine. For the first of these, called the *equal distance rule*, we place the grid-points uniformly within each variable interval as follows:

$$\bar{x}_{jg} = l_j + g\,\frac{u_j - l_j}{G_j + 1},\ \text{for } g = 1, \ldots, G_j,\ \forall j \in \mathcal{N}.$$

For the second technique, called the *incumbent rule*, we utilize the equal distance rule until an incumbent solution is available, and then employ the current incumbent solution, x^*, to locate a single grid-point at the corresponding incumbent solution value x_j^* for each variable $j \in \mathcal{N}$, provided that

$$\min\{(x_j^* - l_j), (u_j - x_j^*)\} \geq 0.01(u_j - l_j). \tag{5.20a}$$

The remaining grid-points for such a variable are next distributed at equal distances within the two partitions $[l_j, x_j^*]$ and $[x_j^*, u_j]$, by respectively allocating $\left\lceil \frac{G_j - 1}{2} \right\rceil$ and $\left\lfloor \frac{G_j - 1}{2} \right\rfloor$ grid-points to the larger and smaller of these intervals. In this context, note that an incumbent solution might be derived at the beginning of the root node analysis using a local optimization solver, and so an x^* might be available for use as above at node zero itself. For the case when (5.20a) does not hold, including the cases when x_j^* lies at one of the bounds of the variable or even outside the current node's interval for x_j, we set $G_j \equiv 0$ and do not generate any grid-points for this variable.

In a third method for locating grid-points, called the *optimal-to-parent rule*, we examine the optimal solution $\bar{x}^{parent}$ to the LP relaxation at the parent node, and for each $j \in \mathcal{N}$ such that $\bar{x}_j^{parent}$ satisfies

$$\min\{(\bar{x}_j^{parent} - l_j), (u_j - \bar{x}_j^{parent})\} \geq 0.01(u_j - l_j), \tag{5.20b}$$

we set one of the grid-points at $\bar{x}_j^{parent}$, and then distribute $\left\lceil \frac{G_j - 1}{2} \right\rceil$ of the remaining grid-points uniformly within $[l_j, \bar{x}_j^{parent}]$ and the rest within $[\bar{x}_j^{parent}, u_j]$ at equal distances, assuming that the former interval is wider, or vice versa otherwise. Moreover, for this rule, the root node relaxation utilizes the grid-factors as determined by the incumbent rule. Again, whenever (5.20b) is violated, we set $G_j \equiv 0$ for this variable.

As mentioned above, the foregoing three grid-point selection rules are applied in concert with the concurrent constraint generation routine. On the other hand, when implementing the sequential constraint generation routine, we utilize the optimal solution $\bar{x}$ obtained for the initial LP relaxation solved at each node to determine the grid-points for the current node by following the same process as used above for x^* (or $\bar{x}^{parent}$), before solving the subsequent augmented node relaxation. We refer to this grid-point assignment method as the *optimal-to-self rule*. Note that this rule is implemented in the stated manner at the root node as well.

The number of bound-grid-factor product constraints can be prohibitively high for large $|\hat{\mathcal{N}}|$, where $\hat{\mathcal{N}} \equiv \{j \in \mathcal{N} : G_j \geq 1\}$. Hence, to curtail the size of the LP relaxations, we adopt two types of reduction strategies that utilize only a subset of the constraints defined in (5.5). In the first of these, which is henceforth utilized as the default implementation of bound-grid-factor constraints, we generate the pure bound-factor product constraints of order δ from (5.5) (which correspond to $J_3 \equiv \emptyset$), but for the case $J_3 \neq \emptyset$, we generate only those members from (5.5) that involve just the variables from $\hat{\mathcal{N}}$. In other words, we adopt the following bound-grid-factor product constraints of order δ:

$$\text{Constraints (5.5) for } J_3 \equiv \emptyset, \text{ plus those corresponding to } J_3 \neq \emptyset \text{ with } (J_1 \cup J_2 \cup J_3^*) \subseteq \hat{\mathcal{N}}^{\delta}. \tag{5.21}$$

Whereas (5.21) itself provides an adequate filter for (5.5) in most cases (except for the equal distance rule where $\hat{\mathcal{N}} = \mathcal{N}$), we propose and test another optional strategy to further reduce the size of the LP relaxation at each node of branch-and-bound enumeration tree other than the root node. Denoting $(\bar{x}, \bar{X})$ as the relaxation solution of the parent node, or that for the initial node

relaxation when considering the subsequent node relaxation for the sequential constraint generation routine, we compute the *total discrepancy* for each variable as given by:

$$\theta_j \equiv \sum_{d=1}^{\delta-1} \sum_{J \in \mathscr{N}^d} \left| \bar{X}_{J \cup j} - \bar{X}_J \bar{x}_j \right|, \forall j \in \mathscr{N}, \tag{5.22}$$

and accordingly, we designate $\bar{\mathscr{N}} \equiv \{j \in \hat{\mathscr{N}} : \theta_j > 0\}$. Then, for the current node relaxation, we include all the bound-factor constraints in (5.5) (i.e., for $J_3 \equiv \emptyset$) plus those bound-grid-factor constraints in (5.5) that correspond to $J_3 \neq \emptyset$ with $(J_1 \cup J_2 \cup J_3^*) \subseteq \bar{\mathscr{N}}^\delta$. This model-reduction technique is called the *positive-discrepancy strategy*. Note that for the sequential constraint generation routine, we can implement this positive-discrepancy rule even at the root node while solving the second of the two node relaxations.

5.4 Illustrative Example

To illustrate the generation of the bound-grid-factor constraints (5.5), consider a two-variable ($n = 2$) polynomial program having degree $\delta = 3$. Assume without loss of generality that the lower and upper bounds on both the variables are 0 and 1, respectively, and furthermore, suppose that we select a single grid-point for each variable ($G_j = 1, \forall j \in \mathscr{N}$), which bisects its interval range, i.e., $\bar{x}_{jg} = 1/2, \forall j = 1,2; g = 1$. The resulting RLT-based LP relaxation has 28 bound-grid-factor constraints, eight of which involve grid-factors. The particular constraints (5.5) that involve at least one grid-factor are listed below for illustration:

$$\begin{aligned}
&J_1 = \{1\}, J_2 = \varnothing, J_3 = \{(1,1)\} : [(x_1)(x_1 - \tfrac{1}{2})^2]_L &&= X_{111} - X_{11} + \tfrac{1}{4}x_1 \geq 0 \\
&J_1 = \varnothing, J_2 = \{1\}, J_3 = \{(1,1)\} : [(1-x_1)(x_1 - \tfrac{1}{2})^2]_L &&= -X_{111} + 2X_{11} - \tfrac{5}{4}x_1 + 1/4 \geq 0 \\
&J_1 = \{2\}, J_2 = \varnothing, J_3 = \{(1,1)\} : [(x_2)(x_1 - \tfrac{1}{2})^2]_L &&= X_{112} - X_{12} + \tfrac{1}{4}x_2 \geq 0 \\
&J_1 = \varnothing, J_2 = \{2\}, J_3 = \{(1,1)\} : [(1-x_2)(x_1 - \tfrac{1}{2})^2]_L &&= -X_{112} + X_{11} + X_{12} - x_1 - \tfrac{1}{4}x_2 + \tfrac{1}{4} \geq 0 \\
&J_1 = \{1\}, J_2 = \varnothing, J_3 = \{(2,1)\} : [(x_1)(x_2 - \tfrac{1}{2})^2]_L &&= X_{122} - X_{12} + \tfrac{1}{4}x_1 \geq 0 \\
&J_1 = \varnothing, J_2 = \{1\}, J_3 = \{(2,1)\} : [(1-x_1)(x_2 - \tfrac{1}{2})^2]_L &&= -X_{122} + X_{12} + X_{22} - \tfrac{1}{4}x_1 - x_2 + \tfrac{1}{4} \geq 0 \\
&J_1 = \{2\}, J_2 = \varnothing, J_3 = \{(2,1)\} : [(x_2)(x_2 - \tfrac{1}{2})^2]_L &&= X_{222} - X_{22} + \tfrac{1}{4}x_2 \geq 0 \\
&J_1 = \varnothing, J_2 = \{2\}, J_3 = \{(2,1)\} : [(1-x_2)(x_2 - \tfrac{1}{2})^2]_L &&= -X_{222} + 2X_{22} - \tfrac{5}{4}x_2 + \tfrac{1}{4} \geq 0.
\end{aligned}$$

5.5 Branch-and-Bound Algorithm

In this section, we briefly describe our branch-and-bound algorithm for solving polynomial programming problems, where, as mentioned in Section 5.1, lower bounds are computed using the RLT-based LP relaxation LP(Ω) that accommodates (5.21), given the (revised) sub-hyperrectangle Ω defining the current node subproblem. At each stage in this process, the node having the least lower bound is selected for further partitioning (best-first strategy), where the branching variable is identified by determining

$$j^* \in \arg\max_{j \in \mathcal{N}} \theta_j,$$

where θ_j is given by (5.22), and where $(\bar{x}, \bar{X})$ represents the solution to the current node's relaxation. As proven by Sherali and Tuncbilek [71], we can achieve convergence to a global optimum by partitioning the selected interval $[l_{j^*}, u_{j^*}]$ at either its mid-point or at the current LP relaxation solution value, $\bar{x}_{j^*}$. In our implementation, we utilized the average of these two values and, accordingly, we split the interval $[l_j^*, u_j^*]$ at the point

$$\frac{\frac{l_{j^*} + u_{j^*}}{2} + \bar{x}_{j^*}}{2}.$$

Note that this also assures theoretical convergence to a global optimum using an identical line of proof as in Sherali and Tuncbilek [71]. Whenever the lower bound (LB) computed at any node satisfies $(v^* - LB) \leq \varepsilon |v^*|$, where v^* is the current incumbent solution value and where $\varepsilon \geq 0$ is a prescribed optimality tolerance (we selected $\varepsilon = 0.01$ in our computations), we fathom the current node. The procedure terminates when all the nodes are fathomed (no active node exists).

Remark 5.1. The experimental results in Paulavičius et al. [60] reveal that breadth-first and depth-first node selection strategies achieve better execution times than the best-first strategy for Lipschitz optimization problems. The authors suggest that the decrease in computational effort may be due to the fact that the breath-first and depth-first strategies follow FIFO and LIFO structures, respectively, and hence save the time normally spent in node insertion and deletion operations that are performed while ordering nodes as in the best-first strategy. However, note that for branch-and-bound algorithms of the type discussed above for solving continuous nonconvex optimization problems to global optimality, the best-first node selection strategy is integral to assuring theoretical convergence. Nonetheless, motivated by the results of Paulavičius et al. [60], we explore the relative benefits of implementing the breadth-first node selection strategy up to a designated level

of the enumeration tree (or a specified time limit), before switching to the best-first strategy to ensure theoretical convergence. □

5.6 Computational Results

In this section, we evaluate the effectiveness of the bound-grid-factor restrictions (5.21) using a set of 15 randomly generated polynomial programming problems from Sherali et al. [76], with five instances for each $(\delta, n, R) \in \{(3, 19, 12), (4, 12, 6), \text{and } (5, 11, 4)\}$, where, as before, δ, n, and R respectively denote the degree of the polynomial program, the number of variables, and the number of constraints (assumed to be all inequalities here, i.e., $R_1 = R$), and where the degree of the i^{th} constraint is set to $[(i-1) \bmod \delta] + 1, \forall i = 1, \ldots, R$. The percentage of nonzero objective and constraint coefficients are taken as 100% and 25%, of which 90% and 50% are taken as positive, respectively. For the degree-five test cases, we modified the right-hand side values of some constraints from those in Sherali et al. [76] to render the problems more difficult to solve using RLT (all test cases are available via the Web-page http://filebox.vt.edu/users/dalkiran/website/).

We begin our discussion in Section 5.6.1 with a root node analysis to demonstrate the tightness of RLT-based LP relaxations when enhanced with additional grid-factor induced restrictions. We explore the performances of the concurrent and sequential constraint generation routines (generically denoted as RLT+GF), and compare them against using simply bound-factor constraints in the traditional RLT approach (denoted simply as RLT in the sequel). Next, in Section 5.6.2, we embed these bound-grid-factor constraints in a branch-and-bound procedure as described in Section 5.5, and present the corresponding computational results for RLT and RLT+GF, as well as for the commercial global optimization software BARON-Version 9.0.6 (Tawarmalani and Sahinidis [84]). Runs with BARON were performed on a workstation having two 2.13 Ghz Intel Xeon processors with 4GB of RAM and running Windows 7, whereas the RLT and RLT+GF procedures were implemented on a workstation having 2.33 GHz Intel Xeon processor with 3.25 GB of RAM and running Windows XP, all subject to a 3600 CPU seconds time limit. The RLT-based LP relaxations were solved using the barrier optimizer of CPLEX 11.1 (we also attempted the dual simplex option, which is often competitive in such contexts, but the barrier approach yielded better results, perhaps not only due to solving the LP relaxations faster, but also due to the nature of the resulting solution belonging to the interior of the optimal face when alternative optimum solutions exist).

The nonlinear programming (local search) solver SNOPT-Version 7 (Gill et al. [26]) was utilized for updating the incumbent solution at each node, with the derived relaxation solution being used as a warm-start solution.

5.6.1 Root Node Analysis

In this section, we compare the root node performances of RLT with RLT+GF, when the latter is implemented using the concurrent and sequential constraint generation routines based on designating one, two, and three grid-points (G_j-values) per variable. For the present analysis, we focus on examining the equal distance and optimal-to-self grid-point assignment rules implemented respectively in combination with the concurrent and sequential constraint generation routines, where for the latter, the second relaxation solved adopts the positive-discrepancy strategy. The incumbent and optimal-to-parent grid-point assignment rules are tested later in Section 5.6.2 along with the other foregoing rules within a branch-and-bound context.

Table 5.2: Root node % optimality gaps (with CPU times in parentheses) for RLT and RLT+GF using the concurrent and sequential constraint generation routines with G grid-points for each variable.

(δ, n, R)	RLT	Concurrent ($G=1$)	Concurrent ($G=2$)	Concurrent ($G=3$)
(3, 19, 12)	30.2% (0.3)	26.6% (0.3)	26.1% (0.4)	25.8% (0.4)
(4, 12, 6)	27.8% (1.2)	21.3% (1.7)	20.3% (2.1)	20% (2.9)
(5, 11, 4)	24.9% (22.1)	15.6% (47.2)	13.5% (63.2)	12.3% (84.5)
Mean	**27.7% (7.9)**	**21.2% (16.4)**	**20% (21.9)**	**19.4% (29.3)**
(δ, n, R)	RLT	Sequential ($G=1$)	Sequential ($G=2$)	Sequential ($G=3$)
(3, 19, 12)	30.2% (0.3)	26.5% (0.7)	26.2% (0.7)	25.9% (0.7)
(4, 12, 6)	27.8% (1.2)	21.6% (2.9)	20.7% (3.1)	20.1% (3.6)
(5, 11, 4)	24.9% (22.1)	16.3% (55.2)	14.3% (64.3)	14% (80.8)
Mean	**27.7% (7.9)**	**21.5% (19.6)**	**20.4% (22.7)**	**20% (28.4)**

For each variant tested, we report in Table 5.2 the average CPU time (over five instances of each size) for solving the RLT-based LP relaxation and the average % optimality gap obtained at the root node (computed as $100\frac{UB^*-LB}{UB^*}$, where LB and UB^* respectively denote the lower bound obtained at the root node and the incumbent solution value produced via BARON within 3600 CPU seconds), and we present in Table 5.3 the average number of bound-grid-factor constraints generated along with the accompanying average computational effort for deriving these constraints. Recall that

we solve two relaxations at each node for the sequential constraint generation routine; hence, the displayed values represent the sum of the associated CPU times.

The results in Table 5.2 reveal that the proposed bound-grid-factor restrictions significantly tighten the relaxation over simply using the bound-factor constraints. While the effect of the number of grid-points generated on the % optimality gap entails diminishing marginal returns, the computational effort continues to increase roughly linearly due to number of grid-factor restrictions added, where the latter are displayed in Table 5.3. Note that the average number of bound-grid-factor restrictions derived using the sequential cut generation routine with the positive-discrepancy strategy is less than that with the concurrent cut generation routine since several variables turn out to coincide with their lower or upper bounds in the initial relaxation solution.

Table 5.3: Number of constraints derived (with CPU times in parentheses) for RLT and RLT+GF using the concurrent and sequential constraint generation routines with G grid-points for each variable.

(δ, n, R)	RLT	Concurrent ($G=1$)	Concurrent ($G=2$)	Concurrent ($G=3$)
(3, 19, 12)	9880 (0.1)	10602 (0.1)	11324 (0.2)	12046 (0.2)
(4, 12, 6)	17550 (0.5)	21228 (0.6)	25050 (0.8)	29016 (0.9)
(5, 11, 4)	65780 (4.1)	89496 (5.9)	115874 (7.9)	144914 (10.2)
Mean	**31070 (1.6)**	**40442 (2.2)**	**50749 (2.9)**	**61992 (3.7)**
(δ, n, R)	RLT	Sequential ($G=1$)	Sequential ($G=2$)	Sequential ($G=3$)
(3, 19, 12)	9880 (0.1)	10420 (0.1)	10959 (0.1)	11499 (0.2)
(4, 12, 6)	17550 (0.5)	19587 (0.6)	21716 (0.6)	23938 (0.7)
(5, 11, 4)	65780 (4.1)	72756 (4.6)	80672 (5.3)	89529 (5.9)
Mean	**31070 (1.6)**	**34254 (1.8)**	**37782 (2.0)**	**41655 (2.3)**

When RLT is compared to RLT+GF, we observe that the average optimality gap of 27.7% produced using RLT decreased to 21.2% and 21.5%, respectively, using the concurrent and sequential constraint generation routines along with a single grid-point per variable. However, on average, the computational effort for solving the corresponding relaxations increased from 7.9 CPU seconds for RLT to 16.4 and 19.6 CPU seconds, respectively, for the concurrent and sequential constraint generation routines. These results indicate that a judicious choice of variables for generating the grid-factor-based restrictions can potentially play a key role in the success of RLT+GF within a branch-and-bound approach.

Next, we compare the incumbent solution values obtained at the root node by RLT and by RLT+GF using the concurrent and sequential constraint generation routines. The incumbent so-

Table 5.4: Quality of incumbent solutions at the root node using RLT and RLT+GF.

		# within 1% of UB^*	# other feas. (Mean % opt. gap)	# that produced no incumbent solution
RLT		5	3 (4.6%)	7
Concurrent	$G=1$	8	2 (6.3%)	5
	$G=2$	8	2 (5.2%)	5
	$G=3$	7	4 (5.7%)	4
Sequential	$G=1$	6	5 (4.9%)	4
	$G=2$	9	4 (5.7%)	2
	$G=3$	8	3 (4.8%)	4

lutions were derived by running SNOPT-Version 7 (Gill et al. [26]) with the optimal solution to the current node's LP relaxation used as a warm-start solution. We implemented SNOPT with its default settings, except that the number of major iterations was limited to 25, and the feasibility of the solution returned by SNOPT was ascertained using a tolerance of 0.0005 for the nonlinear constraints. As displayed in Table 5.4, the basic RLT algorithm failed to produce an incumbent solution at the root node for seven out of 15 instances, whereas this number varied between two-five for the RLT+GF variants. For five of the 15 instances, the incumbent solution values obtained using RLT were within 1% of UB^*, while the RLT+GF variants produced incumbent solutions within 1% of UB^* for six-nine instances.

Based on the overall results obtained for this root node analysis, we continue our computational testing of RLT+GF within the proposed branch-and-bound algorithm using only (at most) a single grid-point per variable.

5.6.2 Performance within a Branch-and-Bound Algorithm

We now experiment with the branch-and-bound algorithm described in Section 5.5 for solving polynomial programming problems, when implemented with and without the proposed grid-factor constraints, where the latter utilize $G_j \leq 1, \forall j \in \mathcal{N}$. In the present context, we initialize the root node analysis by invoking the local solver SNOPT with $x_j = (l_j + u_j)/2, \forall j \in \mathcal{N}$, being used as a starting solution, prior to solving the node-zero relaxation. Table 5.5 presents the average computational times obtained over five instances of each size for RLT, and the % reduction in these average CPU times for RLT+GF using the concurrent and sequential constraint generation

routines in concert with the equal distance, incumbent, optimal-to-parent, and optimal-to-self grid-point selection rules, as well as using the reduced relaxations derived by invoking the positive-discrepancy strategy.

Among the 15 test instances solved, we observed two early termination cases for some of the algorithmic compositions. The results for these two cases are not included in Table 5.5, but are addressed separately below. As displayed in Table 5.5, all the variants of RLT+GF produced better results than RLT implemented as a stand-alone approach, except for the sequential constraint generation routine, which increased the average CPU time by 45%. The computational effort for RLT+GF using the concurrent constraint generation routine and the equal distance rule was reduced by 11.5% as compared with RLT. The main drawback of this variant of RLT+GF is the relatively high CPU time for solving the LP relaxations due to the large number of grid-factor-based constraints. On the other hand, using the optimal-to-parent rule instead, the computational effort decreased by 21.6% as compared with RLT, because of the reduced sizes of the corresponding relaxations (see Table 5.3). The incumbent rule implemented with the positive-discrepancy strategy reduced the average CPU time relative to RLT by 7.1%. Overall, the positive-discrepancy reduction strategy, applied in conjunction with the equal distance and optimal-to-parent rules, produced the best results by decreasing the average CPU times by 23.6% and 23.1%, respectively, in comparison with RLT.

Table 5.5: % reduction in the average CPU times for RLT+GF-based branch-and-bound procedures with respect to RLT.

	RLT	RLT + GF Concurrent		RLT + GF (Positive-discrepancy) Concurrent			Sequential
(δ, n, R)	CPU time (in sec.)	Equal distance	Opt.-to -parent	Equal distance	Incumbent	Opt.-to -parent	Opt.-to -self
(3, 19, 12)	203	26%	26.2%	26.7%	3.5%	25.8%	−50.4%
(4, 12, 6)	225	7.9%	26.2%	23.9%	−0.8%	27.2%	−22.3%
(5, 11, 4)	847	0.8%	12.3%	20.3%	18.5%	16.4%	−62.3%
Average	**425**	**11.5**%	**21.6**%	**23.6**%	**7.1**%	**23.1**%	**−45**%

For the two premature termination instances of respective sizes (3, 19, 12) and (5, 11, 4), we display separately for each method in the upper part of Table 5.6 the number of early terminations (over these two instances with a 3600 CPU seconds time limit), the average % optimality gap obtained (over both the test cases with a specified optimality tolerance of 1% and computed as $100\frac{UB-LB}{UB}$, where LB and UB respectively denote the lower and upper bounds obtained), and

the average CPU time over both the test cases. Note that the computational effort for some of the degree-five instances are larger than the preset time limit of 3600 CPU seconds due to the excess times required for solving the corresponding LP relaxations. Also, for the instances that terminated within the set optimality tolerance of 1%, we simply use 1% in the computations of the average % optimality gaps in Table 5.6, as opposed to using the actual gap upon termination, which is often less than 1%. Hence, the % optimality gap values presented in Table 5.6 are conservative. Overall, based on Tables 5.5 and 5.6, the best performing variant of the RLT+GF approach uses the concurrent constraint generation routine with the optimal-to-parent grid-point rule in coordination with the positive-discrepancy reduction strategy (designated **RLT+GF*** henceforth). Although the reduction in the average CPU time is slightly higher using the equal distance rule instead of the optimal-to-parent rule in this composition, we see that the latter displays a more consistent performance when the two early termination cases are included in the analysis. The early termination instance of size (3, 19, 12) was successfully optimized by RLT+GF* within 3478 CPU seconds, whereas RLT achieved an optimality gap of 1.53% upon premature termination after 3601 CPU seconds. The same instance was solved within 4696 CPU seconds by RLT when no time limit was imposed. On the other hand, the degree-five instance (5, 11, 4) was solved within 3588 and 3006 CPU seconds using RLT and RLT+GF*, respectively. Hence, without time limit restrictions, RLT+GF* reduced the average computational effort over all the 15 instances by 21.1% in comparison with RLT.

Table 5.6: Performances of RLT- and RLT+GF-based branch-and-bound procedures for the two early termination cases of sizes (3, 19, 12) and (5, 11, 4) using the best-first and the breadth-first strategies.

		RLT	RLT + GF Concurrent		RLT + GF (Positive-discrepancy) Concurrent			Sequential
			Equal distance	Opt.-to -parent	Equal distance	Incumbent	Opt.-to -parent	Opt.-to -self
Best- First Strategy	# early term:	1	2	0	1	1	0	2
	Avg. % opt. gap:	1.26%	1.51%	1%	1.1%	1.2%	1%	2.71%
	Avg. CPU time:	3595	3621	3443	3590	3581	3242	3604
Breadth- First Strategy	# early term:	1	1	0	1	1	0	2
	Avg. % opt. gap:	3.02%	1.83%	1%	1.22%	2.84%	1%	7.33%
	Avg. CPU time:	3591	3594	3424	3589	3607	3212	3630

Motivated by the study of Paulavičius et al. [60], we also employed the breadth-first node selection strategy within the branch-and-bound algorithm to solve the 15 test cases. The results indicated that the number of nodes explored and the computational effort did not differ signifi-

cantly in comparison with the best-first strategy whenever the solution process terminated with the required accuracy. On the other hand, as displayed in Table 5.6, we observed that the % optimality gap using the breadth-first approach was higher for all early termination cases in comparison with the results for the best-first strategy.

To provide further insights, we next present results for some alternative implementations of branch-and-bound procedures and constraint generation rules. First of all, when we employed the dual simplex method with the steepest-edge pricing strategy to solve LP relaxations, the respective average CPU times over the 13 instances of Table 5.5 for RLT and RLT+GF* increased to 1114 and 943 CPU seconds versus 410 and 327 CPU seconds as reported using the barrier algorithm, thus resulting in an increase in effort by 172% and 188%, respectively. Next, recall that we define $\hat{\mathcal{N}} \equiv \{j \in \mathcal{N} : G_j \geq 1\}$ in Section 5.3 and generate only the bound-grid-factor constraints identified in (5.21). As a point of interest, we generated the bound-grid-factors according to (5.5) while only restricting J_3 to correspond to the variables in $\hat{\mathcal{N}}$ whenever $J_3 \neq \emptyset$. The average computational times increased with this modification for all constraint generation rules (except when using the equal distance rule). In particular, for RLT+GF*, the resulting average CPU time over 14 instances increased by 6.8% in comparison with the foregoing implementation, while the remaining instance of size (5, 11, 4) achieved an optimality gap of 1.01% within 3636 CPU seconds, whereas this problem was optimized within 3006 CPU seconds using the foregoing implementation.

Table 5.7: % reduction in average CPU times for RLT+GF with exactly $\lfloor \delta/2 \rfloor$ grid-factors in each grid-factor-based constraint with respect to RLT.

(δ, n, R)	RLT	RLT + GF Concurrent		RLT + GF (Positive-discrepancy) Concurrent			Sequential
	CPU time (in sec.)	Equal distance	Opt.-to -parent	Equal distance	Incumbent	Opt.-to -parent	Opt.-to -self
(4, 12, 6)	225	11%	0.8%	9.8%	−1.9%	1.6%	−50.2%
(5, 11, 4)	847	−13.9%	−28.9%	−14.2%	−8.8%	−28.6%	−94%
Average	**536**	**−1.4%**	**−14%**	**−2.2%**	**−5.3%**	**−13.5%**	**−72.1%**

Differing from our approach, the implementation in Sherali and Tuncbilek [74] generated grid-factor product constraints using combinations of grid-factors taken exactly $\lfloor \delta/2 \rfloor$ at a time (albeit for simple box-constrained univariate problems). This modification substantially changes both the size and relative tightness of the resulting relaxations. For instance, when the constraints in (5.21) that involve grid-factors ($J_3 \neq \emptyset$) can only have precisely $\lfloor \delta/2 \rfloor$ grid-factors, the number of related constraints in (5.21) decreases from 3678 and 23716 to 78 and 1452 for degree-four

and five problems, respectively, when the concurrent constraint generation routine is utilized with a single grid-point per variable located using the equal distance rule. Note that we can have at most one grid-factor involved in (5.21) for quadratic and cubic problems in general; hence, the two strategies produce the same results for polynomial programs of degree less than or equal to three. However, for problems of degree four and five, the relative performances using grid-factor-based constraints having up to $\lfloor \delta/2 \rfloor$ versus exactly $\lfloor \delta/2 \rfloor$ grid-factors can vary. Table 5.7 presents the results for RLT+GF using exactly $\lfloor \delta/2 \rfloor$ grid-factors in each grid-factor-based constraints of the type (5.21) whenever $J_3 \neq \emptyset$, over the same set of instances pertaining to Table 5.5. Although some variants of RLT+GF yet outperformed RLT for degree-four problems, the overall effort for the RLT+GF variants increased in comparison to RLT for degree-five problems, when using only precisely $\lfloor \delta/2 \rfloor$ grid-factors within (5.21) whenever $J_3 \neq \emptyset$. This sharply contrasts with the relative efficiency of RLT+GF when using up to $\lfloor \delta/2 \rfloor$ grid-factors in (5.21), as proposed herein (see Table 5.5).

We also utilized BARON-Version 9.0.6 (Tawarmalani and Sahinidis [84]) with default settings while using the same tolerance $\varepsilon = 0.01$ and a time limit of 3600 CPU seconds. BARON terminated prematurely in 14 out of 15 instances with an average optimality gap of 45.9%, whereas RLT+GF* solved all these 14 instances to optimality within 766.2 CPU seconds on average. The remaining instance was optimized by BARON and RLT+GF* in 1377 and 10 CPU seconds, respectively.

Chapter 6

Reduced RLT Representations for Nonconvex Polynomial Programs

This chapter explores equivalent, reduced size Reformulation-Linearization Technique (RLT)-based formulations for polynomial programming problems. Utilizing a basis partitioning scheme for an embedded linear equality subsystem, we show that a strict subset of RLT defining equalities imply the remaining ones. Applying this result, we derive significantly reduced RLT representations and develop certain coherent associated branching rules that assure convergence to a global optimum, along with static as well as dynamic basis selection strategies to implement the proposed procedure. In addition, we enhance the RLT relaxations with v-semidefinite cuts, which are empirically shown to further improve the relative performance of the reduced RLT method over the usual RLT approach. We present computational results for randomly generated instances to test the different proposed reduction strategies and to demonstrate the improvement in overall computational effort when such reduced RLT mechanisms are employed.

6.1 Introduction

The *Reformulation-Linearization Technique* **(RLT)** offers a unified framework for solving nonconvex discrete and continuous optimization problems (see Sherali and Adams [68]). In this chapter,

we focus on applying the RLT to general polynomial programming problems, and explore the generation of equivalent, reduced size RLT representations, augmented with additional valid inequalities, in order to enhance the solvability of such problems via a branch-and-bound algorithm. Accordingly, consider the following polynomial program **PP** of order $\delta \geq 2$, where notationally, given any set $\mathscr{S} \subseteq \mathscr{N} \equiv \{1,\ldots,n\}$, we let $\mathscr{S}^d$ denote a *multi-set of order d*, which is comprised of distinct combinations of indices (arranged in nondecreasing order) that belong to the Cartesian product $\mathscr{S} \times \ldots \times \mathscr{S}$, where the latter involves d repetitions of $\mathscr{S}$. Hence, in particular, $\mathscr{S}^1 \equiv \mathscr{S}$.

$$\textbf{PP:}\quad \text{Minimize}\quad \phi_0(x) \tag{6.1a}$$

subject to

$$\phi_r(x) \geq \beta_r, \forall r = 1,\ldots,R_1 \tag{6.1b}$$

$$\phi_r(x) = \beta_r, \forall r = R_1+1,\ldots,R \tag{6.1c}$$

$$Ax = b \tag{6.1d}$$

$$x \in \Omega \equiv \{0 \leq l_j \leq x_j \leq u_j < \infty, \forall j \in \mathscr{N}\}, \tag{6.1e}$$

where

$$\phi_r(x) \equiv \sum_{t \in T_r} \alpha_{rt} \Big[\prod_{j \in J_{rt}} x_j \Big], \text{ for } r = 0,\ldots,R,$$

and where T_r is an index set for the terms defining $\phi_r(.)$, with α_{rt} being real coefficients associated with the monomials $\prod_{j \in J_{rt}} x_j,\ \forall t \in T_r,\ r = 0,\ldots,R$, where each $J_{rt} \subseteq \cup_{d=1}^{\delta} \mathscr{N}^d$. Here, we have especially identified the presence of a linear equality system (6.1d), where A is $m \times n$ of rank $m < n$, and where (6.1c) then accommodates any other nonlinear equality restrictions defining the model formulation.

Following the RLT procedure described by Sherali and Tuncbilek [71], given the lower and upper bounds in (6.1e), we define the *bound-factors* $(x_j - l_j) \geq 0$ and $(u_j - x_j) \geq 0,\ \forall j \in \mathscr{N}$, and generate the corresponding *bound-factor product (RLT) constraints* composed by taking the products of the bound-factors δ at a time (including repetitions) as follows:

$$\prod_{j \in J_1} (x_j - l_j) \prod_{j \in J_2} (u_j - x_j) \geq 0, \forall (J_1 \cup J_2) \subseteq \mathscr{N}^{\delta}. \tag{6.2}$$

These implied $\binom{2n+\delta-1}{\delta}$ restrictions are then appended to Problem PP, along with any other (op-

tionally generated) RLT constraints obtained by multiplying the original inequality constraints in (6.1b) with themselves as well as with bound-factors, while preserving the degree of the resulting augmented polynomial program as δ. Similarly, the equality constraints (6.1c) and (6.1d) can be utilized to generate RLT constraints by multiplying each of them with distinct monomials of the form $\prod_{j\in J} x_j$ such that the resulting restrictions are of degree less than or equal to δ. It is worthwhile noting that whereas appending Problem PP with such RLT constraints other than (6.2) potentially tightens the relaxation, they are not required for establishing the convergence of the algorithm proposed by Sherali and Tuncbilek [71] to solve Problem PP to global optimality. Moreover, we limit the generation of RLT polynomial constraints to degree δ only to curtail the size of the resulting relaxation, although higher order restrictions can further tighten the derived representation.

Next, we linearize the augmented polynomial program by replacing each distinct monomial $\prod_{j\in J} x_j$ with a new (RLT) variable X_J, for $J \subseteq \cup_{d=2}^{\delta} \mathcal{N}^d$. Note that this produces $\binom{n+\delta}{\delta} - (n+1)$ such additional so-called *RLT variables*. The corresponding RLT-based *linear programming* (LP) *relaxation* is constructed via this substitution process upon dropping from the formulation the defining identities $X_J = \prod_{j\in J} x_j, \forall J$. At each node of the branch-and-bound tree, a lower bound is thus computed via the RLT-based LP relaxation associated with the revised sub-hyperrectangle Ω that defines the current node problem. Accordingly, at the top of the loop in this branch-and-bound process, we select a node having the least lower bound for further partitioning. Denoting $(\bar{x}, \bar{X})$ as the LP relaxation solution obtained for this node, we select a branching variable index as

$$j^* \in \arg\max_{j\in\mathcal{N}} \theta_j,$$

where θ_j is computed as follows:

$$\theta_j \equiv \sum_{d=1}^{\delta-1} \sum_{J\in\mathcal{N}^d} \left|\bar{X}_{J\cup j} - \bar{X}_J \bar{x}_j\right|, \forall j \in \mathcal{N}.$$

We then partition the selected interval $[l_{j^*}, u_{j^*}]$ at the current LP relaxation solution value $\bar{x}_{j^*}$ according to the dichotomy that $\{x_{j^*} \in [l_{j^*}, \bar{x}_{j^*}]\} \vee \{x_{j^*} \in [\bar{x}_{j^*}, u_{j^*}]\}$ provided that $\min\{\bar{x}_{j^*} - l_{j^*}, u_{j^*} - \bar{x}_{j^*}\} \geq 0.05(u_{j^*} - l_{j^*})$; else, we split the interval at its mid-point. This particular choice of branching variable selection and partitioning strategy preserves the arguments of the convergence proof presented by Sherali and Tuncbilek [71]. Whenever the lower bound LB computed for any node subproblem satisfies $(v^* - LB) \leq \varepsilon|v^*|$, where v^* is the incumbent solution value and ε is a speci-

fied optimality gap tolerance, we fathom the corresponding node. The branch-and-bound algorithm terminates when no active node exists.

Notwithstanding the standard RLT constraint generation process, the resulting LP relaxation can be further tightened by utilizing *v-semidefinite cuts* (or simply *SDP cuts*) as described by Sherali and Fraticelli [70] and Sherali et al. [76]. These cuts are generated by imposing positive semidefinite restrictions on dyadic variable-product matrices composed as $M = [vv^t]_L \succeq 0$, where $[.]_L$ denotes the linearization of $[.]$ via the RLT variable substitution process, and where $v \equiv [1, \{\prod_{j\in J} x_j : J \subseteq \mathscr{J}\}]$ for certain appropriately defined sets $\mathscr{J}$ involving selected monomials of order up to $\lfloor \delta/2 \rfloor$ (see Sherali et al. [76]). In this strategy, having solved an RLT-based LP relaxation, and letting $\bar{M}$ denote the matrix M evaluated at the resulting solution, in case $\bar{M}$ is not positive semidefinite, a suitable unit-norm vector $\bar{\gamma}$ is derived (in polynomial time) having $\bar{\gamma}^T \bar{M} \bar{\gamma} < 0$, which yields an associated *v-semidefinite cut* $[(\bar{\gamma}^T v)^2]_L \geq 0$. Incorporating particular classes of such cuts can significantly enhance the performance of the RLT-based algorithm for solving polynomial programming programs as demonstrated by Sherali et al. [76].

The present chapter makes the following specific contributions. First, we introduce equivalent, reduced size RLT formulations for polynomial programming problems that *directly* utilize bases of the matrix A as opposed to bases of substantially larger companion systems as adopted by Liberti [46] and Cafieri et al. [15]. Second, we design and test several static and dynamic basis selection methods to implement the proposed reduction strategy. Third, we enhance the relaxations via v-semidefinite cuts along with the developed basis reduction mechanisms within a branch-and-bound algorithm to solve polynomial programs to global optimality, and we provide extensive computational results that demonstrate a significant performance improvement using the proposed approach.

The remainder of this chapter is organized as follows. To set the stage and facilitate the presentation, Section 6.2 first considers quadratic polynomial programming problems having an embedded linear equality system, and presents the proposed equivalent, reduced RLT representation for this structured problem. Section 6.3 then extends the theory to address more general higher order polynomial programs, which is the main focus of this paper. Section 6.4 describes several static and dynamic basis selection methods for constructing such reduced representations, and computational results are presented in Section 6.5 to test the different relaxation strategies, which also prompts the design of a particularly effective hybrid methodology.

6.2 Quadratic Polynomial Programming Problems

Consider the following nonconvex quadratic polynomial programming problem having an embedded equality subsystem:

$$\begin{aligned}
\textbf{P:}\quad & \text{Minimize} && c_0^T x + x^T H_0 x && \text{(6.4a)}\\
& \text{subject to} && && \\
& && c_i^T x + x^T H_i x \le h_i, \forall i = 1, \ldots, Q && \text{(6.4b)}\\
& && Ax = b && \text{(6.4c)}\\
& && 0 \le l_j \le x_j \le u_j < \infty, \forall j \in \mathcal{N} \equiv \{1, \ldots, n\}, && \text{(6.4d)}
\end{aligned}$$

where A is $m \times n$ of rank $m < n$, and where $c_i \in \mathbb{R}^n$ and $H_i \in \mathbb{R}^{n \times n}$, $\forall i = 0, 1, \ldots, Q$. Problem P arises in several applications and is of interest in its own right (see Floudas and Visweswaran [21]), but such a representation might also be an equivalent quadratic reformulation of a more general polynomial program (see Shor [79] and Sherali and Tuncbilek [73]).

Given a basis B of A for the linear system of equations (6.4c), we adopt the familiar partitioning of x into basic and nonbasic variables x_B and x_N, respectively, to write (6.4c) in the form:

$$Bx_B + Nx_N = b, \tag{6.5}$$

where B represents the columns of the x_B-variables, and N represents the columns of the x_N-variables. Let J_B and J_N denote the respective index sets of the basic and nonbasic variables, where $J_B \cup J_N = \mathcal{N}$.

Now, following the RLT process, we multiply (6.5) by each x_j for $j \in \mathcal{N}$, and linearize the resulting system using the substitution

$$X_{ij} = x_i x_j, \ \forall 1 \le i \le j \le n. \tag{6.6}$$

Denoting by $[.]_L$ the linearization of $[.]$ under (6.6), and letting

$$X_{(ij)} = \begin{cases} X_{ij} & \text{if } i \le j \\ X_{ji} & \text{if } i > j\ , \end{cases} \tag{6.7a}$$

with $X \equiv X_{(ij)} \in \mathbb{R}^{n(n+1)/2}$, and defining

$$X_{(Bj)} \equiv \left[x_B x_j\right]_L,\ X_{(Nj)} \equiv \left[x_N x_j\right]_L, \text{ and } X_{(.j)} \equiv \left[x x_j\right]_L,\ \forall j \in \mathscr{N}, \tag{6.7b}$$

the foregoing RLT operation yields the following system:

$$BX_{(Bj)} + NX_{(Nj)} = bx_j, \forall j \in \mathscr{N}. \tag{6.8}$$

Proposition 6.1. Let the system $Ax = b$ be partitioned according to (6.5) for any basis B of A, and define

$$Z = \{(x,X) : (6.4c), (6.8), \text{ and } X_{ij} = x_i x_j,\ \forall i \leq j \text{ with } i, j \in J_N\}. \tag{6.9}$$

Then, we have (6.6) holding true for any $(x,X) \in Z$.

Proof. Let $(x,X) \in Z$. Then, from (6.8) and (6.9), we get

$$BX_{(Bj)} = (b - Nx_N)x_j, \forall j \in J_N. \tag{6.10}$$

But (6.5) implies that $x_B = B^{-1}(b - Nx_N)$, which together with (6.10) yields

$$X_{(Bj)} = x_B x_j,\ \forall j \in J_N, \text{ i.e., } X_{(ij)} = x_i x_j,\ \forall i \in J_B,\ j \in J_N. \tag{6.11}$$

Furthermore, from (6.8) written for $j \in J_B$, and noting (6.11), we get that,

$$BX_{(Bj)} = (b - Nx_N)x_j, \forall j \in J_B. \tag{6.12}$$

Again, together with (6.5), this implies that

$$X_{(Bj)} = x_B x_j,\ \forall j \in J_B, \text{ i.e., } X_{(ij)} = x_i x_j,\ \forall i, j \in J_B. \tag{6.13}$$

The result now follows by observing (6.9), (6.11), and (6.13). □

Remark 6.1. Noting that $|J_N| = (n-m)$ from (6.4c) and (6.5), Proposition 6.1 asserts that subject to (6.4c) (or (6.5)) and (6.8), if we enforce (6.6) for just the $(n-m)(n-m+1)/2$ identities corresponding to $i \leq j$ in J_N, then the remaining identities in (6.6) will automatically hold true. This concept can be used to curtail RLT relaxations and branching decisions, and improves upon the strategy expounded in Liberti [46, 47] and Liberti and Pantelides [48] by permitting a reformula-

tion based *directly* on the selection of suitable bases B of A, rather than requiring the examination of bases of an enlarged $(mn \times n^2)$ companion system as developed in the latter works. In the same spirit, reduced RLT relaxations for polynomial programs employing bases of a linear companion system of size $m[\binom{n+\delta-1}{\delta-1} - 1] \times [\binom{n+\delta}{\delta} - (n+1)]$ are proposed by Cafieri et al. [15]. A more compact generalization of Proposition 6.1 to polynomial programming problems is likewise presented in Section 6.3 below. □

For the sake of interest, we note that the following more general result holds true, although we shall focus on Proposition 6.1 to achieve computational expediency.

Proposition 6.2. There exists a partitioning of $X \in \mathbb{R}^{n(n+1)/2}$ according to $X = (X_{Q_1}, X_{Q_2})$, where $X_{Q_1} \in \mathbb{R}^{m(n-m)+m(m+1)/2}$ and $X_{Q_2} \in \mathbb{R}^{(n-m)(n-m+1)/2}$ are such that the columns of X_{Q_1} in (6.8) are linearly independent. Moreover, for any such partitioning of X, any feasible solution to the system

$$Ax = b,\ AX_{(.j)} = bx_j,\ \forall j \in \mathscr{N}, \tag{6.14}$$

along with the identities (6.6) for the X_{Q_2}-variables, will also automatically have (6.6) holding true for the X_{Q_1}-variables.

Proof. First of all, note that a partitioning of X as stated in the proposition exists by Proposition 6.1 and Remark 6.1, as for example, by selecting any basis B of A and letting X_{Q_1} and X_{Q_2} be respectively composed of the sets of variables $\{X_{(Bj)}$ for $j \in J_N; X_{(Bj)}$ for $j \in J_B\}$, and $\{X_{(Nj)}$ for $j \in J_N\}$. Now, for any given feasible solution x to (6.4c), let the X_{Q_2}-variables be determined according to (6.6), and consider the resulting residual system (6.8) in the X_{Q_1}-variables. Since the columns of X_{Q_1} in (6.8) are linearly independent, this system either has a unique solution or has no solution. But by construction, fixing the X_{Q_1}-variables according to (6.6) yields a feasible solution to (6.8), and therefore this must be the unique completion of the solution to the residual system in (6.8) or (6.14). □

Now, defining the variables $X \in \mathbb{R}^{n(n+1)/2}$ according to (6.6) and adopting the notation (6.7), we can rewrite (6.4) as follows, where $g_i \in \mathbb{R}^{n(n+1)/2}$ for $i = 0, 1, \ldots, Q$ are appropriate vectors that represent the pure quadratic forms appearing in (6.4a, 6.4b), and where we have included the RLT constraints (6.14) obtained by multiplying $Ax = b$ with $x_j, \forall j \in \mathscr{N}$, within (6.15d), as well as the RLT bound-factor constraints (6.2) within (6.15e). (Note that the algebraic notation in

(6.15e) represents the quadruple bound-factor product relationships $[(x_i - l_i)(x_j - l_j)]_L \geq 0, [(x_i - l_i)(u_j - x_j)]_L \geq 0, [(u_i - x_i)(x_j - l_j)]_L \geq 0$, and $[(u_i - x_i)(u_j - x_j)]_L \geq 0$ linearized under (6.6), $\forall 1 \leq i \leq j \leq n$, where it is understood that for $i = j$, we retain only one of the duplicating second and third product restrictions.)

P1: Minimize $c_0^T x + g_0^T X$ (6.15a)

subject to

$$c_i^T x + g_i^T X \leq h_i, \forall i = 1, \ldots, Q \quad (6.15b)$$

$$Ax = b \quad (6.15c)$$

$$AX_{(.j)} = bx_j, \ \forall j \in \mathcal{N} \quad (6.15d)$$

$$\left[(l_i \leq x_i \leq u_i) * (l_j \leq x_j \leq u_j)\right]_L \geq 0, \ \forall 1 \leq i \leq j \leq n \quad (6.15e)$$

$$l \leq x \leq u \quad (6.15f)$$

$$X_{ij} = x_i x_j, \ \forall 1 \leq i \leq j \leq n. \quad (6.15g)$$

Next, following the concept discussed in Proposition 6.1, we partition $Ax = b$ according to (6.5), and correspondingly derive a reduced form of Problem P1 as stated below by enforcing (6.15g) only for $i \leq j$ in J_N, and also retaining (6.15e) corresponding to only these same indices $i \leq j$ in J_N, and where we have additionally included the implied bounding constraints $l_i l_j \leq X_{ij} \leq u_i u_j, \ \forall 1 \leq i \leq j \leq n$ in (6.16f) for tightening the underlying LP relaxation obtained upon deleting (6.16g).

P2: Minimize $c_0^T x + g_0^T X$ (6.16a)

subject to

$$c_i^T x + g_i^T X \leq h_i, \forall i = 1, \ldots, Q \quad (6.16b)$$

$$Ax = b \quad (6.16c)$$

$$BX_{(Bj)} + NX_{(Nj)} = bx_j, \ \forall j \in \mathcal{N} \quad (6.16d)$$

$$\left[(l_i \leq x_i \leq u_i) * (l_j \leq x_j \leq u_j)\right]_L \geq 0, \ \forall i \leq j, \text{ with } i, j \in J_N \quad (6.16e)$$

$$l \leq x \leq u, \ l_i l_j \leq X_{ij} \leq u_i u_j, \ \forall 1 \leq i \leq j \leq n \quad (6.16f)$$

$$X_{ij} = x_i x_j, \ \forall i \leq j \text{ with } i, j \in J_N. \quad (6.16g)$$

Remark 6.2. Note that the equality constraints of type $Ax = b$ can also be used to directly eliminate the basic variables x_B for any given basis B via the substitution $x_B = B^{-1}(b - Nx_N)$, and we can then correspondingly apply the regular RLT process described by Sherali and Tuncbilek [71]

to the resulting problem in the space of the $n-m$ nonbasic variables. In this context, Sherali and Tuncbilek [72] have shown that the RLT procedure is invariant under such affine transformations. However, a potential drawback of implementing this transformation is that the sparse nonlinear constraints (as well as nonlinear objective terms) may possibly become dense and the resulting linear programming relaxations might become relatively more difficult to solve. In our computations, we compare the relative efforts for solving problems using the proposed reduced RLT procedure as well as by applying RLT to the reduced nonbasic variable space problem representation. □

Let RLT(P1) denote the RLT-based branch-and-bound algorithm described by Sherali and Tuncbilek [71, 72] as applied to Problem P1, where lower bounds are computed via the LP relaxation (6.15a -6.15f), and where partitioning is performed on the hyperrectangle defined by (6.15f). In addition, denote by RLT_{SDP}(P1) the foregoing solution process in which we also incorporate semidefinite programming (SDP)-based cuts as in Sherali and Fraticelli [70] and Sherali et al. [76]. Likewise, define RLT(P2) and RLT_{SDP}(P2) with respect to Problem P2, where in particular, the branching is now performed only on the sub-hyperrectangle defined by $l_j \le x_j \le u_j$, $\forall j \in J_N$. By Proposition 6.1 and the theory expounded by Sherali and Tuncbilek [71], we are assured (infinite) convergence to a global optimum by this partial partitioning process.

Remark 6.3. Note that the linear programming relaxation $\overline{\text{P2}}$ of Problem P2 can be tightened by including (6.15e) in lieu of the constraints (6.16e) and the implied bound restrictions in (6.16f). This would be similar to solving P1 itself via an RLT approach, but while partitioning on just the nonbasic variable intervals. We refer to this strategy as RLT^{NB}(P1) and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(P1), respectively implemented without and with SDP cuts. Moreover, as alluded earlier, we mention (but do not adopt in our implementation) that we can further strengthen $\overline{\text{P2}}$ by incorporating RLT bound-factor product constraints of order $\delta' > \delta$; in this context, note that the constraints in (6.16e) are implied by bound-factor constraints generated via all $(J_1 \cup J_2) \subseteq J_N^{\delta'}$ as proven by Sherali and Tuncbilek [71]. However, this can significantly increase the size of the resulting relaxations and needs a careful design of additional filtering mechanisms. The identities $x_B = B^{-1}b - B^{-1}Nx_N$ can be used to further tighten the bounds on the basic variables, x_B, and hence the implied bounds on the X-variables in (6.16f), by consecutively minimizing and maximizing $B_i^{-1}b - B_i^{-1}Nx_N$ subject to (6.16b)-(6.16f), $\forall i = 1, \ldots, m$, where B_i is the i^{th} row of B. Hence, whenever a node is partitioned, we can perform such a *range reduction* for the basic variables (see also Caprara and Locatelli [16], Ryoo and Sahinidis [65], Sherali and Tuncbilek [72], Sherali and Tuncbilek [74]). However, we do not implement this additional strategy in order to assess the independent effect of the approach

proposed herein, but we recommend its consideration for future research. □

6.3 Polynomial Programming Problems

In this section, we generalize the theory and strategies of Section 6.2 to address nonconvex polynomial programming problems. For the polynomial program **PP** of order δ as given by (6.1a-6.1e), as part of the RLT reformulation process, we multiply the linear equality constraints $Ax = b$ in (6.1d) with distinct monomials $\prod_{j\in J} x_j$ of order $d = 1,\ldots,\delta-1$. Hence, introducing the RLT-variables

$$X_J = \prod_{j\in J} x_j,\ \forall J \subseteq \mathscr{N}^d, d = 2,\ldots,\delta, \tag{6.17}$$

where we also designate $X_j \equiv x_j$ when $J = \{j\} \subseteq \mathscr{N}$, we impose the linearized RLT constraints:

$$\left[(Ax = b) \times \prod_{j\in J} x_j\right]_L,\ \text{yielding } AX_{(.J)} = bX_J,\ \forall J \subseteq \mathscr{N}^d, d = 1,\ldots,\delta-1,$$

where $X_{(.J)} \equiv (X_{(1,\,J)}, X_{(2,\,J)},\ldots,X_{(n,\,J)})$, and where (j,J) is the multi-set that combines j with the indices in J in nondecreasing order. Partitioning $Ax = b$ according to $Bx_B + Nx_N = b$ as before, where B is a basis of A, and using notation similar to that in (6.8), the foregoing set of equations can be written in the corresponding partitioned form as follows:

$$BX_{(BJ)} + NX_{(NJ)} = bX_J,\ \forall J \subseteq \mathscr{N}^d, d = 1,\ldots,\delta-1. \tag{6.18}$$

Furthermore, notationally, given a multi-set $J \subseteq \mathscr{N}^d$ for any $d \in \{2,\ldots,\delta\}$, define $J_B \overline{\cap} J$ as the (multi-) subset of J that contains only the basic variable indices.

Proposition 6.3. Let the equality system $Ax = b$ in (6.1d) be partitioned as $Bx_B + Nx_N = b$ for any basis B of A, and define

$$Z = \left\{(x,X) : (6.1\text{d}), (6.18),\ \text{and } X_J = \prod_{j\in J} x_j,\ \forall J \subseteq J_N^d,\ \text{for } d = 2,\ldots,\delta\right\}. \tag{6.19}$$

Then, we have (6.17) holding true for any $(x,X) \in Z$.

Proof. We establish this result by induction on $d = 2, \ldots, \delta$. For the case $d = 2$, we have (6.17) holding true for any $(x,X) \in Z$ by Proposition 6.1, using (6.1d), (6.18) written for $d = 1$, and $X_J = \prod_{j\in J} x_j$ for $J \subseteq J_N^2$ from (6.19).

Hence, assume that the result is true for $\{2, \ldots, d\}$, where $2 \leq d \leq \delta - 1$, i.e.,

$$X_J = \prod_{j\in J} x_j, \ \forall J \subseteq \mathscr{N}^\Delta, \Delta = 2, \ldots, d, \tag{6.20}$$

and examine the case of $d+1$.

Consider the RLT product constraints (6.18) for $J \subseteq J_N^d$. Since $X_J = \prod_{j\in J} x_j$ and $X_{(NJ)} = x_N \prod_{j\in J} x_j = x_N X_J, \ \forall J \subseteq J_N^d$ by (6.19), we get from (6.18) that

$$X_{(BJ)} = B^{-1}(b - Nx_N)X_J = x_B X_J = x_B \prod_{j\in J} x_j, \ \forall J \subseteq J_N^d.$$

This identity can be restated as follows:

$$X_J = \prod_{j\in J} x_j, \ \forall J \subseteq \mathscr{N}^{d+1} : |J_B \overline{\cap} J| = 1. \tag{6.21}$$

Next, consider (6.18) for $J \subseteq \mathscr{N}^d$ such that $|J_B \overline{\cap} J| = 1$. Noting that $X_J = \prod_{j\in J} x_j$, and $X_{(NJ)} = x_N \prod_{j\in J} x_j = x_N X_J, \ \forall J \subseteq \mathscr{N}^d$ with $|J_B \overline{\cap} J| = 1$ by (6.20) and (6.21), we get from (6.18) that

$$X_{(BJ)} = B^{-1}(b - Nx_N)X_J = x_B X_J = x_B \prod_{j\in J} x_j, \ \forall J \subseteq \mathscr{N}^d : |J_B \overline{\cap} J| = 1.$$

The foregoing identity essentially states that

$$X_J = \prod_{j\in J} x_j, \ \forall J \subseteq \mathscr{N}^{d+1} : |J_B \overline{\cap} J| = 2. \tag{6.22}$$

Continuing in this fashion, by considering (6.18) for $J \subseteq \mathscr{N}^d$ such that $|J_B \overline{\cap} J| = 2$, we get similar to (6.21) and (6.22) that (6.17) holds true for all $J \subseteq \mathscr{N}^{d+1}$ such that $|J_B \overline{\cap} J| = 3$. Finally, by considering (6.18) for $J \subseteq \mathscr{N}^d$ such that $|J_B \overline{\cap} J| = d$, we get that (6.17) holds true for all $J \subseteq \mathscr{N}^{d+1}$ such that $|J_B \overline{\cap} J| = d+1$, i.e., for the case where all indices are basic. Consequently, (6.17) holds true for the case $d+1$. □

Upon including the RLT restrictions (6.18) along with the bound-factor RLT constraints as in (6.23e) and (6.23f) below, and substituting X_J for the monomial $\prod_{j\in J} x_j, \forall J \subseteq \mathcal{N}^d, d = 2,\ldots,\delta$, we reformulate Problem PP as follows, where (6.23a), (6.23b), and (6.23c) are linearizations of (6.1a), (6.1b), and (6.1c), respectively:

PP1: Minimize $[\phi_0(x)]_L$ (6.23a)

subject to

$$[\phi_r(x)]_L \geq \beta_r, \forall r = 1,\ldots,R_1 \quad (6.23b)$$

$$[\phi_r(x)]_L = \beta_r, \forall r = R_1+1,\ldots,R \quad (6.23c)$$

$$Ax = b \quad (6.23d)$$

$$BX_{(BJ)} + NX_{(NJ)} = bX_J, \forall J \subseteq \mathcal{N}^d, d = 1,\ldots,\delta-1 \quad (6.23e)$$

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \forall (J_1 \cup J_2) \subseteq \mathcal{N}^\delta \quad (6.23f)$$

$$l \leq x \leq u \quad (6.23g)$$

$$X_J = \prod_{j\in J} x_j, \forall J \subseteq \mathcal{N}^d, d = 2,\ldots,\delta. \quad (6.23h)$$

Utilizing Proposition 6.3, we reduce the number of defining identities in (6.23h) by imposing them only for $J \subseteq J_N^d,\ \forall d = 2,\ldots,\delta$, within (6.24h) below. Likewise, since we are only required to enforce (6.24h) instead of (6.23h), we retain the RLT bound-factor product constraints in (6.23f) derived only for $(J_1 \cup J_2) \subseteq J_N^\delta$. Furthermore, similar to Problem P2, we explicitly include the implied interval bounding constraints for the X_J-variables as in (6.24g) for strengthening the underlying linear programming relaxation obtained upon deleting (6.24h).

PP2: Minimize $[\phi_0(x)]_L$ (6.24a)

subject to

$$[\phi_r(x)]_L \geq \beta_r, \forall r = 1,\ldots,R_1 \quad (6.24b)$$

$$[\phi_r(x)]_L = \beta_r, \forall r = R_1+1,\ldots,R \quad (6.24c)$$

$$Ax = b \quad (6.24d)$$

$$BX_{(BJ)} + NX_{(NJ)} = bX_J, \forall J \subseteq \mathcal{N}^d, d = 1,\ldots,\delta-1 \quad (6.24e)$$

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \forall (J_1 \cup J_2) \subseteq J_N^\delta \quad (6.24f)$$

$$l \leq x \leq u, \text{ and } \prod_{j\in J} l_j \leq X_J \leq \prod_{j\in J} u_j, \forall J \subseteq \mathcal{N}^d,\ d = 2,\ldots,\delta, |J_B \overline{\cap} J| \geq 1 \quad (6.24g)$$

$$X_J = \prod_{j \in J} x_j, \forall J \subseteq J_N^d, d = 2, \ldots, \delta. \tag{6.24h}$$

To enhance the foregoing reduced RLT-based relaxation, we append within (6.24f) in PP2 those additional constraints from (6.23f) for which the LP relaxation $\overline{\text{PP1}}$ of PP1 yields positive associated dual variables. Hence, in the branch-and-bound process for solving the reduced form PP2, we initially solve the LP relaxation $\overline{\text{PP1}}$ at the root node, and identify the subset of constraints from (6.23f) given by

$$J_{+duals} \equiv \{(J_1, J_2) : (J_1 \cup J_2) \subseteq \mathscr{N}^\delta \setminus J_N^\delta \text{ and } u_{(J_1,\, J_2)} > 0\},$$

where $u_{(J_1,\, J_2)}$ denotes the value of the dual variable associated with the corresponding constraint in (6.23f) at an optimal solution for $\overline{\text{PP1}}$. We then include the particular constraints from (6.23f) for $(J_1, J_2) \in J_{+duals}$ within the reduced relaxation PP2 for each node subproblem, appropriately updated based on the variable bounding intervals for the current node. Let us refer to this set of additional constraints at any node as C_{+duals}. Note that whenever we branch on some variable x_k, say, by splitting its current interval $l_k \leq x_k \leq u_k$ at the value $\bar{x}_k \in (l_l, u_k)$ as discussed in Section 6.1 to obtain the respective bounding restrictions $l_k \leq x_k \leq \bar{x}_k$ and $\bar{x}_k \leq x_k \leq u_k$ for the two child-nodes, we may not obtain a monotonic increase in the lower bound from the parent node to the child-nodes. The reason for this is that, because we have included only a selected subset of constraints (C_{+duals}) corresponding to $(J_1, J_2) \in J_{+duals}$ from (6.23f) within (6.24f) (after further revising the bound on x_k), the imposed bound-factor constraints do not necessarily imply all corresponding lower order bound-factor constraints as per the result in Sherali and Tuncbilek [71]. Hence, in order to preserve monotonicity in the lower bounds, we include certain additional sets of constraints from the parent node subproblem within the current node relaxation as explained next.

Whenever we branch on some variable x_k, then in addition to incorporating the constraints C_{+duals} within PP2 for the child-node subproblem, we also include those bound-factor constraints in C_{+duals} from the parent node formulation (in the same form therein) that involve the particular bound-factor that is associated with the branching restriction. An exception to this rule is when some latter type of constraint contains the corresponding branching restriction bound-factor only once. In this case, Proposition 6.4 below advocates the inclusion of a tighter constraint instead (i.e., Constraint (6.27) below in lieu of (6.25)).

Proposition 6.4. For any parent node problem, consider the following associated bound-factor

RLT constraint in C_{+duals} for some $(J_1, J_2) \in J_{+duals}$:

$$\left[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\right]_L \geq 0. \tag{6.25}$$

Suppose that we branch on a variable x_k, and consider the child-node for which we impose the additional restriction $x_k \geq \bar{x}_k > l_k$ (the case for the child-node corresponding to the restriction $x_k \leq \bar{x}_k < u_k$ is similar). Suppose further that we have $k \in J_1$ and $k \notin (J_1 \cup J_2) - \{k\}$, so that the child-node constraint for this particular $(J_1, J_2) \in J_{+duals}$ is given by

$$\left[(x_k - \bar{x}_k)\prod_{j\in J_1 - k}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\right]_L \geq 0. \tag{6.26}$$

Let $F_{\delta-1}(J_1 - k, J_2) \equiv \prod_{j\in J_1-k}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)$, and consider the corresponding bound-factor constraint:

$$[F_{\delta-1}(J_1 - k, J_2)]_L \geq 0. \tag{6.27}$$

Then, (6.26) and (6.27) imply (6.25).

Proof. Let (x, X) be feasible to (6.26) and (6.27). Then, the left-hand side of (6.25) evaluated at such a solution is given by:

$$\begin{aligned}
[(x_k - l_k)F_{\delta-1}(J_1 - k, J_2)]_L &= [x_k F_{\delta-1}(J_1 - k, J_2)]_L - l_k[F_{\delta-1}(J_1 - k, J_2)]_L \\
&\geq [x_k F_{\delta-1}(J_1 - k, J_2)]_L - \bar{x}_k[F_{\delta-1}(J_1 - k, J_2)]_L \\
&= [(x_k - \bar{x}_k)F_{\delta-1}(J_1 - k, J_2)]_L \\
&\geq 0.
\end{aligned}$$

□

As alluded above, in the case described in Proposition 6.4 (and similarly for the child-node corresponding to the additional restriction $x_k \leq \bar{x}_k$), we include (6.26) and (6.27) for the particular $(J_1, J_2) \in J_{+duals}$ in lieu of (6.26) and (6.25) in order to derive a tighter child-node relaxation.

Furthermore, along any branch within the branch-and-bound tree, constraints of the type (6.25)

or (6.27) (as explained above) that are generated at the upstream nodes are progressively inherited by the downstream nodes. Although this inheritance procedure guarantees tighter child-node relaxations, in order to preserve monotonicity in the lower bounds while keeping the size of the relaxations from growing excessively, we do the following. Consider any node (other than the root node) where, in addition to (6.24f), we have the constraints C_{+duals} plus other restrictions inherited from upstream nodes. Denote $C_{inherit}$ as the latter set of constraints. Now, suppose that we branch on some variable x_k and consider the child-node obtained for the branching restriction $x_k \geq \bar{x}_k$ (the case of $x_k \leq \bar{x}_k$ is similar). Then, to construct $C_{inherit}$ for this child-node's relaxation, denoted $C^{child}_{inherit}$, let C_{active} denote the constraints in the parent's node relaxation that have associated positive dual variables, let $C^{parent}_{inherit}$ denote the inherited constraints present in the parent node relaxation, and let C^{parent}_{k} denote those bound-factor restrictions within C_{+duals} for the parent node relaxation that involve the bound-factor $(x_k - l_k)$. Accordingly, we set

$$C^{child}_{inherit} = \{C^{parent}_{k} \cap C_{active}\} \cup \{C^{parent}_{inherit} \cap C_{active}\}, \tag{6.28}$$

where for any restriction of the type (6.25) in $C^{parent}_{k} \cap C_{active}$ that satisfies the conditions of Proposition 6.4, we utilize the corresponding tighter relaxations (6.27). Let LB_{parent} and LB_{child} be the lower bounds thus obtained for the parent and child-node, respectively. Then, we have the following result:

Proposition 6.5. $\text{LB}_{child} \geq \text{LB}_{parent}$.

Proof. Let PP2^{parent} and PP2^{child} respectively denote the parent and child-node LP relaxations. Consider Problem $\text{PP2}^{parent}_{active}$ that is constructed from the parent node relaxation PP2^{parent} by removing those bound-factor restrictions that have associated zero dual values at optimality from among the constraints in $C_{+duals} \cup C^{parent}_{inherit}$. Hence, $v(\text{PP2}^{parent}_{active}) = v(\text{PP2}^{parent}) = LB_{parent}$, where $v(\text{P})$ denotes the optimal objective function value for any given problem P. But PP2^{child} contains stronger restrictions of the type (6.24b) - (6.24g) than PP2^{parent} (or $\text{PP2}^{parent}_{active}$) based on the updated bound on the branching variable x_k (by [68]), plus all the remaining constraints (other than (6.24b) - (6.24g)) from $\text{PP2}^{parent}_{active}$ by (6.28) (possibly tightened via Proposition 6.4), in addition to other constraints in C_{+duals}. Hence, $LB_{child} = v(\text{PP2}^{child}) \geq v(\text{PP2}^{parent}_{active}) = LB_{parent}$. □

In our computational results, we found that this enhancement of PP2 significantly tightens the underlying relaxations, while suitably filtering the RLT constraints to control its size. Henceforth,

we shall therefore assume that PP2 has been thus enhanced.

We refer to the likewise generalized RLT-based branch-and-bound algorithms described in Section 6.2 as applied to Problems PP1 and (the enhanced) PP2, by RLT(PP1) and RLT(PP2), respectively. Similarly, the respective algorithms that incorporate SDP cuts are denoted by RLT_{SDP}(PP1) and RLT_{SDP}(PP2). Furthermore, note that the discussions in Remarks 6.2-6.3 hold true for polynomial programming problems as well. In particular, the RLT-based branch-and-bound algorithms RLT^{NB}(PP1) and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1) as applied to Problem PP1 involve partitioning on just nonbasic variables, and are implemented without and with SDP cuts, respectively. For the variants RLT(PP2), RLT_{SDP}(PP2), RLT^{NB}(PP1), and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1) that branch only on the nonbasic variables, we select a branching variable index as

$$j^* \in \arg\max_{j \in J_N} \theta_j,$$

where θ_j is computed as follows:

$$\theta_j \equiv \sum_{d=1}^{\delta-1} \sum_{J \in J_N^d} \left| \bar{X}_{J \cup j} - \bar{X}_J \bar{x}_j \right|, \forall j \in J_N.$$

The remainder of the branch-and-bound algorithm remains the same as that described in Section 6.1.

6.4 Static and Dynamic Basis Selection Techniques

In this section, we propose techniques for selecting a basis B for implementing the RLT reduction process for polynomial programming problems, which is then accordingly employed in generating Problem PP2 and optimizing it via RLT(PP2) and RLT_{SDP}(PP2), or utilized within the algorithmic procedures RLT^{NB}(PP1) and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1). We refer to this as a *static approach*. We also describe a *dynamic approach* in which the basis B is suitably revised during the algorithmic process in order to restructure Problem PP2 in a desirable fashion. Several alternative mechanisms are devised to obtain different variants of these static and dynamic approaches.

6.4.1 Static Approach

In this approach, we begin by solving the LP relaxation $\overline{\text{PP1}}$ of Problem PP1, which is derived by deleting (6.23h). Let $(\bar{x}, \bar{X})$ be the optimal solution obtained for Problem $\overline{\text{PP1}}$, and let $\lambda_J, \forall J \subseteq \mathcal{N}^\delta$, be the maximum of the resulting dual multipliers associated with the bound-factor constraints in (6.23f) that contain the variable X_J. The following alternative procedures determine an ordered set L of indices $j \in \mathcal{N}$, for each of which the corresponding basis B is then determined by utilizing the first m linearly independent columns of A that are associated with the variables having indices selected in the order in which they appear within L. The motivation for these methods is to explicitly enforce the product relationships (6.17) for the variables that are less likely to take on values at their range bounds, thereby tending to accelerate the convergence of the branch-and-bound process.

Method S.1: For the first method, we examine $\lambda_J, \forall J \subseteq \mathcal{N}^\delta$, in nonincreasing order and select the index set $J = J_1^*$ having the largest λ_J-value. Accordingly, we incorporate the distinct indices $j \in J_1^*$ in arbitrary (say, increasing) order within the list L, and next consider the ordered list of $\lambda_J, \forall J \subseteq \mathcal{N}^\delta - J_1^*$, to select the second index set J_2^* to insert within L. Continuing in this fashion, we construct the ordered list L of indices $j = 1, \ldots, n$, and then we reverse the order of indices in this list for subsequently selecting the corresponding basis B.

Method S.2: As an alternative to Method S.1, we construct the list L by arranging the indices $j \in \mathcal{N}$ in nondecreasing order of $\sum_{J \subseteq \mathcal{N}^{\delta-1}} (1 + k_{jJ}) \lambda_{(\{j\} \cup J)}$, where k_{jJ} is the number of times that j appears within J.

Method S.3: For the third method, we compute the number of bound-factor constraints, Q, to be generated in (6.24f) for the reduced RLT strategy. After sorting the constraints in (6.23f) in nonincreasing order of their dual values, we pick the first Q (possibly fewer) constraints associated with nonzero dual values to construct the set $\mathcal{N}_Q^\delta$ of the corresponding multi-sets $J \in (J_1 \cup J_2)$. For each variable index $j \in \mathcal{N}$, we compute $\sum_{J \subseteq \mathcal{N}_Q^\delta} k_{jJ}$ and sort the variables in nondecreasing order of this measure to form the list L.

Method S.4: This method constructs the desired list L based on an aggregate violation measure with respect to the RLT variable relationships. More specifically, for each $j \in \mathcal{N}$, we compute the total violation with respect to (6.17) via $\sum_{d=1}^{\delta-1} \sum_{J \subseteq \mathcal{N}^d} |\bar{x}_j \bar{X}_J - \bar{X}_{(\{j\} \cup J)}|$, and sort the variables $j = 1, \ldots, n$ in nondecreasing order of this measure within the list L.

6.4.2 Dynamic Approach

In this approach, for Method D.k, k $= 1,\ldots,4$, we respectively implement the method S.k, k $= 1,\ldots,4$, at the root node of the branch-and-bound tree to commence the branch-and-bound algorithm. Then, based on a specified *depth-parameter* Δ, for any active node selected at depth Δ in the branch-and-bound tree, we compute an updated vector of (dual) multipliers associated with (6.23f) as given by taking an element-wise maximum of the corresponding dual multipliers associated with (6.23f) that were computed for $\overline{\text{PP1}}$ at the root node, and the corresponding dual multipliers for the current node (where we take undefined elements as zero in the latter case). Accordingly, we invoke the particular Method S.k, k $\in \{1,\ldots,4\}$, at the current node using this resulting updated (dual) vector to recompute the basis B of A. The recomputed basis is then subsequently used to formulate (the enhanced) Problem PP2 for the present node and for all its descendents. Note that different bases can therefore be possibly specified for reformulating Problem PP2 at the different active nodes at depth Δ, and the particular reduced RLT algorithmic procedure is applied accordingly to each node's reformulated problem.

Remark 6.4. As a variant of Methods D.k for k $= 1,\ldots,4$, we could specify multiple depth parameters $\Delta_1 < \Delta_2 < \ldots < \Delta_{\mathfrak{D}}$ for some $\mathfrak{D} \geq 2$, and invoke a similar reformulation of Problem PP2 based on the determination of a revised basis for nodes at each of the levels $\Delta_1,\ldots,\Delta_{\mathfrak{D}}$ of the branch-and-bound tree. We advocate this investigation for future research. □

6.5 Computational Results

In this section, we evaluate the relative effectiveness of utilizing the proposed reduced size RLT formulations as compared to the original RLT formulation within a branch-and-bound algorithm. We begin by studying the performance of the different delineated static basis selection methods for implementing RLT(PP2), both with and without v-semidefinite cuts (Algorithms RLT(PP2) and RLT$_{\text{SDP}}$(PP2)). Next, utilizing the best performing (static) basis selection method as identified in the foregoing experiment, we compare the performance of RLT implemented on Problems PP1 and PP2, where for the former, we also investigate partitioning the bounding intervals only for the nonbasic variables (Algorithms RLT(PP1), RLT$^{\text{NB}}$(PP1), and RLT(PP2), respectively). In addition, we also examine the performance of these procedures with SDP cuts (RLT$_{\text{SDP}}$(PP1), RLT$^{\text{NB}}_{\text{SDP}}$(PP1),

and RLT_{SDP}(PP2), respectively). All algorithms were implemented on a workstation having a 2.33 GHz Intel Xeon processor with 3.25 GB of RAM and running Windows XP. Furthermore, we also present corresponding results using the commercial software BARON (Version 9.0.6) [66, 84], which was implemented on a workstation having two 2.13 Ghz Intel Xeon processors with 4GB of RAM and running Windows 7. Finally, the polynomial test problems were also optimized using the mixed-integer nonlinear programming software Couenne [12] on a workstation having 2.4 Ghz Intel Xeon processor with 8GB of RAM and running Linux. (The last two sets of runs were made on different workstations since BARON, Couenne, and CPLEX 11.1 were installed on different computers.) The RLT algorithms employ the dual optimizer with the steepest descent pricing rule implemented within CPLEX (Version 11.1) while using default settings for solving the underlying LP relaxations, and utilize SNOPT Version 7 [26] as the nonlinear programming (local search) solver for computing upper bounds by initializing this search process from the respective solutions obtained for the corresponding LP relaxations. Furthermore, we used Matlab® (Version R2008a [52]) for determining the rank of the candidate matrices during the basis selection process.

To facilitate an appropriate testing environment, we randomly generated polynomial programs for given values of the degree δ, number of variables (n), number of constraints (m and R), and densities of the objective function and constraints. The percentage of nonzero objective and constraint coefficients were specified as 100% and 25%, of which 50% were taken as positive, and the degree of the i^{th} constraint was set to $[(i-1) \mod \delta]+1, \forall i = 1,\ldots,R_1$. Each of the constraint functions was evaluated at l_j and $u_j, \forall j \in \mathcal{N}$, and the right-hand side was set equal to the average of these two values. The generated instances were checked for feasibility prior to computational analysis and the right-hand sides of the inequality constraints for infeasible instances were suitably decreased to attain feasibility. We did not include any nonlinear equality constraints; hence $R = R_1$. By varying the degree of the program, the number of variables and inequality constraints, and the size of the equality system as controllable problem parameters, we tried to assess the effectiveness of the proposed algorithms over a wide range of polynomial programming test problems. Utilizing 32 relatively moderately-sized problems in a primary analysis as described below, we first select the best performing basis selection method for formulating Problem PP2, and provide comparisons against applying RLT to Problem PP1, both with and without SDP cuts. Following this, we further evaluate the most competitive algorithmic variants using a second set of 10 more challenging larger-sized polynomial programming test problems. (All test instances are available via the Webpage http://filebox.vt.edu/users/dalkiran/website/.) For our computational experiments, we used $\varepsilon = 0.001$ as the optimality gap tolerance, and set a run-time limit of 500 and 3600 CPU seconds,

respectively, for the relatively moderate and large-sized problem instances.

Table 6.1: Performances of different static basis selection methods with RLT(PP2) and RLT_{SDP}(PP2).

	δ (# of	Average CPU Time				$n_{\text{premature}}$				Average % opt. gap*			
	instances)	S.1	S.2	S.3	S.4	S.1	S.2	S.3	S.4	S.1	S.2	S.3	S.4
RLT(PP2)	4 (12)	135	110	115	160	2	1	1	3	4.6	2.2	2.2	2.3
	5 (12)	199	150	143	197	4	3	3	4	24.9	6.8	6.7	10.9
	6 (8)	204	256	204	221	3	4	3	3	6.4	3.1	2.6	2.8
	Overall	**176**	**162**	**148**	**189**	**9**	**8**	**7**	**10**	**14.2**	**4.3**	**4.3**	**5.9**
RLT_{SDP}(PP2)	4 (12)	42	35.6	34.6	84.8	0	0	0	1	-	-	-	0.1
	5 (12)	35.1	26.6	25.1	25.7	0	0	0	0	-	-	-	-
	6 (8)	22	21.2	23	38.7	0	0	0	0	-	-	-	-
	Overall	**34.4**	**28.6**	**28.1**	**51.1**	**0**	**0**	**0**	**1**	-	-	-	**0.1**

* The average % optimality gap computed over the premature termination cases.

We begin with an explorative analysis to assess the relative performances of the four different static basis selection methods S.1,..., S.4 discussed in Section 6.4.1, as well as the effect of v-semidefinite cuts on computational effort. Table 6.1 provides a summary of the results obtained by reporting the average CPU times over the stated number of instances for each δ-value, along with the number of premature termination cases ($n_{\text{premature}}$), and the associated average percentage optimality gaps over such cases. As evident from Table 6.1, the performance differences between the tested basis selection methods underscores the importance of selecting an appropriate basis. The basis selection method S.3 outperformed the other methods when used within Algorithm RLT(PP2). Upon enhancing this procedure with the v-semidefinite cuts described in [76] (Algorithm RLT_{SDP}(PP2)), the performance improved significantly, also alleviating the premature termination cases. The average CPU time improvements for RLT_{SDP}(PP2) over RLT(PP2) using the respective basis selection methods S.1, S.2, S.3, and S.4 were 81%, 82%, 81%, and 73%. Moreover, the reductions in the overall average CPU times for RLT_{SDP}(PP2) using S.3 over the alternatives S.1, S.2, and S.4 were 18.3%, 1.9%, and 45%, respectively. Based on these results, we designated S.3 as the basis selection method for all subsequent runs.

It is worth mentioning here that some of the early termination cases were due to numerical issues encountered while solving the underlying LP relaxations. Decreasing the feasibility tolerance of CPLEX to 10^{-9} from its default value of 10^{-6} helped resolve some of these premature termination cases, but naturally increased the effort required for other instances. Hence, we retained

the default settings for the feasibility tolerance in our computational analysis. Another relevant note on implementation issues is related to the inheritance of SDP cuts. Although the SDP cuts generated for any node subproblem are valid for all descendent nodes, we let each current node relaxation inherit only those SDP cuts that were generated at upstream nodes up to four levels in our implementation. Note that this limited SDP cut inheritance scheme can possibly (though not likely) affect the monotonicity of the lower bounds as established by Proposition 6.5.

Table 6.2 presents the detailed results obtained for the best performing static basis selection method S.3 when utilized within Algorithms RLT(PP2) and RLT_{SDP}(PP2), and also includes comparative results for Algorithms RLT(PP1) and RLT^{NB}(PP1), along with their SDP cut-enhanced variants RLT_{SDP}(PP1) and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1). Five common instances out of the 32 test problems terminated prematurely using Algorithms RLT(PP2) and RLT(PP1) with an average optimality gap of 5.9% and 8.0%, respectively. Additionally, RLT(PP1) failed to solve the instance (δ, n, m, R) = (4, 10, 3, 4) and terminated with a 0.14% optimality gap, whereas RLT(PP2) optimized this instance within 397 CPU seconds. Two other instances with (δ, n, m, R) = (5, 8, 4, 6) and (6, 5, 3, 6) terminated prematurely using RLT(PP2) with a 0.3% average optimality gap, whereas RLT(PP1) optimized these instances within 34 CPU seconds, on average. However, when implemented with SDP cuts, all problems were optimized by both procedures within the set time limit, where the average CPU times were 40.4 seconds for RLT_{SDP}(PP1) versus 28.1 seconds for the proposed algorithm RLT_{SDP}(PP2), yielding a 30.4% savings in effort. We also implemented RLT^{NB}(PP1) and $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1), as delineated in Remark 6.3. Although RLT^{NB}(PP1) decreased the average computational effort as compared to RLT(PP1), it turned out that when these methods were enhanced with SDP cuts, $\text{RLT}^{\text{NB}}_{\text{SDP}}$(PP1) consumed more effort than both RLT_{SDP}(PP1) and RLT_{SDP}(PP2). Hence, RLT_{SDP}(PP1) and RLT_{SDP}(PP2) were selected for further comparison.

Using the basis selection method S.3, we also applied Remark 6.2 to eliminate the basic variables x_B via the substitution $x_B = B^{-1}(b - Nx_N)$, and accordingly reformulated the problems equivalently in the space of the $n-m$ nonbasic variables. Among the 32 test problems, we obtained three early termination cases using RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$, where the overall average CPU effort for this procedure was 52.7 seconds, as compared with 28.1 CPU seconds (or a savings of 46.7%) for RLT_{SDP}(PP2). However, for individual problems, the relative effectiveness of RLT_{SDP}(PP2) versus RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$ mainly depends on the size of the LP relaxations and the quality of the lower bounds at the root node. RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$ typically produces worse lower bounds at the root node than RLT_{SDP}(PP2), but the size of the relaxations for the former approach is smaller

Table 6.2: CPU times for RLT(PP2), RLT(PP1), and RLT$^{\text{NB}}$(PP1) with and without v-semidefinite cuts.

(δ, n, m, R)	RLT(PP2)	RLT(PP1)	RLT$^{\text{NB}}$(PP1)	RLT$_{\text{SDP}}$(PP2)	RLT$_{\text{SDP}}$(PP1)	RLT$^{\text{NB}}_{\text{SDP}}$(PP1)
(4, 8, 3, 4)	4.3	10.8	10.9	2.6	5	5.1
(4, 8, 3, 6)	20.1	70.2	53.6	18.4	41.6	32.3
(4, 8, 4, 4)	7.1	19.4	25.6	4.1	7.6	7.6
(4, 8, 4, 6)	500.3	500.5	225.1	41.5	84.9	93.8
(4, 9, 3, 4)	21.6	36.6	37.6	12.7	19	19.3
(4, 9, 3, 6)	5.3	6	6.5	6.2	5.1	5.4
(4, 9, 4, 4)	13.7	13.9	14.1	14.1	17.3	13
(4, 9, 4, 6)	30.1	40.9	35	25.5	34.9	32
(4, 10, 3, 4)	397.1	505.9	504.7	156.2	156.1	238.3
(4, 10, 3, 6)	95.3	92.8	93.4	50.3	37.9	38.6
(4, 10, 4, 4)	9.8	9.7	10.2	11.9	7.3	7.8
(4, 10, 4, 6)	272.3	148.5	158	71.1	73.8	74
Average	**114.7**	**121.2**	**97.9**	**34.6**	**40.9**	**47.3**
(5, 6, 3, 4)	17.4	48.5	52.7	4.6	7	6.6
(5, 6, 3, 6)	11.1	25.4	24.2	3	6.3	6.3
(5, 6, 4, 4)	1	1.2	1.2	1.1	1.2	1.1
(5, 6, 4, 6)	1.3	2.2	2.2	1.2	1.2	1.2
(5, 7, 3, 4)	500.8	500.1	152.5	25.3	45.8	45.1
(5, 7, 3, 6)	9.1	8	8.2	8.2	26.9	27.1
(5, 7, 4, 4)	500.6	501.2	500.7	25.5	46.9	50.5
(5, 7, 4, 6)	19.7	44.3	34.7	19.6	36.7	28.7
(5, 8, 3, 4)	85.3	106.9	124.3	76.8	126.6	127.6
(5, 8, 3, 6)	63.1	66.6	96.3	32.6	78.9	80.1
(5, 8, 4, 4)	6.1	5.3	5.3	5.5	5	5.1
(5, 8, 4, 6)	502.1	53.2	96.7	97.3	42.2	163.7
Average	**143.1**	**113.6**	**91.6**	**25.1**	**35.4**	**45.3**
(6, 5, 2, 4)	500.7	500.7	500.6	11.6	22.5	22.5
(6, 5, 2, 6)	3.8	8.2	7.5	4.3	6.9	6.8
(6, 5, 3, 4)	2.5	2.5	2.4	3	2.5	2.6
(6, 5, 3, 6)	500.4	14.9	14.5	3	5.3	4.6
(6, 6, 2, 4)	503.5	501.4	500.6	37.9	93.3	93.8
(6, 6, 2, 6)	92.8	206.2	227.8	101.5	210.7	258.7
(6, 6, 3, 4)	15.5	20.8	21	11.7	24.4	25
(6, 6, 3, 6)	15.9	30.3	34	11.2	12.7	12.4
Average	**204.4**	**160.6**	**163.5**	**23**	**47.3**	**53.3**
Overall Avg.	**147.8**	**128.2**	**111.9**	**28.1**	**40.4**	**48**

than those for the latter approach, which results in faster LP solution times for the former algorithm. We therefore designed a hybrid algorithmic approach, denoted **RLT$_{\text{SDP}}$(Hybrid)**, which is based on a parameter μ computed as the product of the ratios $\frac{\text{GAP}_1}{\text{GAP}_2}$ and $\frac{\text{N}_1}{\text{N}_2}$, where GAP_1 and

GAP_2 are the respective root node optimality gaps obtained using Algorithms RLT_{SDP}(PP2) and RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$ (prior to generating SDP cuts), and where N_1 and N_2 are the respective products of the number of variables and constraints in the LP relaxations for these two approaches. Hence, RLT_{SDP}(Hybrid) computes μ at the root node, and whenever $\mu < 1$, it proceeds by implementing RLT_{SDP}(PP2); otherwise, it implements RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$. This hybrid algorithm suitably compromises between the robustness of RLT_{SDP}(PP2) and the swiftness of RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$, and it optimized all the 32 test problems within 19.4 CPU seconds on average, versus 28.1 and 52.7 CPU seconds, respectively, for RLT_{SDP}(PP2) and RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$, resulting in respective savings in effort of 31% and 63.2%.

Table 6.3: Performances of different dynamic basis selection methods with RLT(PP2) and RLT_{SDP}(PP2).

	δ (# of	Average CPU Time				$n_{\text{premature}}$				Average % opt. gap*			
	instances)	D.1	D.2	D.3	D.4	D.1	D.2	D.3	D.4	D.1	D.2	D.3	D.4
RLT(PP2)	4 (12)	127	110	115	130	2	1	1	2	0.7	2.2	2.2	29.4
	5 (12)	166	151	144	156	3	3	3	3	32.5	6.3	6.7	10
	6 (8)	208	259	210	183	3	4	3	2	6.3	3.7	3.3	1.4
	Overall	**162**	**163**	**149**	**153**	**8**	**8**	**7**	**7**	**14.7**	**4.5**	**4.6**	**13**
RLT_{SDP}(PP2)	4 (12)	44.8	35.6	34.9	113.4	0	0	0	1	-	-	-	28.7
	5 (12)	74.6	26.7	25.4	65.9	1	0	0	1	1.5	-	-	2
	6 (8)	22.5	21.7	24	27.9	0	0	0	0	-	-	-	-
	Overall	**50.4**	**28.8**	**28.6**	**74.2**	**1**	**0**	**0**	**2**	**1.5**	**-**	**-**	**15.4**

* The average % optimality gap computed over the premature termination cases.

Next, we considered the dynamic basis selection methods. Table 6.3 presents the results obtained, displaying similar information as in Table 6.1 for the case of the static methods. When implemented within RLT(PP2), the dynamic methods D.1 and D.4 with $\Delta = 1$ outperformed the corresponding static methods S.1 and S.4, while decreasing the effort by 8.4% and 19.2%, respectively. However, when utilized within RLT_{SDP}(PP2), the methods D.1 and D.4 increased the average CPU time as compared with S.1 and S.4 by 46.6% and 45.3%, respectively. On the other hand, the static and dynamic variants of the other two basis selection methods performed similarly. This experiment reveals that dynamically updating the basis can have different effects, depending on the basis selection method used and other accompanying algorithmic strategies implemented such as SDP cuts. On a positive note, our best performing static basis selection method S.3 remained robustly superior under both variations of using SDP cuts and dynamically updating the basis. Hence, we continue with the static basis selection method S.3 for the remaining runs.

Table 6.4: Performances of RLT_{SDP}(PP2), RLT_{SDP}(PP1), and RLT_{SDP}(Hybrid) for solving the 10 larger-sized problems.

	Original problem in $\mathbb{R}^n$			Original problem in $\mathbb{R}^{n-m}$	
	RLT_{SDP}(PP2)	RLT_{SDP}(PP1)		RLT_{SDP}(PP1)	RLT_{SDP}(Hybrid)
(δ, n, m, R)	CPU time	CPU time	% opt. gap	CPU time	CPU time
(4, 11, 1, 6)	1557	2330		1329	1561
(4, 11, 2, 6)	354	587		948	355
(4, 12, 1, 6)	310	453		2495	316
(4, 12, 2, 6)	1912	3642	0.5	1801	1915
(5, 9, 1, 6)	320	726		752	329
(5, 9, 2, 6)	3622	3691	23	408	498
(6, 7, 1, 6)	1290	3208		511	1301
(6, 7, 2, 6)	314	1325		54	318
(7, 6, 1, 6)	3003	3635	9	1532	1601
(7, 6, 2, 6)	433	1436		22	359
Average	**1312**	**2103**	**11%**	**985**	**855**

Next, we compare the performances of RLT_{SDP}(PP2), RLT_{SDP}(PP1), and RLT_{SDP}(Hybrid) using the set of 10 larger-sized polynomial programming instances, where S.3 is utilized for selecting the basis within RLT_{SDP}(PP2) and RLT_{SDP}(Hybrid). Table 6.4 presents the results obtained, including those for the implementation of RLT_{SDP}(PP1) in the reduced nonbasic variable space $\mathbb{R}^{n-m}$. The proposed algorithm RLT_{SDP}(PP2) reduced the average computational times in comparison with RLT_{SDP}(PP1) by 37.6%. RLT_{SDP}(PP1) terminated prematurely for three test cases with an average optimality gap of 10.9%, whereas all three instances were optimized by RLT_{SDP}(PP2) within 2846 CPU seconds on average. As seen in Table 6.4, using the substitution process of Remark 6.2 to reformulate the original problem itself in $\mathbb{R}^{n-m}$, reduced the computational effort required by RLT_{SDP}(PP1), in comparison with its respective implementations in $\mathbb{R}^n$ by 53.2%. Note that the basis utilized for the substitution process greatly influences the performance of RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$. For instance, the respective average CPU times were 3315, 1838, 985, and 2936 seconds when the methods S.1, S.2, S.3, and S.4 were utilized for selecting the basis within RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$. Continuing with S.3, we observe that although this reduced implementation of RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$ also decreased the average computational effort in comparison with RLT_{SDP}(PP2) by 24.9%, the latter performed better on three of the 10 instances. Indeed, as seen from Table 6.4, the proposed algorithm RLT_{SDP}(Hybrid) that automatically composes these two procedures optimized all instances within 855 CPU seconds on average, reducing the compu-

tational effort required by RLT_{SDP}(PP1) and RLT_{SDP}(PP2) in $\mathbb{R}^n$, and by RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$, by 59.3%, 34.8%, and 13.2%, respectively. Hence, overall, we advocate RLT_{SDP}(Hybrid) for implementation.

Table 6.5: Performances of Couenne, BARON, and RLT_{SDP}(Hybrid) for solving the 10 larger-sized problems.

	Original problem in $\mathbb{R}^n$				Original problem in $\mathbb{R}^{n-m}$		RLT_{SDP}(Hybrid)
	Couenne		BARON		BARON		
(δ, n, m, R)	CPU time	% opt. gap	CPU time	% opt. gap	CPU time	% opt. gap	CPU time
(4, 11, 1, 6)	3613	420	3600	289	3600	290	1561
(4, 11, 2, 6)	3630	524	3600	279	3600	328	355
(4, 12, 1, 6)	3646	361	3600	192	3600	461	316
(4, 12, 2, 6)	3617	746	3600	504	3600	547	1915
(5, 9, 1, 6)	3692	276	3600	143	3600	145	329
(5, 9, 2, 6)	3653	602	3600	336	3600	108	498
(6, 7, 1, 6)	3688	294	3600	179	3600	76	1301
(6, 7, 2, 6)	3628	149	3600	119	3600	16	318
(7, 6, 1, 6)	3611	559	3600	296	1917		1601
(7, 6, 2, 6)	3647	150	1906		37		359
Average	**3643**	**408%**	**3431**	**260%**	**3075**	**246%**	**855**

Finally, we compare the performances of Couenne, BARON, and RLT_{SDP}(Hybrid). The results are presented in Table 6.5. In nine out of ten instances, Couenne and BARON terminated prematurely with respective average optimality gaps of 437% and 260%. The other instance was optimized in 1906 CPU seconds using BARON, whereas Couenne could only decrease the optimality gap to 150% within 3600 CPU seconds time limit. The implementation of BARON in the reduced nonbasic variable space decreased the required effort by 10% in comparison with its implementation in $\mathbb{R}^n$, but was yet unable to solve eight of the ten problem instances, having an unresolved average optimality gap of 246%. Overall, our proposed algorithm RLT_{SDP}(Hybrid) reduced the average computational times of Couenne, BARON in $\mathbb{R}^n$, and BARON in $\mathbb{R}^{n-m}$ by 77%, 75%, and 72%, respectively.

Chapter 7

Summary, Conclusions, and Future Research

Our primary objective in this dissertation was to contribute to the field of discrete and continuous nonconvex optimization and, in particular, to underscore the importance of good mathematical formulations as well as the strength and computational tractability of relaxations for solving such problems via branch-and-bound or branch-and-cut approaches. In alignment with this objective, we have explored and, theoretically as well as empirically, compared equivalent mathematical formulations, especially, in the context of polynomial programming problems and decision tree optimization. Furthermore, we have demonstrated the importance of tight and computationally tractable relaxations by developing v-semidefinite cuts and bound-grid-factor constraints that have resulted in substantial savings in computational effort for optimizing polynomial programs.

The first part of this dissertation studied risk management problems via a decision tree optimization approach. In Chapter 3, we addressed the strategic reduction of risk by allocating available resources to reduce failure probabilities of system safety components and subsequent losses related to final outcomes, as well as by selecting optimal strategic decisions or system design alternatives in the context of a hazardous event. Using a novel decision tree approach, the problem was modeled as a nonconvex mixed-integer 0-1 factorable program. We developed a specialized branch-and-bound algorithm in concert with polyhedral outer-approximations, valid inequalities, alternative linearization schemes, and range reduction strategies, and established its theoretical

convergence to global optimality. Among the various algorithmic variants tested, we advocate the strategy that implements branching variable selection rule C along with the arithmetic (or geometric) mean splitting rule, while utilizing the linearization method LM2 (6% CPU time reduction over LM1 on average, but 13% reduction on average for relatively higher decision node density problems); generating tangential supports via the BES($H = 4$) scheme; invoking the range reduction mechanism (all instances were solved to ε-optimality with range reduction compared to 100% early termination without range reduction), and incorporating the derived valid inequalities (13% CPU time reduction on average). In our computational experience, this proposed algorithmic approach outperformed the commercial software BARON (Version 8.1.5), yielding a more robust performance along with an 89.9% savings in effort on average.

The decision tree optimization research can be extended in several directions. For instance, instead of minimizing the overall risk, we could minimize the maximum risk, optimize the mean-variance based risk function for the system using Equations (3.5a) and (3.6), or minimize the conditional value at risk (CVaR). Additionally, as explained in Remark 3.2, we can use a diminishing marginal return loss function in lieu of the linear correspondent given by (3.4), as well as replace Bernoulli events with more general multistate events. Furthermore, as an algorithmic extension, we could explore a GUB-based partitioning strategy whenever we branch on a ϕ-variable. Finally, it is of interest to investigate specific applications of the proposed generic DTO problem framework in event-decision contexts that arise in various areas such as homeland security and health-care.

In the second part of the dissertation, we developed two classes of valid inequalities for enhancing Reformulation-Linearization Technique (RLT)-based linear programming (LP) relaxations for solving polynomial programming problems. In Chapter 4, we investigated the generation of valid inequalities via semidefinite programming constructs and derived new cutting plane strategies by imposing positive semidefiniteness on suitable dyadic variable-product matrices. Among alternative strategies, an effective RLT-based branch-and-cut algorithm incorporating v-semidefinite (or SDP) cuts (denoted RLT+SDP) was designed using composite cuts of two types ($v_{1112}\&v_{2011}$, or $v_{1142}\&v_{2011}$ for sparse problems), with cut inheritance from parent nodes, while generating $\bar{n}$ cuts per cycle with a restricted eigenvalue EV-10 (sorted) cut generation strategy for up to 5 cycles per node. The results demonstrated that the RLT+SDP scheme (using $v_{1112}\&v_{2011}$) reduced the computational effort by 44% and 77%, respectively over RLT as stand-alone approach and the commercial global optimization solver BARON (Version 8.1.5) for a test-bed of 60 randomly

generated polynomial programs. Moreover, the same significant advantage of RLT+SDP (using v_{1142}&v_{2011}) over RLT was observed when solving relatively more challenging (sparse) problems from the literature.

There are several variations and extensions that can be explored for v-semidefinite cuts. For instance, RLT-based linear programming relaxations enhanced by the proposed SDP cuts can be utilized to design efficient global optimization algorithms for factorable programming problems (Sherali and Wang [75]), mixed-integer linear and polynomial 0-1 programming problems (Sherali and Adams [68]), and quadratic assignment problems, among others. Also, note that a polynomial program of order δ can be equivalently transformed into one of order two (quadratic polynomial program) by defining new variables. This so-called *quadrification process* can be conducted in multiple ways as demonstrated in Shor [79] and Sherali and Tuncbilek [73]. However, having done this, although this would yield a weaker RLT relaxation than that obtained via the original polynomial program itself, even if all the possible quadrifications are *simultaneously* incorporated within the model (see Sherali and Tuncbilek [73] for a proof), the quadrified problem permits the derivation of SDP cuts using $v^{(1)}$ directly (which coincides with $v^{(2)}$ in this case). The comparative strengths of utilizing the RLT relaxations as enhanced by the SDP cuts can be also investigated, when applied to the quadrified versus the original polynomial programs.

Continuing in the same vein, in Chapter 5, we derived a new class of bound-grid-factor constraints to enhance RLT-based relaxations for solving polynomial programming problems. We introduced certain concurrent and sequential constraint generation routines that operate in concert with various equal distance, incumbent, optimal-to-parent, and optimal-to-self grid-factor assignment rules, as well as using reduced sized relaxations derived via the positive-discrepancy strategy. Among the alternative combinations, superlative results were obtained for the concurrent routine along with the optimal-to-parent rule and the positive-discrepancy strategy (designated RLT+GF*). In the computational results obtained using a test-bed of 15 randomly generated polynomial programming instances from Sherali et al. [76], RLT+GF* decreased the required CPU times by 21.1% over RLT, on average. On the other hand, the commercial software BARON (Version-9.0.6) terminated prematurely in 14 of the 15 instances within the 3600 CPU seconds time limit, yielding an average optimality gap of 45.9%, and consumed 1377 CPU seconds for the remaining instance, in comparison with 10 seconds required by RLT+GF*. We also examined several alternative implementations of constraint generation routines and branch-and-bound algorithms. When the dual simplex method with steepest-edge pricing strategy was utilized for solving the RLT+GF*-based

LP relaxations, the average CPU time increased by 188% in comparison with the barrier interior point optimizer. Moreover, the breadth-first node selection strategy yielded comparable results with RLT+GF* in comparison with the best-first strategy, but resulted in significantly higher optimality gaps for early termination cases for other algorithmic compositions. Finally, using exactly $\lfloor \delta/2 \rfloor$ grid-factors in each grid-factor based constraint when $\delta \geq 4$ as implemented in Sherali and Tuncbilek [74], as opposed to using up to $\lfloor \delta/2 \rfloor$ grid-factors as proposed herein, resulted in a 49.7% increase in effort on average over nine instances for the RLT+GF* procedure (excluding an early termination case).

Several extensions of the present study can be explored for future research. First, the RLT-based LP relaxations can be simultaneously tightened using bound-grid-factor constraints and v-semidefinite cuts (see Sherali et al. [76]) for solving polynomial programming problems. Second, the bound-grid-factor constraints can be implemented for quadrified problems (see [79, 73]) to compare their effect on solving quadrified versus original polynomial polynomial formulations. Finally, we could utilize bound-grid-factor constraints for solving more general nonlinear optimization problems, such as factorable programming problems [53, 75, 83].

In the last part of this dissertation, we addressed the derivation of reduced size RLT-based formulations for polynomial programming problems that have linear equality subsystems. In Chapter 6, we showed that a strict subset of the RLT defining identities, selected via a suitable basis partitioning scheme, implies the remaining defining identities and enables the removal of a major portion of the RLT constraints, while maintaining an equivalent representation. When both the original and reduced relaxation procedures were enhanced with v-semidefinite cuts, the static basis selection methods S.1, S.2, and S.3 implemented within RLT$_{\text{SDP}}$(PP2) outperformed RLT$_{\text{SDP}}$(PP1), decreasing the effort by 14.9%, 29.1%, and 30.4%, respectively, on a test-bed of 32 moderately sized instances, whereas S.4 increased the average CPU time for RLT$_{\text{SDP}}$(PP2) by 26.4%. Using the best performing method S.3, which tends to retain the most contributing bound-factor relationships from the original RLT formulation within the reduced representation, we further solved 10 relatively larger-sized polynomial programming problems. The results indicated a reduction in CPU times for the proposed algorithm RLT$_{\text{SDP}}$(PP2) over RLT$_{\text{SDP}}$(PP1) and BARON (Version 9.0.6) by 37.6% and 61.8%, respectively. Using the basis selection method S.3, we also designed a hybrid algorithm, denoted RLT$_{\text{SDP}}$(Hybrid), which performs a more extensive node zero analysis to automatically compose a solution strategy that either implements RLT$_{\text{SDP}}$(PP2) or the original RLT$_{\text{SDP}}$(PP1) procedure in a reduced nonbasic variable space in $\mathbb{R}^{n-m}$. This composite procedure

RLT_{SDP}(Hybrid) more robustly optimized all instances, reducing the average CPU time required by RLT_{SDP}(PP1) and RLT_{SDP}(PP2) in $\mathbb{R}^n$, and by RLT_{SDP}(PP1) in $\mathbb{R}^{n-m}$, by 52%, 31%, and 63.2%, respectively, for the 32 moderately sized problems, and by 59.3%, 34.8%, and 13.2%, respectively, for the 10 larger-sized problems. Furthermore, the respective average CPU times for the 10 larger-sized problems were 3643, 3431, and 3075 CPU seconds for Couenne and BARON in $\mathbb{R}^n$, and for BARON in $\mathbb{R}^{n-m}$, versus 855 CPU seconds for RLT_{SDP}(Hybrid) (yielding a savings in effort of 76.5%, 75.1%, and 72.2%, respectively). Hence, we recommend RLT_{SDP}(Hybrid) as a method of choice among the tested procedures for solving polynomial programming problems.

This research can be extended in several directions. For instance, additional constraint filtering techniques (see Sherali and Tuncbilek [74] and Sherali et al. [77], for example) can be employed to further reduce the size of the RLT relaxations. Also, similar to the use of SDP cuts, the underlying RLT-based linear programming relaxation can be tightened via grid-factor product restrictions (Sherali and Dalkiran [69]) as well as using suitable higher order RLT constraints. Similarly, we could tighten the variable bounds by implementing range reduction strategies as recommended in Remark 6.3 (see also Caprara and Locatelli [16], Ryoo and Sahinidis [65], Sherali and Tuncbilek [72], and Sherali and Tuncbilek [74] for a general discussion on range reduction strategies), while specifically exploiting the linear equality system. Finally, as suggested in Remark 6.4, certain multiple-depth variants of the dynamic basis selection strategy can be investigated, which utilize information from the root node and the current node to update the basis dynamically in a repetitive fashion.

We are currently developing an RLT-based open source software using the techniques developed in this research effort, including the features of reduced RLT representations, semidefinite cuts, bound-grid-factor constraints, range reduction strategies, and certain filtering techniques. Several extensions to this software can be implemented. For instance, we suggest integrating within the preprocessing step a quadrification process, which, despite weakening the underlying relaxations, yet has the potential to efficiently handle relatively large-sized RLT-based formulations for higher order polynomials. This software can be further modified to exploit problem structures, including sparsity patterns, and accordingly be designed to implement suitable specialized relaxations. Utilizing the theory introduced in Sherali and Wang [75], factorable programming problems can also be optimized in future versions of this software. As an aside, for benchmarking purposes, the polynomial programming test problems utilized in this dissertation have been made available for other researchers via the Web-page http://filebox.vt.edu/users/dalkiran/website/.

It is our belief and hope that the contributions of this dissertation will serve to develop effective algorithms for solving more general nonconvex optimization problems, including mixed-integer and factorable programs. Furthermore, the decision tree optimization approach may provide a useful management tool in the arenas of health-care and homeland security, such as in the design of preventive health monitoring programs and the allocation of resources to a series of contingent programs for mitigating the risk, thereby benefiting society in a broader context.

Bibliography

[1] C. Acosta and N. Siu. Dynamic event trees in accident sequence analysis: Application to steam generator tube rupture. *Reliability Engineering and System Safety*, 41:135–154, 1993.

[2] J. D. Andrews and L. M. Bartlett. Genetic algorithm optimization of a firewater deluge system. *Quality and Reliability Engineering International*, 19:39–52, 2003.

[3] J. D. Andrews and S. J. Dunnett. Event-tree analysis using binary decision diagrams. *IEEE Transactions on Reliability*, 49(2):230–238, 2000.

[4] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7:337 –363, 1995.

[5] K. M. Anstreicher. Semidefinite programming versus the Reformulation-Linearization Technique for nonconvex quadratically constrained quadratic programming. *Journal of Global Optimization*, 43(2–3):471–484, 2009.

[6] K. M. Anstreicher. On convex relaxations for quadratically constrained quadratic programming. Manuscript, Department of Management Sciences, University of Iowa, July 2010.

[7] K. M. Anstreicher and S. Burer. Computable representations for convex hulls of low-dimensional quadratic forms. *Mathematical Programming Series B*, 124:33–43, 2010.

[8] E. Artin. Uber die zerlegung definiter funktionen in quadrate. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 5(1):100–115, 1927.

[9] P. Artzner, F. Delbaen, J. M. Eber, and D. Heath. Coherent measures of risk. *Mathematical Finance*, 9(3):203–228, 1999.

[10] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty. *Nonlinear Programming: Theory and Algorithms*. John Wiley&Sons, New York, NY, third edition, 2006.

[11] G. K. Beim and B. F. Hobbs. Event tree analysis of lock closure risks. *Journal of Water Resources Planning and Management*, 123(3):169–178, 1997.

[12] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009.

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

[14] S. Burer and D. Vandenbussche. Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound. *Computational Optimization and Applications*, 43(2):181–195, 2009.

[15] S. Cafieri, P. Hansen, L. Liberti, L. Létocart, and F. Messine. Tight and compact convex relaxations for polynomial programming problems. Manuscript, LIX, École Polytechnique, F-91128 Palaiseau, France.

[16] A. Caprara and M. Locatelli. Global optimization problems and domain reduction strategies. *Mathematical Programming*, 125(1):123–137, 2010.

[17] R. B. Dillon, M. E. Paté-Cornell, and S. D. Guikema. Programmatic risk analysis for critical engineering systems under tight resource constraints. *Operations Research*, 51(3):354–370, 2003.

[18] J. B. Dugan, K. J. Sullivan, and D. Coppit. Developing a low-cost high-quality software tool for dynamic fault-tree analysis. *IEEE Transactions on Reliability*, 49(1):49–59, 2000.

[19] Y. Dutuit and A. Rauzy. A linear-time algorithm to find modules of fault trees. *IEEE Transactions on Reliability*, 45(3):422–425, 1996.

[20] M. L Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, 1981.

[21] C. A. Floudas and V. Visweswaran. *Quadratic Optimization. In: R. Horst, P. M. Pardalos. (eds.) Handbook of Global Optimization*, pages pp. 217–270. Kluwer Academic Publishers, Boston, MA., 1995.

[22] C. A. Floudas, P. M. Pardalos, J. S. Adjiman, W. R. Esposito, Z. H. Gumus, S. T. Harding, J. L. Klepeis, C. A. Meyer, and C. A. Schweiger. *Handbook of Test Problems in Local and Global Optimization*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1999.

[23] H. Föllmer and A. Schied. Convex measures of risk and trading constraints. *Finance and Stochastics*, 6:429–447, 2002.

[24] H. Furuta and N. Shiraishi. Fuzzy importance in fault tree analysis. *Fuzzy Sets and Systems*, 12:205–213, 1984.

[25] A. M. Geoffrion. Lagrangean relaxation for integer programming. *Mathematical Programming Studies*, 2:82–114, 1974.

[26] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005.

[27] K. R. Hayes. Identifying hazards in complex ecological systems - Part 1: Fault tree analysis for biological invasions. *Biological Invasions*, 4:235–249, 2002.

[28] S. He, Z. Li, and S. Zhang. Approximation algorithms for homogeneous polynomial optimization with quadratic constraints. *Mathematical Programming*, 125(2):353–383, 2010.

[29] C. Helmberg. Semidefinite programming. *European Journal of Operational Research*, 137 (3):461–482, 2002.

[30] D. Henrion and J.B. Lasserre. GloptiPoly: Global optimization over polynomials with Matlab and SeDuMi. *ACM Transactions on Mathematical Software*, 29(2):165–194, 2003.

[31] D. Henrion, J.B. Lasserre, and J. Löfberg. GloptiPoly 3: moments, optimization and semidefinite programming. *Optimization Methods and Software*, 24(4–5):761–779, 2009.

[32] W. Hock and K. Schittkowski. *Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin Heidelberg-New York, 1981.

[33] D. Huang, T. Chen, and M. J. Wang. Fuzzy set approach for event tree analysis. *Fuzzy Sets and Systems*, 118(1):153–165, 2001.

[34] Y. Jiang, J. D. McCalley, and T. Van Voorhis. Risk-based resource optimization for transmission system maintenance. *IEEE Transactions on Power Systems*, 21(3):1191–1200, 2006.

[35] S. Kaplan and B. J. Garrick. On the quantitative definition of risk. *Risk Analysis*, 1:11–27, 1981.

[36] R. Kenarangui. Event-tree analysis by fuzzy probability. *IEEE Transactions on Reliability*, 40(1):120–124, 1991.

[37] F. I. Khan and S. A. Abbasi. Analytical simulation and Profat II: A new methodology and a computer automated tool for fault tree analysis in chemical process industries. *Journal of Hazardous Materials*, 75(1):1–27, 2000.

[38] M. Kojima and L. Tunçel. On the finite convergence of successive SDP relaxation methods. *European Journal of Operational Research*, 143(2):325–341, 2002.

[39] M. Kojima, S. Kim, and H. Waki. Sparsity in sum of squares of polynomials. *Mathematical Programming*, 103(1):45–62, 2005.

[40] H. Konno, N. Kawadai, and H. Tuy. Cutting plane algorithms for nonlinear semi-definite programming problems with applications. *Journal of Global Optimization*, 25(2):141–155, 2003.

[41] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal of Optimization*, 11(3):796–817, 2001.

[42] J. B. Lasserre. An explicit equivalent positive semidefinite program for nonlinear 0-1 programs. *SIAM Journal on Optimization*, 12(3):756–769, 2002.

[43] J. B. Lasserre. Semidefinite programming vs. LP relaxations for polynomial programming. *Mathematics of Operations Research*, 27(2):347–360, 2002.

[44] J. B. Lasserre. Convergent SDP-relaxations in polynomial optimization with sparsity. *SIAM Journal on Optimization*, 17(3):822–843, 2006.

[45] M. Laurent and F. Rendl. *Semidefinite Programming and Integer Programming. In Aardal, K., Nemhauser, G. and Weismantel, R. (eds.): Handbook on Discrete Optimization*, pages pp. 393–514. Elsevier, Amsterdam, The Netherlands, 2005.

[46] L. Liberti. Linearity embedded in nonconvex programs. *Journal of Global Optimization*, 33: 157–196, 2005.

[47] L. Liberti. Effective RLT tightening in continuous bilinear programs. Internal Report 2003.18, Politecnico di Milano, 20133 Milano, Italy.

[48] L. Liberti and C. C. Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36:161–189, 2006.

[49] V. Manousiouthakis and D. Sourlas. A global optimization approach to rationally constrained rational programming. *Chemical Engineering Communications*, 115:127–147, 1992.

[50] H. Markowitz. Portfolio selection. *Journal of Finance*, 7:77–91, 1952.

[51] H. Markowitz. *Mean-variance Analysis in Portfolio Choice and Capital Markets*. Blackwell Publishing, Oxford, UK, 1987.

[52] MATLAB. *version 7.6.0 (R2008a)*. The MathWorks Inc., Natick, Massachusetts, 2008.

[53] G. P. McCormick. Computability of global solutions to factorable nonconvex programs: Part I - Convex underestimating problems. *Mathematical Programming*, 10:147–175, 1976.

[54] A. F. Mehr and I. Y. Tumer. Risk-based decision-making for managing resources during the design of complex space exploration systems. *Journal of Mechanical Design*, 128:1014–1022, 2006.

[55] J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281, 1995.

[56] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimizatio*. Wiley, Interscience, New York, NY, 1995.

[57] M. S. Nikulin. *Risk of a Statistical Procedure*. In: Encyclopaedia of Mathematics. Kluwer Academic Publishers, Bostan, MA., 1987.

[58] P. A. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, 96(2):293–320, 2003.

[59] P. A. Parrilo and B. Sturmfels. *Minimizing polynomial functions*, pages 83–99. In: Algorithmic and quantitative real algebraic geometry. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, AMS, Providence, RI, 2003.

[60] R. Paulavičius, J. Žilinskas, and A. Grothey. Investigation of selection strategies in branch and bound algorithm with simplicial partitions and combination of Lipschitz bounds. *Optimization Letters*, 4(2):173–183, 2010.

[61] S. Prajna, A. Papachristodoulou, P. Seiler, and P. A. Parrilo. *SOSTOOLS and its control applications*, pages 273–292. In: Positive polynomials in control. Springer-Verlag, Berlin Heidelberg, 2005.

[62] A. Rauzy. New algorithms for fault tree analysis. *Reliability Engineering and System Safety*, 40:203–211, 1993.

[63] S. M. Ross. *Introduction to Probability Models*. Academic Press, New York, NY, 2006.

[64] A. Ruszczyński and A. Shapiro. Optimization of convex risk functions. *Mathematics of Operations Research*, 31:433–452, 2006.

[65] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996.

[66] N. V. Sahinidis and M. Tawarmalani. *BARON 9.0.6: Global Optimization of Mixed-Integer Nonlinear Programs,* User's Manual, 2010.

[67] K. Schittkowski. *More Test Examples for Nonlinear Programming Codes*. Springer-Verlag New York Inc., New York, NY., 1987.

[68] H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publishers, Boston, MA., 1999.

[69] H. D. Sherali and E. Dalkiran. Combined bound-grid-factor constraints for enhancing RLT relaxations for polynomial programs. *Journal of Global Optimization*, Forthcoming, 2011.

[70] H. D. Sherali and B. M. P. Fraticelli. Enhancing RLT relaxations via a new class of semidefinite cuts. *Journal of Global Optimization*, 22:233–261, 2002.

[71] H. D. Sherali and C. H. Tuncbilek. A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique. *Journal of Global Optimization*, 2(1):101–112, 1992.

[72] H. D. Sherali and C. H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic programming problems. *Journal of Global Optimization*, 7:1–31, 1995.

[73] H. D. Sherali and C. H. Tuncbilek. Comparison of two Reformulation-Linearization Technique based linear programming relaxations for polynomial programming problems. *Journal of Global Optimization*, 10:381–390, 1997.

[74] H. D. Sherali and C. H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, 21(1):1–9, 1997.

[75] H. D. Sherali and H. Wang. Global optimization of nonconvex factorable programming problems. *Mathematical Programming*, 89:459–478, 2001.

[76] H. D. Sherali, E. Dalkiran, and J. Desai. Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts. Manuscript, Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA.

[77] H. D. Sherali, J. C. Smith, and W. P. Adams. Reduced first-level representations via the Reformulation-Linearization Technique: Results, counterexamples, and computations. *Discrete Applied Mathematics*, 101:247–267, 2000.

[78] H. D. Sherali, J. Desai, and T. S. Glickman. Optimal allocation of risk-reduction resources in event trees. *Management Science*, 54(7):1313–1321, 2008.

[79] N. Z. Shor. Dual quadratic estimates in polynomial and Boolean programming. *Annals of Operations Research*, 25:163–168, 1990.

[80] N. Z. Shor. *Nondifferentiable Optimization and Polynomial Problems*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998.

[81] R. M. Sinnamon and J. D. Andrews. Improved accuracy in quantitative fault tree analysis. *Quality and Reliability Engineering International*, 13:285–292, 1997.

[82] R. M. Sinnamon and J. D. Andrews. Improved efficiency in qualitative fault tree analysis. *Quality and Reliability Engineering International*, 13:293–298, 1997.

[83] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99:563–591, 2004.

[84] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103:225–249, 2005.

[85] N. H. Ulerich and G. J. Powers. On-line hazard aversion and fault diagnosis in chemical processes: The digraph + fault-tree method. *IEEE Transactions on Reliability*, 37(2):171–177, 1988.

[86] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[87] R. J. Vanderbei and H. Y. Benson. On formulating semidefinite programming problems as smooth convex nonlinear optimization problems. Technical Report, Department of Operations Research and Financial Engineering, Princeton University, Princeton, NJ, 1999.

[88] V. Visweswaran and C. A. Floudas. Unconstrained and constrained global optimization of polynomial functions in one variable. *Journal of Global Optimization*, 2:73–99, 1992.

[89] H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. *SIAM Journal on Optimization*, 17(1):218–242, 2006.

[90] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto. SparsePOP – A sparse semidefinite programming relaxations of polynomial optimization problems. *ACM Transactions on Mathematical Software*, 35(2):15:1–13, 2008.

[91] H. Wolkowicz, R. Saigal, and L. Vandenberghe. *Handbook of Semidefinite Programming*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2000.