USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

ESCOLA DE DOUTORAMENTO INTERNACIONAL DA USC

Brais González Rodríguez

Tese de doutoramento

Advances in Polynomial Optimization

Santiago de Compostela, 2022

Programa de Doutoramento en Estatística e Investigación Operativa

ESCOLA DE DOUTORAMENTO
INTERNACIONAL DA USC
(EDIUS)

TESE DE DOUTORAMENTO

Advances in Polynomial Optimization

Brais González Rodríguez

ESCOLA DE DOUTORAMENTO INTERNACIONAL DA USC

PROGRAMA DE DOUTORAMENTO EN ESTATÍSTICA E INVESTIGACIÓN OPERATIVA

SANTIAGO DE COMPOSTELA

2022

DECLARACIÓN DO AUTOR DA TESE

Advances in Polynomial Optimization

D. Brais González Rodríguez

Presento a miña tese, seguindo o procedemento adecuado ao Regulamento, e declaro que:

1) A tese abarca os resultados da elaboración do meu traballo.

2) No seu caso, na tese faise referencia ás colaboracións que tivo este traballo.

3) A tese é a versión definitiva presentada para a súa defensa e coincide coa versión enviada en formato electrónico.

4) Confirmo que a tese non incorre en ningún tipo de plaxio doutros autores nin de traballos presentados por min para a obtención doutros títulos.

En Santiago de Compostela, a 21 de setembro de 2022

Brais González Rodríguez

AUTORIZACIÓN DO DIRECTOR E DA DIRECTORA DA TESE

Advances in Polynomial Optimization

D. Julio González Díaz

Dª. Beatriz Pateiro López

INFORMAN:

Que a presente tese se corresponde co traballo realizado por D. Brais González Rodríguez, baixo a nosa dirección, e autorizamos a súa presentación, considerando que reúne os requisitos esixidos no Regulamento de Estudos de Doutoramento da USC, e que como directores desta non incorremos nas causas de abstención establecidas na lei 40/2015.

De acordo co indicado no Regulamento de Estudos de Doutoramento, declaran tamén que a presente tese de doutoramento é idónea para ser defendida en base á modalidade de monográfica con reprodución de publicacións, nos que a participación do doutorando foi decisiva para a súa elaboración e as publicacións se axustan ao Plan de Investigación.

En Santiago de Compostela, a 21 de setembro de 2022

Julio González Díaz

Beatriz Pateiro López

A mi abuelo Arturo

Preface

Esta tesis doctoral es el final de un largo camino que, en realidad, comenzó hace mucho más de cuatro años. Durante este tiempo, he tenido la suerte de estar acompañado por muchas personas, las cuales, de una forma u otra, son parte de esta memoria. Me gustaría, por tanto, agradecérselo a todas ellas de forma general, pues resulta imposible particularizar cada uno de los agradecimientos. No obstante, sí que me gustaría agradecer de forma particular a ciertas personas que han tenido un impacto especial a lo largo de estos años.

En primer lugar, y como no podría ser de otra forma, a mis directores de tesis: Julio y Bea. A Julio tengo que agradecerle mucho, pues no solamente ha sido mi director de tesis, sino que me ha acompañado a lo largo de mi vida académica durante los últimos ocho años. Comenzó siendo mi profesor en el segundo año de carrera y gracias a él descubrí mi pasión por la optimización. Además, Julio me dirigió las prácticas del Grado en Matemáticas, el TFG, el TFM y, finalmente, esta tesis doctoral. Gracias a Julio he crecido académicamente durante todos estos años y estoy seguro de que sin él esta tesis doctoral no existiría. Gracias Julio por ser un apoyo constante durante estos años, por estar siempre dispuesto a ayudarme y aconsejarme cuando lo necesito, por transmitirme en todo momento tu confianza en lo que hago y, sobre todo, por hacer que disfrute de mi trabajo día tras día. Gracias por ser mucho más que un director de tesis, no podría haber elegido mejor.

A mi codirectora Bea tengo que agradecerle que, desde el primer día en el que Julio y yo le propusimos que fuese codirectora, siempre me ha brindado su ayuda y su disponibilidad para tratar cualquier tema. Académicamente, sus conocimientos han sido muy importantes en el desarrollo de esta tesis. Gracias Bea por tu actitud amable y cercana desde el primer día y por haberme echado una mano siempre que lo he necesitado.

Gracias a todos mis coautores y coautoras, por formar parte de esta tesis y por todo lo que habéis aportado, no solo a este documento, sino a mi propio crecimiento académico. He disfrutado mucho trabajando con vosotros. A special thanks to Bissan Ghaddar, for always being willing to welcome me to Canada and help me whenever I needed it. It is a pleasure working with you.

Durante todos estos años, he tenido el placer de formar parte de la Facultad de Matemáticas, lugar en el que me he sentido como en casa desde el primer día. Me gustaría agradecer por tanto a las personas que forman parte de la Facultad y que han hecho esto posible. Del mismo modo, estos últimos cuatro años he formado parte del Departamento

de Estadística, Análisis Matemático y Optimización, cuyos miembros me han tratado como uno más desde el primer día. Inevitablemente, tengo que agradecer de forma especial a Balbina, pues he compartido docencia con ella durante cuatro años y siempre me ha transmitido su confianza, desde el primer día. Gracias también por confiar en mí para impartir por primera vez un curso de Python en la Facultad, que parece que ya se ha convertido en una tradición, y que ha sido de gran importancia en mi propia formación académica.

Durante algunos de estos años he tenido el placer de trabajar en el Instituto Tecnológico de Matemática Industrial. No tengo más que palabras de agradecimiento a la gente con la que he compartido este tiempo, pues desde el primer día me recibieron con los brazos abiertos. Gracias a ellos y ellas aprendí en meses lo que me habría llevado años aprender por mi cuenta.

Gracias a todas esas personas que luchan por nuestros derechos día tras día, por mejorar nuestras condiciones laborales, por conseguir lo que nos corresponde y por hacer que este camino sea mucho más fácil de transitar. Podéis estar orgullosos y orgullosas de lo que habéis conseguido en los últimos años.

Quiero agradecer de forma muy especial a los alumnos y alumnas que he tenido durante estos cuatro años en la asignatura de Programación Lineal y Entera. Me siento realmente afortunado de haberos dado clase; pues no solo he disfrutado enseñándoos, sino que también he aprendido mucho con vosotros. Puedo afirmar abiertamente que los jueves y los viernes han sido los mejores días de la semana, no porque la semana llegase a su fin, sino porque eran los días en los que os daba clase y compartía tiempo con vosotros. Muchas gracias a todos y todas, ya que si he disfrutado tanto dando clase ha sido fundamentalmente gracias a vosotros. De una forma u otra, formáis también parte de esta memoria.

No puedo finalizar sin agradecer a todos mis amigos y amigas que, de una forma u otra, me acompañan día tras día: a los que estaban presentes en mi vida antes de comenzar esta etapa y a los que he hecho durante estos años. Desde que empecé la carrera, he tenido el placer de conocer a gente maravillosa, que me ha apoyado siempre que lo he necesitado y con la he vivido experiencias inolvidables. También quiero agradecer a todas esas personas que he conocido en los diferentes congresos a los que he asistido; es un placer viajar a cualquier lugar y encontrarse siempre con gente que te recibe con los brazos abiertos. Un agradecimiento especial a todos los que han formado parte en el día a día de esta etapa predoctoral estos últimos cuatro años, pues son los que me han sacado una sonrisa en los momentos más complicados. Con ellos he vivido momentos únicos e inolvidables y han conseguido que estos años hayan sido sensacionales. Con esta memoria se cierra una etapa de mi vida y, sin lugar a dudas, me siento realmente afortunado de haberla compartido

con vosotros y vosotras. Gracias por estar siempre ahí y acompañarme en este camino.

Por último, pero no por ello menos importante, gracias a mi familia, por haberme apoyado durante toda mi vida. A mi padre, por apoyarme desde pequeño en todos mis proyectos y estar ahí siempre que lo he necesitado. Gracias por todo el cariño recibido durante estos años y por demostrarme que las adversidades nos hacen más fuertes. A mi madre, por transmitirme desde el día que nací los mejores valores posibles. Espero algún día poder acercarme a ser una persona tan maravillosa como ella. Gracias por apoyarme siempre, por cuidarme y por enseñarme a ser mejor día tras día. A mis abuelas, por enseñarme que lo realmente importante en la vida es ser una buena persona. Y, finamente, a mi abuelo Arturo, por ser un referente a seguir para mí. Gracias por enseñarme tantas y tan buenas cosas en la vida. Gracias por estar orgulloso de tu nieto, igual que yo lo estoy de ti. Gracias, simplemente, por formar parte de mi vida.

Al fin y al cabo, una tesis doctoral no consiste solamente en un proyecto de investigación, sino que va mucho más allá. Una tesis doctoral es una etapa más en la vida y me considero extremadamente afortunado de haberla compartido con todos vosotros y vosotras. Muchas gracias.

Brais González Rodríguez

Contents

Abstract

Polynomial optimization has a wide range of practical applications in fields such as optimal control, energy and water networks, facility location, management science, and finance. It also generalizes relevant optimization problems thoroughly studied in the literature, such as mixed-binary linear optimization, quadratic optimization, and complementarity problems. As finding globally optimal solutions is an extremely challenging task, the development of efficient techniques for solving polynomial optimization problems is of particular relevance. In this thesis we provide a detailed study of different techniques to solve this kind of problems and we introduce some nobel approaches in this field, including the use of statistical learning techniques. Furthermore, we also present a practical application of polynomial optimization to finance and more specifically, portfolio design.

KEYWORDS: Polynomial optimization; Reformulation-Linearization Technique; Algorithm; Branch and bound; Machine learning; Statistical learning; Conic optimization; Portfolio design.

Introduction

Polynomial optimization is a subclass of nonlinear optimization and it has a wide range of practical applications in fields such as optimal control, energy and water networks, process engineering, facility location, economics and equilibrium, management science, and finance. Polynomial optimization generalizes several special classes of problems that have been thoroughly studied in optimization, including mixed-binary linear optimization, convex/nonconvex quadratic optimization, and complementarity problems.

Polynomial optimization problems are $\mathcal{NP}$-hard in the general case so finding globally optimal solutions may be an extremely challenging task. Therefore, the development of efficient techniques for solving them is of particular relevance. There are many solvers for solving nonlinear optimization problems, but very few specific for polynomial programming problems. The main goal of this thesis is to implement a specific algorithm for solving polynomial programming problems and develop, implement, and evaluate different enhancements. In Chapter 1 we present some results that provide the theoretical foundations for the developments in the rest of the manuscript. First, we introduce a branch-and-bound algorithm for solving polynomial optimization problems, based on the Reformulation-Linearization-Technique (RLT). Then, we present some theoretical results to prove the convergence of the algorithm and of some extensions of it.

In Chapter 2 we present our own implementation of the RLT-based algorithm. This results in a new solver, `RAPOSa`, which, importantly, provides a benchmark to study the performance of different enhancements of the baseline RLT-based algorithm. Chapter 2 also conducts a detailed computational analysis of these enhancements. Some of them produce a great improvement in the performance of the RLT-based algorithm. One of the most notable enhancement is the use of different branching rules. In mixed-integer linear programming, the selection of the branching variable in a branch-and-bound algorithm has great relevance so we extend this to polynomial optimization. Thus, in Chapter 2, we present new branching rules and evaluate their impact on the performance of `RAPOSa`. Moreover, we also disscuss in detail two bound tightening tecqniques, obtaining remarkable results when we use them in `RAPOSa`. Furthermore, thanks to `RAPOSa`, we can run many computational experiments whose results are discussed in detail in the subsequent chapters.

However, it is not only important to develop techniques for solving these problems in general, but also to be able to adapt specific techniques to different classes of problems.

Depending on the characteristics of the problems, some approaches might be better than others. In recent years, there has been a substantial rise on the use of statistical learning techniques to improve the performance of branch-and-bound schemes. Most of the focus so far has been set on mixed-integer linear programming (MILP) problems and, in particular, on learning to choose the branching variable. Since branch-and-bound schemes are also at the core of most global optimization algorithms for solving polynomial optimization problems, it is natural to use the insights gained for MILP problems to improve algorithms for nonlinear optimization problems and, in particular, polynomial optimization ones. In Chapter 3 we discuss different branching rules for the core branch-and-bound algorithm in `RAPOSa`. Furthermore, we use statistical learning techniques to select the optimal branching rule based on the features of each problem, obtaining remarkable results.

Moreover, it is valuable to take the knowledge from linear and convex optimization to solve polynomial programming problems more efficiently. What the developments of the last several decades have shown was that semidefinite programming (SDP) and second-order cone programming (SOCP) are also a central tool in addressing nonconvexity. The detailed study of these techniques can improve the efficiency of different polynomial optimization solving techniques as we explain in detail in Chapter 4 and Chapter 5. First, in Chapter 4 we present new constraints based on SDP and SOCP. Adding these constraints to the linear relaxations produced by the RLT-based algorithm reduces the search space in the branch-and-bound tree, tightening the feasible region of the relaxations. However, it leads to higher solving times. This trade-off is carefully studied in Chapter 4. Then, in Chapter 5, we present an enhanced version of optimality-based bound tightening (OBBT) using SDP and SOCP constraints. The goal of this chapter is to combine what we have learnt in Chapter 2 regarding bound tightening techniques with what we have learnt in Chapter 4 regarding SDP and SOCP. Further, we show the robustness of the statistical learning techniques introduced in Chapter 3, using them in Chapter 4 and Chapter 5.

We conclude this thesis with Chapter 6, discussing a portfolio design problem that can be cast as a polynomial programming problem. We propose a model for a portfolio selection problem with transaction costs and two levels of decision-making. On the one hand, there is a broker-dealer that controls the fees to be charged on the different securities in order to maximize his benefit. On the other hand, there is an investor who chooses his portfolio trying to minimize risk (using Conditional Value at Risk as risk measure) while ensuring a minimum level of return. First, we model this situation as a bilevel optimization problem and then, using duality, we transform this bilevel problem into a polynomial optimization problem. We conclude Chapter 6 with a detailed computational analysis of some situations arising with different risk profiles and different hierarchy for the bilevel problem.

Chapter 1

Fundamentals of the Reformulation-Linearization Technique

1.1 Introduction

Polynomial optimization problems have been extensively studied in the literature over the last few decades. They are of great relevance both from a theoretical and a practical point of view when solving problems from real-world situations that can be modelled as polynomial programming problems. From the theoretical point of view, Hägglöf et al. (1995) give a survey of different techniques for finding global optima using Gröbner bases. Moreover, Kamyar and Peet (2015) discuss different algorithms for solving polynomial optimizations problems, such as Quantifier Elimination, Reformulation-Linearization Technique, Blossoming and Gröbner basis methods. Another field of research in recent years is the use of second-order cone programming (SOCP) and semidefinite programming (SDP) in the context of polynomial optimization. For instance, Ahmadi and Majumdar (2016) solve three classes of problems arisen in real-world situations, modelling them as polynomial optimization problems, and using SDP and SOCP techniques. Lasserre (2006) proposes to use SDP relaxations to solve polynomial optimization problems, exploiting their sparsity. However, not only exact methods have been studied in the literature. He et al. (2010) propose polynomial-time approximation algorithms to solve polynomial optimization problems. Moreover, some authors have studied several real-world applications. Ghaddar et al. (2016) present a polynomial optimization formulation of an alternating current optimal power flow (ACOPF) problem. Ghaddar et al. (2017) show how water network design problems can be modelled as polynomial optimization problems.

Sherali and Tuncbilek (1992) present a new algorithm based on the Reformulation-Linearization Technique (hereafter RLT) for finding global optima of polynomial optimization problems, based on solving successive linear relaxations in a branch-and-bound scheme. Sherali et al. (2012a), Sherali et al. (2012b), and Dalkiran and Sherali (2013) present some enhancements of the RLT-based algorithm. This chapter has two main objectives.

First, to introduce the RLT-based algorithm to the reader. Second, to present and prove new theoretical results to ensure the convergence of some extensions of the RLT-based algorithm.

The outline of the chapter is as follows. First, in Section 1.2, we present some basic definitions. Then, in Section 1.3, we present a summary of how the RLT-based algorithm works in conjunction with theoretical results to prove the convergence of the algorithm. Finally, in Section 1.4 we discuss some new theoretical results, in order to prove the convergence in the presence of some modifications of the RLT-based algorithm.

1.2 Preliminaries

Before explaining how the RLT-based algorithm works, we have to introduce some notation and definitions.

Definition 1.1. Given a vector of variables $\boldsymbol{x} = (x_1, \dots, x_n)^\intercal$, a *monomial* is any expression of the form $m(\boldsymbol{x}) = \prod_{j=1}^n x_j^{\beta_j}$, where $\beta_j \in \mathbb{N} \cup \{0\}$ for each $j \in \{1, \dots, n\}$.

Definition 1.2. Given a monomial $m(\boldsymbol{x}) = \prod_{j=1}^n x_j^{\beta_j}$, its *degree* is defined as $\sum_{j=1}^n \beta_j$.

Definition 1.3. Given a vector of variables $\boldsymbol{x} = (x_1, \dots, x_n)^\intercal$, a *polynomial* is the sum of a finite number of monomials multiplied by coefficients, that is, any expression of the form $\phi(\boldsymbol{x}) = \sum_{t \in T} \alpha_t \prod_{j=1}^n x_j^{\beta_{tj}}$, where $T \in \mathbb{N}$, $\alpha_t \in \mathbb{R}$ for each $t \in T$, and $\beta_{tj} \in \mathbb{N} \cup \{0\}$ for each $j \in \{1, \dots, n\}$ and $t \in T$.

Definition 1.4. Given a polynomial $\phi(\boldsymbol{x}) = \sum_{t \in T} \alpha_t \prod_{j=1}^n x_j^{\beta_{tj}}$, its *degree* is defined as the greatest of the degrees of all the monomials in the polynomial, that is, the degree is $\max_{t \in T} \{\sum_{j=1}^n \beta_{tj}\}$.

Definition 1.5. Given a finite set N and a map $p : N \to \mathbb{N} \cup \{0\}$, a *multiset* is a pair (N, p), where p denotes the multiplicity of each element. The *cardinal* of a multiset is $|(N, p)| := \sum_{j \in N} p(j)$.

Throughout this thesis, the use of multisets is quite limited and we make the following abuse of notation: given a set $N = \{1, \dots, n\}$, we denote the multiset $\{1, \overset{p(1)}{\dots}, 1, \dots, n, \overset{p(n)}{\dots}, n\}$ by (N, p). Furthermore, if the multiplicity of all elements of the multiset is the same, that is, if $p(j) = \delta$ for each $j \in N$, we denote the multiset $\{1, \overset{\delta}{\dots}, 1, \dots, n, \overset{\delta}{\dots}, n\}$ by N^δ.

Using multisets, one can express any monomial of the form $m(\boldsymbol{x}) = \prod_{j=1}^n x_j^{\beta_j}$ with degree δ as $m(\boldsymbol{x}) = \prod_{j \in J} x_j$, for some $J \subset N^\delta$. Throughout this thesis, by monomial $J \subset N^\delta$ we mean the monomial $m(\boldsymbol{x}) = \prod_{j \in J} x_j$. Similarly, one can express any polynomial

of the form $\phi(\boldsymbol{x}) = \sum_{t\in T} \alpha_t \prod_{j=1}^{n} x_j^{\beta_{tj}}$ with degree δ as $\phi(\boldsymbol{x}) = \sum_{t\in T} \alpha_t \prod_{j\in J_t} x_j$, where $J_t \subset N^\delta$ for each $t \in T$.

From now on, N denotes the set $\{1, \ldots, n\}$.

Definition 1.6. A *polynomial optimization problem* is an optimization problem of the following form:

$$\begin{array}{lll} \text{minimize} & \phi_0(\boldsymbol{x}) & \\ \text{subject to} & \phi_r(\boldsymbol{x}) \geq \beta_r, & r = 1, \ldots, R_1 \\ & \phi_r(\boldsymbol{x}) = \beta_r, & r = R_1 + 1, \ldots, R \\ & \boldsymbol{x} \in \Omega \subset \mathbb{R}^n, & \end{array} \qquad (PP(\Omega))$$

where each $\phi_r(\boldsymbol{x}) = \sum_{t\in T_r} \alpha_{rt} \prod_{j\in J_{rt}} x_j$ is a polynomial of degree $\delta_r \in \mathbb{N}$, $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n : 0 \leq l_j \leq x_j \leq u_j < \infty, \forall j \in N\} \subset \mathbb{R}^n$ is a hyperrectangle containing the feasible region, and the degree of the problem is defined as $\delta = \max_{r\in\{0,\ldots,R\}} \delta_r$. Hereafter, we suppose $\delta \geq 2$.

1.3 The RLT-Based Algorithm

In this section we explain the fundamentals of the RLT-based algorithm, following the contents of Sherali and Tuncbilek (1992). In order to do this, we have to introduce some concepts first.

Definition 1.7. Given a polynomial optimization problem of the form ($PP(\Omega)$), we define the *bounding factors* as the constraints of the form $x_j - l_j \geq 0$ and $u_j - x_j \geq 0$ for each $j \in N$.

Definition 1.8. Given a polynomial optimization problem of the form ($PP(\Omega)$) and a monomial $J \subset N^\delta$ with $2 \leq |J| \leq \delta$, we define the corresponding *RLT identity* as $X_J = \prod_{j\in J} x_j$, where X_J is called an *RLT variable*. Throughout this thesis, we sometimes make an abuse of notation and denote variable x_j by $X_{\{j\}}$.

Definition 1.9. Given a point $\boldsymbol{x}$ and the polynomial programming problem ($PP(\Omega)$), we define the extension of $\boldsymbol{x}$ as $E(\boldsymbol{x}) = (X_J)_{J\subset N^\delta}$, where $X_J = \prod_{j\in J} x_j$ for each $J \subset N^\delta$.

Definition 1.10. Given a polynomial $\phi(\boldsymbol{x})$, we define its *linearization* $[\phi(\boldsymbol{x})]_L$ as the result of replacing every monomial of degree greater than one in $\phi(\boldsymbol{x})$ by the corresponding RLT variable.

Sherali and Tuncbilek (1992) define the linear relaxation of problem ($PP(\Omega)$) as

$$
\begin{array}{rll}
\text{minimize} & [\phi_0(\boldsymbol{x})]_L & \\
\text{subject to} & [\phi_r(\boldsymbol{x})]_L \geq \beta_r, & r = 1, \ldots, R_1 \\
& [\phi_r(\boldsymbol{x})]_L = \beta_r, & r = R_1 + 1, \ldots, R \\
& [\prod_{j \in J_1}(x_j - l_j) \prod_{j \in J_2}(u_j - x_j)]_L \geq 0, & \forall J_1 \cup J_2 \subset N^\delta, \\
& & |J_1 \cup J_2| = \delta \\
& \boldsymbol{x} \in \Omega \subset \mathbb{R}^n. &
\end{array}
\tag{$LP(\Omega)$}
$$

The constraints of the form $F_\delta(J_1, J_2) = \prod_{j \in J_1}(x_j - l_j) \prod_{j \in J_2}(u_j - x_j) \geq 0$ are called *bound-factor constraints* and each of them consists of the product of δ bounding factors. Sherali and Tuncbilek (1992) prove that the number of bound-factor constraints corresponds with the combinatorial number $2n + \delta - 1$ over δ.

With these ingredients, Sherali and Tuncbilek (1992) prove three lemmas needed to ensure the convergence of the RLT-based algorithm introduced in that paper. We include the proof of the lemmas in this thesis, based on the work just mentioned, but making explicit some arguments that are omitted there. Later in this chapter, we will need to check that these proofs are valid under other assumptions.

Lemma 1.1. *Let $v[PP(\Omega)]$ be the optimal value of ($PP(\Omega)$) and let $v[LP(\Omega)]$ be the optimal value of $LP(\Omega)$. Then, $v[LP(\Omega)] \leq v[PP(\Omega)]$. Moreover, for any optimal solution of ($LP(\Omega)$) of the form $(\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$, $\bar{\boldsymbol{x}}$ is an optimal solution of $PP(\Omega)$.*

Proof. Let $\phi_0[PP(\Omega)](\boldsymbol{x})$ be the objective function of ($PP(\Omega)$) and $\phi_0[LP(\Omega)](\boldsymbol{x})$ the objective function of ($LP(\Omega)$). Let $\bar{\boldsymbol{x}}$ be a feasible solution of ($PP(\Omega)$). Then $(\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$ is a feasible solution of ($LP(\Omega)$) and $\phi_0[PP(\Omega)](\bar{\boldsymbol{x}}) = \phi_0[LP(\Omega)](\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$. Thus, $v[LP(\Omega)] \leq v[PP(\Omega)]$.

Moreover, given an optimal solution $(\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$ of ($LP(\Omega)$), it holds that $\bar{\boldsymbol{x}}$ is feasible to ($PP(\Omega)$) and $\phi_0[PP(\Omega)](\bar{\boldsymbol{x}}) = \phi_0[LP(\Omega)](\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$. Therefore, $\bar{\boldsymbol{x}}$ is an optimal solution of ($PP(\Omega)$). □

Lemma 1.2. *The bound-factor constraints of the form $[F_{\delta'}(J_1, J_2)]_L \geq 0$ where $J_1 \cup J_2 \subset N^\delta$, $|J_1 \cup J_2| = \delta'$ and $1 \leq \delta' < \delta$ are implied by the bound-factor constraints of the form $[F_\delta(J_1, J_2)]_L \geq 0$ where $J_1 \cup J_2 \subset N^\delta$, $|J_1 \cup J_2| = \delta$.*

Proof. For each δ' with $1 \leq \delta' < \delta$, $j \in N$ and J_1, J_2 where $J_1 \cup J_2 \subset N^\delta$, $|J_1 \cup J_2| = \delta'$, consider the constraints $[F_{\delta'+1}(J_1 \cup \{j\}, J_2)]_L \geq 0$ and $[F_{\delta'+1}(J_1, J_2 \cup \{j\})]_L \geq 0$. It holds

that

$$[F_{\delta'+1}(J_1 \cup \{j\}, J_2)]_L + [F_{\delta'+1}(J_1, J_2 \cup \{j\})]_L =$$
$$[(x_j - l_j)F_{\delta'}(J_1, J_2)]_L + [(u_j - x_j)F_{\delta'}(J_1, J_2)]_L =$$
$$[x_j F_{\delta'}(J_1, J_2)]_L - l_j[F_{\delta'}(J_1, J_2)]_L + u_j[F_{\delta'}(J_1, J_2)]_L - [x_j F_{\delta'}(J_1, J_2)]_L =$$
$$(u_j - l_j)[F_{\delta'}(J_1, J_2)]_L.$$

Since $u_j - l_j \geq 0$, it follows that $[F_{\delta'}(J_1, J_2)]_L \geq 0$ is implied by $[F_{\delta'+1}(J_1 \cup \{j\}, J_2)]_L \geq 0$ and $[F_{\delta'+1}(J_1, J_2 \cup \{j\})]_L \geq 0$. Therefore, using the principle of induction, the proof is complete. □

Corollary 1.3. *In (LP(Ω)), all variables X_J, where $J \subset N^\delta$, have finite lower and upper bounds due to bound-factor constraints and as a consequence of Lemma 1.2. Therefore, all variables of (LP(Ω)) are bounded and (LP(Ω)), if feasible, has a finite optimum.*

Proof. Let X_J be an arbitrary variable where $J \subset N^\delta$. We prove that X_J has finite lower and upper bounds by induction on $|J|$.

Suppose that $|J| = 2$, where $J = \{j_1, j_2\}$. Using Lemma 1.2, the bound-factor constraint $[F_2(J_1, J_2)]_L \geq 0$, where $J_1 = J$ and $J_2 = \emptyset$, is implied by those in $LP(\Omega)$. Thus, it holds that

$$[F_2(J_1, J_2)]_L = [(x_{j_1} - l_{j_1})(x_{j_2} - l_{j_2})]_L = X_J - l_{j_2}x_{j_1} - l_{j_1}x_{j_2} + l_{j_1}l_{j_2} \geq 0$$

and, hence, $X_J \geq l_{j_2}x_{j_1} + l_{j_1}x_{j_2} - l_{j_1}l_{j_2}$ so X_J has a finite lower bound.

Similarly, using Lemma 1.2, the bound-factor constraint $[F_2(J_1, J_2)]_L \geq 0$, where $J_1 = \{j_1\}$ and $J_2 = \{j_2\}$, is implied by those in $LP(\Omega)$. Thus, it holds that

$$[F_2(J_1, J_2)]_L = [(x_{j_1} - l_{j_1})(u_{j_2} - x_{j_2})]_L = -X_J + u_{j_2}x_{j_1} + l_{j_1}x_{j_2} - l_{j_1}u_{j_2} \geq 0$$

and, hence, $X_J \leq u_{j_2}x_{j_1} + l_{j_1}x_{j_2} - l_{j_1}u_{j_2}$ so X_J has a finite upper bound.

Suppose now the result is valid for $|J| < \delta'$ and let's prove it for $|J| = \delta'$, where $\delta' \leq \delta$ and $J = \{j_1, \ldots, j_{\delta'}\}$.

Using Lemma 1.2, the bound-factor constraint $[F_2(J_1, J_2)]_L \geq 0$, where $J_1 = J$ and $J_2 = \emptyset$, is implied by those in $LP(\Omega)$. Thus, it holds that

$$[F_{\delta'}(J_1, J_2)]_L = \Big[\prod_{j \in J}(x_j - l_j)\Big]_L = X_J + f(\boldsymbol{x}, \boldsymbol{X}) \geq 0$$

where $f(\boldsymbol{x}, \boldsymbol{X})$ is a linear function where all RLT variables are associated to monomials with degree smaller than δ so they are bounded. Therefore, $X_J \geq -f(\boldsymbol{x}, \boldsymbol{X})$ and hence,

X_J has a finite lower bound.

Let $J_1 = J \setminus \{j_{\delta'}\}$ and $J_2 = \{j_{\delta'}\}$. Using Lemma 1.2, the bound-factor constraint $[F_2(J_1, J_2)]_L \geq 0$ is implied by those in $LP(\Omega)$. It holds that

$$[F_{\delta'}(J_1, J_2)]_L = \Big[(u_{j_{\delta'}} - x_{j_{\delta'}}) \prod_{j \in J \setminus \{j_{\delta'}\}} (x_j - l_j)\Big]_L = -X_J + f(\boldsymbol{x}, \boldsymbol{X}) \geq 0$$

where $f(\boldsymbol{x}, \boldsymbol{X})$ is a linear function where all RLT variables are associated to monomials with degree smaller than δ so they are bounded. Therefore, $X_J \leq f(\boldsymbol{x}, \boldsymbol{X})$ and hence, X_J has a finite upper bound. □

Lemma 1.4. *Let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ be a feasible solution of $LP(\Omega)$. If $\bar{x}_p = l_p$ for a certain $p \in N$, then $\bar{X}_{J\cup\{p\}} = l_p \bar{X}_J$ for each $J \subset N^\delta$ with $1 \leq |J| \leq \delta - 1$. Similarly, if $\bar{x}_p = u_p$ for a certain $p \in N$, then $\bar{X}_{J\cup\{p\}} = u_p \bar{X}_J$ for each $J \subset N^\delta$ with $1 \leq |J| \leq \delta - 1$.*

Proof. We prove the result using the principle of induction over $|J|$. First, consider $J = \{q\}$ with $q \in N$ so $|J| = 1$. Using Lemma 1.2, bound-factor constraints from $(LP(\Omega))$ imply that

$$[(x_p - l_p)(x_q - l_q)]_L = X_{\{q,p\}} - l_q x_p - l_p x_q + l_p l_q \geq 0,$$

$$[(x_p - l_p)(u_q - x_q)]_L = -X_{\{q,p\}} + u_q x_p + l_p x_q - l_p u_q \geq 0.$$

Therefore, it holds that $l_q(x_p - l_p) + l_p x_q \leq X_{\{q,p\}} \leq l_p x_q + u_q(x_p - l_p)$.

Evaluating the above in $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ and taking into account that $\bar{x}_p = l_p$, it holds that $\bar{X}_{\{q,p\}} = l_p x_q$.

Assume now that $\bar{X}_{J\cup\{p\}} = l_p \bar{X}_J$ holds for $|J| \in \{1, \ldots, t-1\}$ and consider $J \subset N^\delta$ with $|J| = t$ where $2 \leq t \leq \delta - 1$. For each $q \in J$, using Lemma 1.2, bound-factor constraints from $(LP(\Omega))$ imply that

$$\Big[(x_p - l_p)(x_q - l_q) \prod_{j \in J \setminus \{q\}} (x_j - l_j)\Big]_L \geq 0,$$

$$\Big[(x_p - l_p)(u_q - x_q) \prod_{j \in J \setminus \{q\}} (x_j - l_j)\Big]_L \geq 0.$$

Let us write $\prod_{j\in J\setminus\{q\}}(x_j - l_j) = \prod_{j\in J\setminus\{q\}} x_j + \phi(\boldsymbol{x})$ where $\phi(\boldsymbol{x})$ is a polynomial of degree no more than $t - 2$. Therefore,

$$X_{J\cup\{p\}} - l_p X_J \geq l_q(X_{J\setminus\{q\}\cup\{p\}} - l_p X_{J\setminus\{q\}}) + [l_p x_q \phi(\boldsymbol{x}) - x_p x_q \phi(\boldsymbol{x})]_L + l_q[x_p \phi(\boldsymbol{x}) - l_p \phi(\boldsymbol{x})]_L,$$

$$X_{J\cup\{p\}} - l_p X_J \leq u_q(X_{J\setminus\{q\}\cup\{p\}} - l_p X_{J\setminus\{q\}}) + [l_p x_q \phi(\boldsymbol{x}) - x_p x_q \phi(\boldsymbol{x})]_L + u_q[x_p \phi(\boldsymbol{x}) - l_p \phi(\boldsymbol{x})]_L.$$

By the induction hypothesis $\bar{X}_{J\setminus\{q\}\cup\{p\}} = l_p \bar{X}_{J\setminus\{q\}}$. Thus, evaluating the above in $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ and taking into account that $\bar{x}_p = l_p$, the right-hand sides of both inequalities become zero. Therefore, $\bar{X}_{J\setminus\{p\}} = l_p \bar{X}_J$.

The case for $\bar{x}_p = u_p$ can be proved similarly. □

Equipped with the previous lemmas, Sherali and Tuncbilek (1992) define the RLT-based algorithm as a branch-and-bound algorithm consisting of solving linear relaxations of the polynomial programming problem. The linear relaxation solved in each node of the branch-and-bound tree is built as in ($LP(\Omega)$). After solving each node, we select the branching variable x_p maximizing a certain function of the violations of the RLT identities. Specifically, we select the branching variable x_p using the following formula:

$$\begin{gathered} p \in \arg\max_{j\in N}\{\theta_j\}, \text{ where} \\ \theta_j = \max_{J\subset N^\delta, 1\le|J|\le\delta-1}\{|\bar{X}_{J\cup\{j\}} - \bar{x}_j \bar{X}_J|\} \text{ for each } j \in N, \end{gathered} \tag{1.1}$$

where $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ if the optimal solution of the corresponding node.

Then, we do a spatial branching by adding the bound $x_p \le \bar{x}_p$ to one child node and $x_p \ge \bar{x}_p$ to the other one, where $\bar{x}_p$ is the optimal value for x_p in the current node. Note that those bound-factor constraints in which x_p appears are also updated in the linear relaxations of the child nodes.

In the following, we explain how the RLT-based algorithm works, in order to solve the polynomial optimization problem ($PP(\Omega)$).

Initialization.

- Initialize the current best solution $\boldsymbol{x}^* = \emptyset$ and the best objective value (upper bound) $v^* = \infty$. Let $k = 1$, $t = 1$, $tot = 1$, $Q_1 = \{1\}$ and $\Omega^{1,1} = \Omega$.
- Solve $LP(\Omega^{1,1})$, leading to an optimal solution $(\boldsymbol{x}^{1,1}, \boldsymbol{X}^{1,1})$ of $LP(\Omega^{1,1})$. If $LP(\Omega^{1,1})$ is infeasible, then the original problem ($PP(\Omega)$) is also infeasible and the algorithm terminates.
- Select branching variable x_p using the branching rule (1.1). If $\theta_p^{k,t_1} = 0$, then $\boldsymbol{x}^{1,1}$ is feasible in ($PP(\Omega)$) and hence, it is an optimal solution of ($PP(\Omega)$) and the algorithm terminates. Otherwise continue to Stage 1.

Stage 1. Take problem $LP(\Omega^{k,t})$ and let x_p be the branching variable, with $\theta_p^{k,t} > 0$. Since $\theta_p^{k,t} > 0$, because of Lemma 1.4, it holds that $l_p^{k,t} < x_p^{k,t} < u_p^{k,t}$, where $l_p^{k,t}$ and $u_p^{k,t}$ are the lower and upper bound, respectively, of x_p in $LP(\Omega^{k,t})$. The following

subsets of $\Omega^{k,t}$ are defined:

$$\begin{aligned}\Omega^{k,t_1} &= \Omega^{k,t} \cap \{\boldsymbol{x} \in \mathbb{R}^n : l_p^{k,t} \leq x_p \leq x_p^{k,t}\} \text{ and} \\ \Omega^{k,t_2} &= \Omega^{k,t} \cap \{\boldsymbol{x} \in \mathbb{R}^n : x_p^{k,t} \leq x_p \leq u_p^{k,t}\},\end{aligned} \tag{1.2}$$

taking indexes $t_1 = tot + 1$ and $t_2 = tot + 2$. Update $Q_k = (Q_k \setminus \{t\}) \cup \{t_1, t_2\}$, $tot = tot + 2$, and continue to Stage 2.

Stage 2a. Solve $LP(\Omega^{k,t_1})$. If it is infeasible, continue to Stage 2b. Otherwise, let $(\boldsymbol{x}^{k,t_1}, \boldsymbol{X}^{k,t_1})$ be the optimal solution of $LP(\Omega^{k,t_1})$ with an optimal value of $LB_{k,t_1} = v[LP(\Omega^{k,t_1})]$. Select now the branching variable x_p using (1.1).

- If $\theta_p^{k,t_1} = 0$ then $\boldsymbol{x}^{k,t_1}$ is feasible for the original problem ($PP(\Omega)$) and we update $Q_k = Q_k \setminus \{t_1\}$. If $v[LP(\Omega^{k,t_1})] < v^*$, then $\boldsymbol{x}^* = \boldsymbol{x}^{k,t_1}$ and $v^* = LB_{k,t_1}$ are updated.

Stage 2b. Solve $LP(\Omega^{k,t_2})$. If it is infeasible, continue to Stage 3. Otherwise, let $(\boldsymbol{x}^{k,t_2}, \boldsymbol{X}^{k,t_2})$ be the optimal solution of $LP(\Omega^{k,t_2})$ with an optimal value of $LB_{k,t_2} = v[LP(\Omega^{k,t_2})]$. Select now the branching variable x_p using (1.1).

- If $\theta_p^{k,t_2} = 0$ then $\boldsymbol{x}^{k,t_2}$ is feasible for the original problem ($PP(\Omega)$) and we update $Q_k = Q_k \setminus \{t_2\}$. If $v[LP(\Omega^{k,t_2})] < v^*$, then $\boldsymbol{x}^* = \boldsymbol{x}^{k,t_2}$ and $v^* = LB_{k,t_2}$ are updated.

Stage 3. Update $Q_{k+1} = Q_k \setminus \{t \in Q_k : LB_{k,t} \geq v^*\}$ (fathoming). If $Q_{k+1} = \emptyset$ the algorithm terminates. Otherwise, for each $t \in Q_{k+1}$, we update $\Omega^{k+1,t} = \Omega^{k,t}$, $(\boldsymbol{x}^{k+1,t}, \boldsymbol{X}^{k+1,t}) = (\boldsymbol{x}^{k,t}, \boldsymbol{X}^{k,t})$, $LB_{k+1,t} = LB_{k,t}$, and $\theta_p^{k+1,t} = \theta_p^{k,t}$ and let $k = k + 1$.

Stage 4. Select (k, t) where $t \in \arg\min_{t \in Q_k}\{LB_{k,t}\}$ and go to Stage 1.

Sherali and Tuncbilek (1992) prove that the previous algorithm converges to a global optimum of ($PP(\Omega)$). In this thesis, we include a proof based on the one from Sherali and Tuncbilek (1992), but making explicit some arguments that are omitted there, due to the fact that later on we rely on it to prove that certain variations of the RLT-based algorithm maintain convergence.

Lemma 1.5. *For any infinite branch generated by the RLT-based algorithm, it holds that* $v[PP(\Omega)] \geq LB_{k,t(k)}$ *for each* $k \in K \subset \mathbb{N}$, *being* K *an infinite set.*

Proof. By the construction of the algorithm, we never select nodes with an optimal value greater than the optimum of the problem. This is true because, at each iteration k, it holds the following:

(1) Any point $(\boldsymbol{x}, E(\boldsymbol{x}))$ feasible in $LP(\Omega^{k,t})$, with $\boldsymbol{x}$ feasible in $(PP(\Omega))$, is feasible either in $LP(\Omega^{k,t_1})$ or in $LP(\Omega^{k,t_2})$. Indeed, this is derived from how branching is done in (1.2) and the expression of the bound-factor constraints.

(2) The RLT-based algorithm only closes a branch at node $LP(\Omega^{k,t(k)})$ if the node is infeasible or if $|\bar{X}_{J\cup\{j\}} - \bar{x}_j \bar{X}_J| = 0$ for each $J \subset N^\delta$, $1 \leq |J| \leq \delta - 1$, and $j \in N$, being $(\bar{\boldsymbol{x}}^{k,t(k)}, \bar{\boldsymbol{X}}^{k,t(k)})$ the optimal solution of $LP(\Omega^{k,t(k)})$.

(3) We select the node by taking from Q_k the one with the lowest optimal value.

(4) At Stage 3, we update $Q_{k+1} = Q_k \setminus \{t \in Q_k : LB_{k,t} \geq v^*\}$, that is, we eliminate from Q_k the nodes with an optimal value greater than or equal to the best solution found at iteration k. □

Theorem 1.6. *Using the RLT-based algorithm to solve $(PP(\Omega))$, one of the two following claims holds true:*

(1) The RLT-based algorithm terminates finitely with the incumbent solution being optimal to $(PP(\Omega))$.

(2) An infinite sequence of iterations is generated such that, along any infinite branch of the branch-and-bound tree, any accumulation point of the sequence of variables $\boldsymbol{x}$ generated solving linear relaxations is a global optimum of $(PP(\Omega))$.

Proof. First note that, at Stage 1 of the algorithm, all the points $(\boldsymbol{x}, E(\boldsymbol{x}))$ feasible in $LP(\Omega^{k,t})$, verifying that $\boldsymbol{x}$ is feasible in $(PP(\Omega))$, are feasible either in $LP(\Omega^{k,t_1})$ or in $LP(\Omega^{k,t_2})$. Indeed, this is derived from how branching is done in (1.2) and the expression of the bound-factor constraints. Furthermore, if the algorithm terminates in a finite number of iterations k, then $Q_k = \emptyset$. Thus, the value of v^* at iteration k is the optimal one from $(PP(\Omega))$, because all the nodes with a lower optimal value were already solved before iteration k and v^* is the best value found for a feasible solution of $(PP(\Omega))$.

Suppose then that the algorithm does not terminate in a finite number of iterations and consider an arbitrary infinite branch of the branch-and-bound tree. Let $\{\Omega^{k,t(k)}\}_{k\in K}$ be the sequence of nested partitions of that branch, where $t(k) \in Q_k$ for each $k \in K \subset \mathbb{N}$, being K an infinite set. Let $(\boldsymbol{x}^{k,t(k)}, \boldsymbol{X}^{k,t(k)})$ be the solution of the node $LP(\Omega^{k,t(k)})$. The sequence $\{\boldsymbol{x}^{k,t(k)}\}$ is bounded because $\{\boldsymbol{x}^{k,t(k)}\}$ belongs to the bounded set Ω. Therefore, there is at least one convergent subsequence (Bolzano-Weierstrass theorem). Let $\{\boldsymbol{x}^{k,t(k)}\}_{k\in K_1} \to \bar{\boldsymbol{x}}$ be an arbitrary convergent subsequence, where $K_1 \subset K$ is an infinite set. The proof is reduced to proving that $\bar{\boldsymbol{x}}$ is an optimal solution of $(PP(\Omega))$, since any accumulation point is the limit of such a subsequence.

Since $\{\Omega^{k,t(k)}\}_{k\in K_1}$ is an infinite sequence, there is at least one variable in which the algorithm branches an infinite number of times. Let x_p be one such variable and let $K_2 \subset K_1$ be the set of all iterations $k \in K_1$ in which the branching variable is x_p. Moreover, there is an infinite set $K_3 \subset K_2$ and a set of indexes $J' \subset N^\delta$, with $1 \leq |J'| \leq \delta - 1$, such that

$$\left|X^{k,t(k)}_{J'\cup\{p\}} - x^{k,t(k)}_p X^{k,t(k)}_{J'}\right| \geq \left|X^{k,t(k)}_{J\cup\{j\}} - x^{k,t(k)}_j X^{k,t(k)}_{J}\right| \tag{1.3}$$

for each $k \in K_3$, $J \subset N^\delta$, $1 \leq |J| \leq \delta - 1$, and $j \in N$.

Now, since the sequence $\{(\boldsymbol{x}^{k,t(k)}, \boldsymbol{X}^{k,t(k)}, \boldsymbol{l}^{k,t(k)}, \boldsymbol{u}^{k,t(k)})\}_{k\in K_3}$ is bounded because of Corollary 1.3, there is a convergent subsequence $\{(\boldsymbol{x}^{k,t(k)}, \boldsymbol{X}^{k,t(k)}, \boldsymbol{l}^{k,t(k)}, \boldsymbol{u}^{k,t(k)})\}_{k\in K_4} \to (\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}}, \bar{\boldsymbol{l}}, \bar{\boldsymbol{u}})$, where $K_4 \subset K_3$. Therefore, it holds that $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ is feasible in $LP(\bar{\Omega})$ being $\bar{\Omega} = \{\boldsymbol{x} \in \mathbb{R}^n : 0 \leq \bar{l}_j \leq x_j \leq \bar{u}_j \text{ for each } j \in N\}$.

Moreover, taking into account how the branching is done in (1.2), it holds that, for each $k, k' \in K_4$ with $k' > k$, $x^{k,t(k)}_p \notin (l^{k',t(k')}_p, u^{k',t(k')}_p)$. Therefore, since $\bar{x}_p \in [\bar{l}_p, \bar{u}_p]$, either $\bar{x}_p = \bar{l}_p$ or $\bar{x}_p = \bar{u}_p$. Thus, by Lemma 1.4, $\bar{X}_{J'\cup\{p\}} = \bar{x}_p \bar{X}_{J'}$.

As a consequence, using (1.3) and making k tend to ∞, it holds that $\bar{X}_{J\cup\{j\}} = \bar{x}_j \bar{X}_J$ for each $J \subset N^\delta$, $1 \leq |J| \leq \delta - 1$, and $j \in N$. Thus, $\bar{\boldsymbol{x}}$ is feasible in $(PP(\Omega))$ and $[\phi_0]_L(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}}) = \phi_0(\bar{\boldsymbol{x}}) \geq v[PP(\Omega)]$, where $[\phi_0]_L$ is the objective function of $(LP(\Omega))$ and ϕ_0 the objective function of $(PP(\Omega))$.

Finally, using Lemma 1.5 and making k tend to ∞, it holds that $v[PP(\Omega)] \geq [\phi_0]_L(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}}) = \phi_0(\bar{\boldsymbol{x}})$. Thus, $\bar{\boldsymbol{x}}$ is an optimal solution of $(PP(\Omega))$. □

1.4 Theoretical Extensions of the RLT-Based Algorithm

The aim of this section is to prove that previous results are also valid for some generic variations of the RLT-based algorithm. Along this thesis, we present some enhancements for the RLT-based algorithm and these results guarantee the convergence of the resulting algorithm.

The following result allows us to modify the branching rule of the RLT-based algorithm without compromising the convergence of the algorithm. Moreover, we can give weights to the different variables of the problem, which opens a range of possibilities to prioritise the violations produced by certain variables. This kind of modifications are studied in detail in Section 2.5.4 and Chapter 3.

Theorem 1.7. *If the branching rule (1.1) is changed to one of the following family*

$$p \in arg\ max_{j\in N}\{\theta_j\},\ where$$
$$\theta_j = \sum_{J\subset N^\delta, 1\leq|J|\leq\delta-1} w(j,J)|\bar{X}_{J\cup\{j\}} - \bar{x}_j\bar{X}_J|\ for\ each\ j\in N, \tag{1.4}$$

where $w(j,J) > 0$ for each $j \in N$ and $J \subset N^\delta$, then the algorithm maintains the convergence to a global optimum.

Proof. First, note that Lemma 1.1, Lemma 1.2, and Lemma 1.4 only apply to the first linear relaxation $LP(\Omega)$. Then, any change in the branching rule does not affect the proof of the lemmas.

Regarding Lemma 1.5, when we change the branching rule, we might invalidate the result because we might close a branch with $\theta_j = 0$, for each $j \in N$, but with $\bar{X}_J^{k,t(k)} \neq \prod_{j\in J}\bar{x}_j^{k,t(k)}$ for some $J \subset N^\delta$, and invalidate (2). Nevertheless, that is not possible using the branching rule (1.4), due to the fact that $\theta_j = 0$ if and only if $\bar{X}_{J\cup\{j\}} - \bar{x}_j\bar{X}_J = 0$ for each $J \subset N^\delta$ and $j \in N$. Moreover, a change in the branching rule does not affect claims (1), (3), and (4).

Finally, regarding the proof of Theorem 1.6, all arguments remain trivially true. □

Definition 1.11. A linear constraint of the form $f(\boldsymbol{x}, \boldsymbol{X}) \geq 0$ (or $f(\boldsymbol{x}, \boldsymbol{X}) = 0$) is a *valid cut* for ($PP(\Omega)$) if, for each feasible point $\boldsymbol{x}$ in ($PP(\Omega)$), it holds that $f(\boldsymbol{x}, E(\boldsymbol{x})) \geq 0$ ($f(\boldsymbol{x}, E(\boldsymbol{x})) = 0$).

The following result ensures that we can add valid cuts to the linear relaxations generated by the RLT-based algorithm without compromising its convergence. This allows, in certain cases, to reduce the number of iterations of the algorithm, as well as to speed up the process of solving the linear relaxations. This is studied in detail in Section 2.5.6 and in Chapter 4.

Theorem 1.8. *Let $\{f_s(\boldsymbol{x}, \boldsymbol{X}) \geq 0\}_{s\in S_1}$ and $\{f_s(\boldsymbol{x}, \boldsymbol{X}) = 0\}_{s\in S_2}$ be a collection of valid cuts. Then, any subcollection of these valid cuts can be added to any of the linear relaxations of the branch-and-bound tree, maintaining the convergence of the RLT-based algorithm to a global optimum.*

Proof. Proof of Lemma 1.1 remains true because the cuts do not change to objective function of ($LP(\Omega)$) and, for each $\boldsymbol{x}$ feasible in ($PP(\Omega)$), $(\boldsymbol{x}, E(\boldsymbol{x}))$ is feasible in ($LP(\Omega)$) with and without the valid cuts.

Proof of Lemma 1.2 is valid because is only about bound-factor constraints, not the linear relaxations themselves.

Proof of Lemma 1.4 remains true because in order to prove the result, only bound-factor constraints are used.

Regarding Lemma 1.5, claims (2), (3), and (4) remain true. Moreover, taking into account that valid cuts do not eliminate in the linear relaxations any point $(\boldsymbol{x}, E(\boldsymbol{x}))$, with $\boldsymbol{x}$ feasible in $(PP(\Omega))$, (1) remains also true.

Finally, regarding the proof of Theorem 1.6, all arguments remain trivially true. □

Therefore, additional constraints can be added to linear relaxations of the branch-and-bound tree without compromising the convergence of the algorithm, as long as such constraints are valid cuts.

Finally, we present a theorem which provides a natural extension of the RLT-based algorithm for mixed-integer polynomial programming problems.

Definition 1.12. A mixed-integer *polynomial optimization problem* is an optimization problem of the following form:

$$\begin{array}{rll} \text{minimize} & \phi_0(\boldsymbol{x}) & \\ \text{subject to} & \phi_r(\boldsymbol{x}) \geq \beta_r, & r = 1, \ldots, R_1 \\ & \phi_r(\boldsymbol{x}) = \beta_r, & r = R_1 + 1, \ldots, R \\ & \boldsymbol{x} \in \Omega \subset \mathbb{R}^n & \\ & x_j \in \mathbb{Z}, & j \in N_{\mathrm{I}} \subset N. \end{array} \qquad (PP^I(\Omega))$$

We define its (mixed-integer) linear relaxation as the following mixed-integer linear programming problem:

$$\begin{array}{rll} \text{minimize} & [\phi_0(\boldsymbol{x})]_L & \\ \text{subject to} & [\phi_r(\boldsymbol{x})]_L \geq \beta_r, & r = 1, \ldots, R_1 \\ & [\phi_r(\boldsymbol{x})]_L = \beta_r, & r = R_1 + 1, \ldots, R \\ & [\prod_{j \in J_1}(x_j - l_j) \prod_{j \in J_2}(u_j - x_j)]_L \geq 0, & \forall J_1 \cup J_2 \subset N^\delta, \ |J_1 \cup J_2| = \delta \\ & \boldsymbol{x} \in \Omega \subset \mathbb{R}^n & \\ & x_j \in \mathbb{Z}, & j \in N_{\mathrm{I}} \subset N. \end{array} \qquad (LP^I(\Omega))$$

Theorem 1.9. *The RLT-based algorithm converges to a global optimum of* $(PP^I(\Omega))$ *if we replace* $(LP(\Omega))$ *relaxations with* $(LP^I(\Omega))$ *ones.*

Proof. First, regarding the proof of Lemma 1.1, it remains true because $(LP^I(\Omega))$ and $(LP(\Omega))$ have the same objective function and $(\boldsymbol{x}, E(\boldsymbol{x}))$ is feasible in $(LP^I(\Omega))$ if it is feasible in $(LP(\Omega))$.

Lemma 1.2 remains true because it is only related to the properties of bound-factor constraints and all the equalities used in the proof are valid regardless these new integer constraints.

Regarding Lemma 1.4, the proof remains valid because only bound-factor constraints are used.

The proof of Lemma 1.5 remains valid because integer constraints are the same at ($PP^I(\Omega)$) and at ($LP^I(\Omega)$) and then, (1) remains true. Furthermore, (2), (3), and (4) remain also true.

Finally, regarding Theorem 1.6, all arguments remain trivially true. □

Bibliography

Ahmadi, A. A. and A. Majumdar (2016): "Some applications of polynomial optimization in operations research and real-time decision making," *Optimization Letters*, 10, 709–729.

Dalkiran, E. and H. D. Sherali (2013): "Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality," *Journal of Global Optimization*, 57, 1147–1172.

Ghaddar, B., M. Claeys, M. Mevissen, and B. J.Eck (2017): "Polynomial optimization for water networks: global solutions for the valve setting problem," *European Journal of Operational Research*, 261, 450–459.

Ghaddar, B., J. Marecek, and M. Mevissen (2016): "Optimal power flow as a polynomial optimization problem," *IEEE Transactions on Power Systems*, 31, 539–546.

He, S., Z. Li, and S. Zhang (2010): "Approximation algorithms for homogeneous polynomial optimization with quadratic constraints," *Mathematical Programming*, 125, 353–383.

Hägglöf, K., P. O. Lindberg, and L. Svensson (1995): "Computing global minima to polynomial optimization problems using Gröbner bases," *Journal of Global Optimization*, 7, 115–125.

Kamyar, R. and M. M. Peet (2015): "Polynomial optimization with applications to stability analysis and control - alternatives to sum of squares," *Discrete and Continuous Dynamical Systems - Series B*, 20, 2383–2417.

LASSERRE, J. B. (2006): "Convergent SDP-relaxations in polynomial optimization with sparsity," *SIAM Journal on Optimization*, 17, 822–843.

SHERALI, H. D., E. DALKIRAN, AND J. DESAI (2012a): "Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts," *Computational Optimization and Applications*, 52, 483–506.

SHERALI, H. D., E. DALKIRAN, AND L. LIBERTI (2012b): "Reduced RLT representations for nonconvex polynomial programming problems," *Journal of Global Optimization*, 52, 447–469.

SHERALI, H. D. AND C. H. TUNCBILEK (1992): "A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique," *Journal of Global Optimization*, 2, 101–112.

Chapter 2

Enhancing the RLT-Based Algorithm: RAPOSa, a New Solver

2.1 Introduction

This chapter is based on González-Rodríguez et al. (2022) and we discuss some enhancements of the RLT-based algorithm, introducing `RAPOSa` (**R**eformulation **A**lgorithm for **P**olynomial **O**ptimization - **Sa**ntiago), a new global optimization solver specifically designed for polynomial programming problems with box-constrained variables. It is a C++ implementation of the RLT-based branch-and-bound algorithm introduced in Sherali and Tuncbilek (1992), explained in detail in Chapter 1. Although it is not open source, `RAPOSa` is freely distributed and available for Linux, Windows and MacOS. It can also be run from `AMPL` (Fourer et al., 1990) and from NEOS Server (Czyzyk et al., 1998). The RLT-based scheme in `RAPOSa` solves polynomial programming problems by successive linearizations embedded into a spatial branch-and-bound scheme. At each iteration, a linear solver must be called, and `RAPOSa` has been integrated with a wide variety of linear optimization solvers, both open source and commercial, including those available via Google OR-tools (Perron and Furnon, 2022). Further, auxiliary calls to nonlinear solvers are also performed along the branch-and-bound tree to improve the performance of the algorithm, and again both open source and commercial solvers are supported as long as they can be called from `.nl` files (Gay, 1997, 2005). More information about `RAPOSa` can be found at `https://raposa.usc.es`.

In conjunction with the introduction of `RAPOSa`, the other major contribution of this chapter is to study the impact of different enhancements on the performance of the RLT-based algorithm. We discuss not only the individual impact of a series of enhancements, but also the impact of combining them. To this end, Section 2.4 and Section 2.5 contain computational analyses on the use of J-sets (Dalkiran and Sherali, 2013), warm starting of the linear relaxations, changes in the branching rule, the introduction of bound tightening techniques (Belotti et al., 2009, 2012; Puranik and Sahinidis, 2017) and the addition

of SDP cuts (Sherali et al., 2012a), among others. Interestingly, RAPOSa incorporates a fine-grained distributed parallelization of the branch-and-bound core algorithm, which delivers promising speedups as the number of available cores increases.

The most competitive configurations of RAPOSa according to the preceding extensive analysis are then compared to three popular state-of-the-art global optimization solvers: BARON (Tawarmalani and Sahinidis, 2005; Sahinidis, 2021), Couenne (Belotti et al., 2009) and SCIP (Bestuzheva et al., 2021). The computational analysis is performed on two different data sets. The first one, DS, is a set of randomly generated polynomial programming problems of different degree introduced in Dalkiran and Sherali (2016) when studying their own RLT implementation: RLT-POS.[1] The second data set, MINLPLib, contains the polynomial programming problems with box-constrained and continuous variables available in MINLPLib (Bussieck et al., 2003). The main results can be summarized as follows: i) in DS, all configurations of RAPOSa clearly outperform BARON, Couenne and SCIP, with the latter performing significantly worse than all the other solvers and ii) in MINLPLib, differences in performance are smaller across solvers, with SCIP exhibiting a slightly superior performance. Importantly, the enhanced versions of RAPOSa are clearly superior in this data set to the baseline configuration.

The outline of the chapter is as follows. In Section 2.2 we present different enhancements that have been introduced in recent years. In Section 2.3 we discuss some specifics of the implementation of RAPOSa and of the testing environment. In Section 2.4 we present some preliminary computational results, in order to define a configuration of RAPOSa that can be used as the baseline to assess the impact of the main enhancements, discussed in Section 2.5. In Section 2.6 we present the comparative study with BARON, Couenne and SCIP. Finally, we conclude in Section 2.7.

2.2 RLT: State of the Art and Main Contributions

2.2.1 Enhancements of the Original RLT-Based Algorithm

In this section we briefly discuss six enhancements of the basic RLT-based algorithm. All of them are part of the current implementation of RAPOSa and its impact on the performance of the algorithm is thoroughly analysed in Section 2.4 and Section 2.5.

[1] Unfortunately, we could not include RLT-POS in our comparative study, since this implementation is not publicly available.

J-sets

This enhancement was introduced in Dalkiran and Sherali (2013), where the authors prove that it is not necessary to consider all the bound-factor constraints in the linear relaxation ($LP(\Omega)$). Specifically, they prove two main results. The first one is that convergence to a global optimum is ensured even if only the bound-factor constraints associated with the monomials that appear in the original problem are included in the linear relaxation. The second result is that convergence to a global optimum is also ensured without adding the bound-factor constraints associated with monomials $J \subset J'$, provided the bound-factor constraints associated with J' are already incorporated. Equipped with these two results, the authors identify a collection of monomials, which they call J-sets, such that convergence to a global optimum is still guaranteed if only the bound-factor constraints associated with these monomials are considered.

The use of J-sets notably reduces the number of bound-factor constraints in the linear relaxation ($LP(\Omega)$) (although it still grows exponentially fast as the size of the problem increases). The main benefit of this reduction is that it leads to smaller LP relaxations and, hence, the RLT-based algorithm requires less time in each iteration. The drawback is that the linear relaxations become less tight because they have less constraints and more iterations may be required for convergence. Nevertheless, practice has shown that the approach with J-sets is clearly superior. We corroborate this fact in the numerical analysis of Section 2.4.1.

It is important to note that the theoretical results in Chapter 1 can be easily adapted to accommodate the use of J-sets and, therefore, the convergence of the RLT-based algorithm is still guaranteed.

Use of an Auxiliary Local NLP Solver

Most branch-and-bound algorithms for global optimization rely on auxiliary local solvers and the RLT-based algorithm can also profit from them, as already discussed in Dalkiran and Sherali (2016). They proposed to call the nonlinear local solver at certain nodes of the branch-and-bound tree. In each call, the nonlinear local solver is provided with an initial solution, which is the one associated to the lower bound of the RLT-based algorithm at that moment.

This strategy helps to decrease the upper bound more rapidly and, hence, allows to close the optimality gap more quickly. The only drawback is the time used in the call to the nonlinear solver. Practice has shown that, in general, it is beneficial to call it only at certain nodes instead of doing so at each and every node.

Products of Constraint Factors and Bounding Factors

This enhancement was already mentioned in Sherali and Tuncbilek (1992) when first introducing RLT for polynomial programming problems. It consists of defining tighter linear programming relaxations by strengthening *constraint factors* of the form $\phi_r(\boldsymbol{x}) - \beta_r \geq 0$ of degree less than δ. More precisely, one should take products of these constraint factors and/or products of bounding factors in such a way that the resulting degree is no more than δ. Similar strengthening can also be associated to equality constraints $\phi_r(\boldsymbol{x}) = \beta_r$ of degree less than δ, by multiplying them by variables of the original problem.

Note that, although the new constraints are tighter than the original ones in the nonlinear problem, this is not necessarily so in the linear relaxations. Thus, we also preserve the original constraints to ensure that the resulting relaxations are indeed tighter and, therefore, the lower bound may increase more rapidly. Since the addition of many of these stronger constraints may complicate the solution of the linear relaxations, one should carefully balance these two opposing effects.

Branching Rule

The design of a branch-and-bound algorithm requires to properly define different components of the algorithm. The most widely studied ones are the search strategy in the resulting tree, pruning rules, and branching rule; refer, for instance, to Achterberg et al. (2005) and Morrison et al. (2016). In Section 2.5.4 we focus on the latter of these components. More precisely, we study the impact of the rule for the selection of the branching variable on the performance of the RLT-based algorithm.

Bound Tightening

Bound tightening techniques are at the core of most global optimization algorithms for nonlinear problems (Belotti et al., 2009, 2012; Puranik and Sahinidis, 2017). These techniques allow to reduce the search space of the algorithm by adjusting the bounds of the variables of the problem. Two main approaches have been discussed in the literature: i) optimality-based bound tightening, OBBT, in which bounds are tightened by solving a series of relaxations of minor variations of the original problem and ii) feasibility-based bound tightening, FBBT, in which tighter bounds are deduced directly by exploring the problem constraints. Since OBBT is computationally demanding while FBBT is not, combined schemes are often used, under which OBBT is only performed at the root node and FBBT is performed at each and every node of the branch-and-bound tree.

These techniques lead to tighter relaxations and, therefore, they do not only reduce the search space, but they also help to increase the lower bound of the optimization algorithm more rapidly. Moreover, since the resulting linear relaxations are not harder to solve than the original ones, bound tightening techniques are often very effective at improving the performance of global optimization algorithms.

SDP Cuts

This enhancement is introduced in Sherali et al. (2012a) and consists of adding specific linear constraints, called positive semidefinite cuts (SDP cuts), to the linear relaxations generated along the branch-and-bound tree. These constraints are built as follows. First, a matrix of the form $\boldsymbol{M} = [\boldsymbol{y}\boldsymbol{y}^\intercal]$ is defined, where $\boldsymbol{y}$ can be any vector defined using variables and products of variables of the original problem. By definition, matrix $\boldsymbol{M}$ is positive semidefinite in any feasible solution, because it is defined by the product of a vector by itself. Now, let $\boldsymbol{M}_L = [\boldsymbol{y}\boldsymbol{y}^\intercal]_L$ be the matrix obtained when each monomial in $\boldsymbol{M}$ is replaced by the corresponding RLT variable. Therefore, given a solution of the current linear relaxation, we can evaluate $\boldsymbol{M}_L$ at this solution, obtaining matrix $\bar{\boldsymbol{M}}_L$. If this matrix is not positive semidefinite, then we can identify a valid linear cut to be added to the linear relaxation. More precisely, if there is a vector $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}^\intercal \bar{\boldsymbol{M}}_L \boldsymbol{\alpha} < 0$, then constraint $\boldsymbol{\alpha}^\intercal \boldsymbol{M}_L \boldsymbol{\alpha} \geq 0$ is added to the linear relaxation.

In Sherali et al. (2012a) this process is thoroughly explained and several strategies are discussed, such as different approaches to take vector $\boldsymbol{y}$, different methods to find $\boldsymbol{\alpha}$, and the number of cuts to add in each iteration.

2.3 Implementation and Testing Environment

2.3.1 Implementation

Each execution of `RAPOSa` leans on two solvers: one for solving the linear problems generated in the branch-and-bound process and one to compute upper bounds as mentioned in Section 2.2.1. The default (free) configuration of `RAPOSa` runs with `Glop` (Backer et al., 2015) and `Ipopt` (Wächter and Biegler, 2006), although the best results are obtained using commercial solver `Gurobi` (Gurobi Optimization, LLC, 2022) instead of `Glop`. Currently, `RAPOSa` supports the following solvers:

- Linear solvers: `Gurobi` (Gurobi Optimization, LLC, 2022), `Glop` (Backer et al., 2015), and `CLP` (Forrest et al., 2020).[2]

[2]The connection with `Glop` and `CLP` was implemented using Google OR-tools (Perron and Furnon, 2022).

- Nonlinear local solvers: `Ipopt` (Wächter and Biegler, 2006), `Knitro` (Byrd et al., 2006), `MINOS` (Murtagh and Saunders, 1978), and `CONOPT` (Drud, 1985).

`RAPOSa` has been implemented in `C++`, and it is important to clarify that it connects differently to the two types of solvers. Since solving the linear problems is the most critical part of the performance, it connects with the linear solvers through their respective `C++` libraries. In the case of the nonlinear local solvers, the number of calls is significantly lower, and `RAPOSa` sends them an intermediate `.nl` file with the original problem and a starting point.

Importantly, the user does not need to explicitly generate `.nl` files to execute `RAPOSa`, since it can also be run from an `AMPL` interface (Fourer et al., 1990). Moreover, `RAPOSa` can also be executed on NEOS Server (Czyzyk et al., 1998).

2.3.2 The Testing Environment

All the executions reported in this chapter have been performed on the supercomputer Finisterrae II, provided by Galicia Supercomputing Centre (CESGA). Specifically, we used computational nodes powered with 2 deca-core Intel Haswell 2680v3 CPUs with 128GB of RAM connected through an Infiniband FDR network, and 1TB of hard drive.

Regarding the data sets, we use two different sets of problems. The first one is taken from Dalkiran and Sherali (2016) and consists of 180 instances of randomly generated polynomial programming problems of different degrees, number of variables, and density.[3] The second data set comes from the well-known benchmark MINLPLib (Bussieck et al., 2003), a library of mixed-integer nonlinear programming problems. We have selected from MINLPLib those instances that are polynomial programming problems with box-constrained and continuous variables, resulting in a total of 168 instances. Hereafter we refer to the first data set as DS and to the second one as MINLPLib.[4]

All solvers have been run taking as stopping criterion that the relative or absolute gap is below the threshold 0.001. The time limit was set to 10 minutes in comparisons between different configurations of `RAPOSa` and to 1 hour in the comparison between `RAPOSa` and other solvers.

[3] By density we mean the proportion of monomials in the problem to the total number of possible monomials (given the degree of the problem).

[4] Instances from DS can be downloaded at `https://raposa.usc.es/files/DS-TS.zip` and instances from MINLPLib can be downloaded at `https://raposa.usc.es/files/MINLPLib-TS.zip`.

2.4 Preliminary Results and RAPOSa's Baseline Configuration

The main goal of this section is to add to `RAPOSa`'s RLT basic implementation a minimal set of enhancements that make it robust enough to define a baseline version to assess, in Section 2.5, the impact of the rest of the enhancements. First, in Section 2.4.1 we show that the inclusion of both the J-sets enhancement and the auxiliary nonlinear solver are crucial in order to be able to get a competitive solver when tackling the problems in DS and MINLPLib. Then, in Section 2.4.2 we present a comparison of the performance of different LP solvers and thereafter all new enhancements are validated and tested on the best performing one. Last, but not least, in Section 2.4.3 we present the results of a parallel version of `RAPOSa`'s RLT implementation, to illustrate the potential of improvement of this type of branch-and-bound algorithms when run on multi-core processors. Yet, in order to provide fair comparisons in the rest of the paper, particularly in Section 2.6, these parallelization capabilities won't be used beyond Section 2.4.3.

Before starting to go over the numerical results, we briefly explain the tables and figures used to discuss them. The main reporting tool is a series of summary tables. Each of these tables contains two blocks of five rows, one for each data set, and as many columns as configurations of `RAPOSa` or solvers are being compared. The information of these rows is as follows:

Solved. Number of solved instances. In brackets we show number of instances solved by at least one configuration and the total number of instances in the corresponding data set.

Gap $= \infty$**.** Number of instances in which the algorithm terminated with an infinite optimality gap. In brackets we show number of instances with an infinite gap for all configurations and, again, the total number of instances in the corresponding data set.

Time. Geometric mean time,[5] but disregarding those instances solved by all configurations in less than 5 seconds and also those not solved by any configuration within the time limit. In brackets we show the remaining number of instances.

Gap. Geometric mean gap, but with the following considerations: i) instances solved by all configurations under study are discarded, ii) instances for which no configuration

[5]The use of geometric means is becoming the standard in benchmarking, since they are less sensitive to outliers than the standard mean.

could return a gap after the time limit are also discarded, and iii) when a configuration is not able to return a lower or upper bound after the time limit, we assign to it a relative optimality gap of 10^5. In brackets we show the remaining number of instances.

Nodes. Geometric mean of the number of nodes generated by the RLT-based algorithm in instances solved by all configurations. In brackets we show the number of such instances.

Moreover, for each table discussed in the text there are two associated performance profiles (Dolan and Moré, 2002) for each data set although, for the sake of brevity, most of them have been relegated to Appendix 2.A. The first performance profile is for the running times and the second one for the relative optimality gaps. They contain, respectively, the instances involved in the computations of the geometric mean times and the geometric mean gaps as described above (in brackets we show the corresponding number of instances). In the x-axis we represent ratios of running times or relative optimality gaps, while in the y-axis we represent the percentage of instances in which the corresponding configuration has a ratio lower than the value on the x-axis. For each instance, the ratios are computed dividing running times or relative optimality gaps of each configuration by the best configuration in that instance.[6]

2.4.1 *J*-sets and Nonlinear Solver

We start by jointly evaluating the impact of the introduction of J-sets, discussed in Section 2.2.1, and the use of an auxiliary local NLP solver (NLS). Regarding the latter, `Ipopt` is run at the root node and whenever the total number of solved nodes in the branch-and-bound tree is a power of two. We tried other strategies, but we did not observe a significant impact on the resulting performance. In Table 2.1 we present the results of four different configurations: i) `RAPOSa`'s RLT basic implementation (No J-sets, No NLS), ii) the inclusion of the auxiliary nonlinear solver (No J-sets), iii) the inclusion of J-sets (No NLS), and iv) the inclusion of both the auxiliary nonlinear solver and J-sets (With J-sets and NLS).

The results in Table 2.1 show that J-sets have a huge impact on the performance of the RLT-based algorithm in both data sets. It does not matter whether we assess their impact

[6] It is worth noting that the reliability of performance profiles is sometimes limited when comparing more than two configurations/solvers (Gould and Scott, 2016). Yet, we think that, when complemented with the numeric results in the tables, they help to obtain a clearer image of the numerical analysis developed in this paper.

Data Set		No J-sets, No NLS	No J-sets	No NLS	With J-sets and NLS
DS	solved (128/180)	36	64	106	127
	gap = ∞ (0/180)	144	0	74	0
	geom. time (125)	393.29	−33.36%	−96.39%	−98.13%
	geom. gap (145)	88069.92	−100.00%	−99.99%	−100.00%
	geom. nodes (35)	67.94	−58.63%	−42.03%	−69.56%
MINLPLib	solved (93/124)	69	70	86	90
	gap = ∞ (2/124)	55	4	38	2
	geom. time (36)	175.64	−48.05%	−95.44%	−98.18%
	geom. gap (60)	15848.93	−100.00%	−99.46%	−100.00%
	geom. nodes (63)	58.11	−49.48%	−41.76%	−69.00%

Table 2.1: Impact of J-sets and auxiliary nonlinear solver.

with respect to `RAPOSa`'s RLT basic implementation or with respect to the version that already incorporates the nonlinear solver: there are dramatic gains in all dimensions. What is a bit surprising is that the configurations with J-sets do even reduce the average number of nodes explored in problems solved by all configurations. Since the version without J-sets leads to tighter relaxations, one would expect to observe a faster increase in the lower bounds and a reduction of the total number of explored nodes (at the cost of higher solve time at each node). A careful look at the individual instances reveals that the number of explored nodes turns to be quite close for most instances. Yet, there are a few "outliers" that required many more nodes to be solved for the version without J-sets, producing a large impact on the average and also in the geometric average. It might be worth studying further whether these outliers appeared in the configuration without J-sets by coincidence or if there is some structural reason that leads to this effect.

It is worth noting that for MINLPLib the number of instances reported is 124 out of the 168 instances of this data set. This is because, for the remaining 44 instances, the version without J-sets did not even manage to solve the root node within the time limit. Not only it did not manage to return any bounds, but it ran out of time when generating the linear relaxation at the root node and we removed these instances from the analysis.

We move now to the impact of the inclusion of a local solver. Again, Table 2.1 shows that there is a huge impact on the performance of the RLT-based algorithm in both data sets. Performance improves again in all dimensions, and especially so at closing the gap, since the number of instances for which some bound is missing goes down, with respect to `RAPOSa`'s RLT basic implementation, from 144 to 0 in DS and from 55 to 4 in MINLPLib. Similarly, with respect to the version that already uses J-sets, the number goes down from 74 to 0 in DS and from 38 to 2 in MINLPLib. We have run similar computational tests with different local NLP solvers and the results are quite robust. Therefore, the specific

choice of local solver does not seem to have a significant impact on the final performance of the RLT-based algorithm.

2.4.2 Different LP Solvers

The results of Table 2.1 in the preceding section were obtained using the commercial solver `Gurobi` (Gurobi Optimization, LLC, 2022) for the linear relaxations. We now check to what extent the chosen LP solver can make a difference in the performance of the RLT-based algorithm. To this end, we rerun the executions of the version of `RAPOSa` with J-sets and the auxiliary nonlinear solver but with two open source linear solvers: `Clp` (Forrest et al., 2020) and `Glop` (Backer et al., 2015). The results in Table 2.2 show that `Gurobi`'s performance is superior to both `Clp` and `Glop`. On the other hand, the two open-source solvers are close to one another. This is confirmed by the performance profiles in Figure 2.6 in Appendix 2.A. Thus, for the remainder of this paper, all the configurations of `RAPOSa` are run using `Gurobi` as the solver for the linear relaxations.

Data Set		Clp	Glop	Gurobi
DS	solved (128/180)	119	119	127
	gap = ∞ (0/180)	0	0	0
	geom. time (75)	63.58	+11.36%	−22.91%
	geom. gap (63)	0.03	+0.35%	−66.73%
	geom. nodes (117)	136.70	−1.36%	+0.35%
MINLPLib	solved (102/168)	92	98	99
	gap = ∞ (6/168)	7	9	6
	geom. time (35)	43.79	+4.62%	−61.02%
	geom. gap (74)	0.40	+4.78%	−30.42%
	geom. nodes (89)	84.25	−5.00%	−2.13%

Table 2.2: Impact of different linear solvers.

2.4.3 Parallelized RAPOSa

We now move to an enhancement of a completely different nature. `RAPOSa`, as well as all branch-and-bound algorithms, benefits from parallelization on multi-core processors. The way of exploring the solutions tree in this type of strategy makes solving each node an independent operation, suitable to be distributed through processors in the same or different computational nodes. Hence, `RAPOSa` has been parallelized, adapting the classic master-slave paradigm: a master processor guides the search in the tree, containing a queue with pending-to-solve nodes (leaves), which are sent to a set of worker processors. In this section, we present the computational results of our parallel version of `RAPOSa`, showing the obtained speedup as a function of the number of cores.

More precisely, for each instance, `RAPOSa` was initially run in sequential mode (using one core) and a time limit of 10 minutes. Next, the parallel version was tested varying the numbers of cores and prompted to stop when each execution reached the same degree of convergence as in the nonparallel version. Thus, the analysis focuses on the improvement on the time that the parallel version needs to reach the same result as the sequential one.

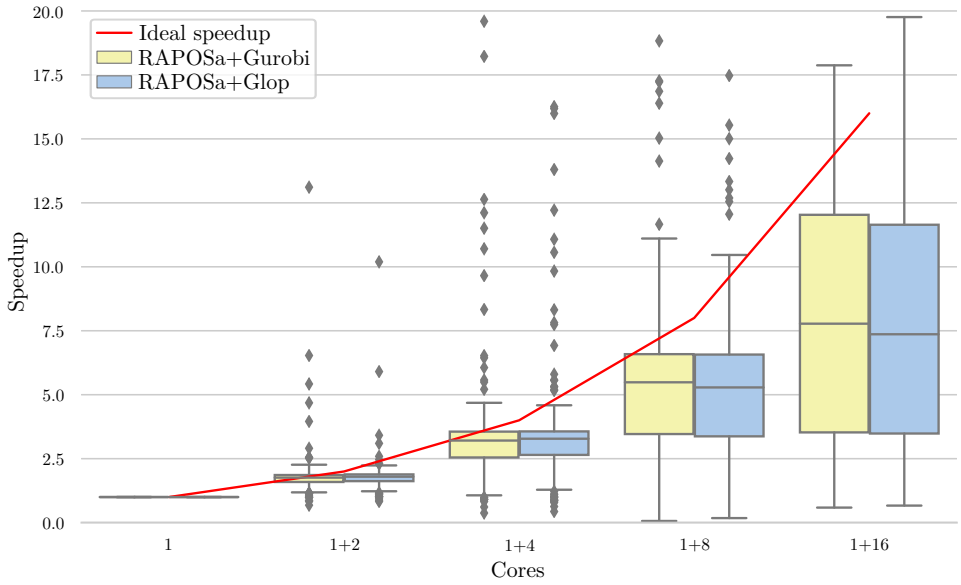


Figure 2.1: Speedup of the parallel version of `RAPOSa` in DS.

Figure 2.1 shows the evolution of the speedup with the number of cores for DS.[7] In the x-axis we represent the number of cores (master+slaves) used by `RAPOSa`, while the y-axis shows the box plot of the speedup on the 180 DS instances. The red line describes the ideal speedup: the number of workers. As can be seen, the scalability of the speedup obtained by the parallel version of `RAPOSa` is generally good, close to the ideal one when the number of cores is small (3 and 5) or with a reasonable performance with a larger number of cores (9 and 17). Furthermore, the speedup does not seem to depend on the linear solver (`Gurobi` or `Glop`) used by `RAPOSa`.

2.5 Computational Analysis of Different Enhancements

In this section we present a series of additional enhancements of the basic implementation of the RLT-based algorithm and try to assess not only the individual impact of each of

[7] We do not include the results for MINLPLib because the majority of the instances are either very easy or very difficult, so the performance of the parallel version is harder to assess.

them, but also the aggregate impact when different enhancements are combined. In order to do so, we define a baseline version of `RAPOSa` that is used as the reference for the analysis, with the different enhancements being added to this baseline. Given the results in the preceding section, this baseline configuration is set to use J-sets, `Ipopt` as the auxiliary local NLP solver[8], and `Gurobi` as the linear one. No parallelization is used.

2.5.1 Warm Generation of J-sets

We start with an enhancement that is essentially about efficient coding, not really about the underlying optimization algorithm. Along the branch-and-bound tree, the only difference between a problem and its father problem is the upper or lower bound of a variable and the bound-factors constraints in which this variable appears. Because of this, it is reasonable to update only those bound-factor constraints that have changed instead of regenerating all the bound-factor constraints of the child node.

It is important to highlight that there are two running times involved in the process. One is the running time used to generate the bound-factor constraints, which is higher in the case that `RAPOSa` regenerates all the bound-factor constraints. The other running time is the one used to identify the bound-factor constraints that change between the parent and the child node. This time is nonexistent in the case that `RAPOSa` regenerates all the bound-factor constraints. As one could expect, the 0.00% percentage of improvement in the last row of Table 2.3a reflects the fact that the warm generation of J-sets has no impact on the resulting tree.

Data Set		Baseline	Warm J-sets
DS	solved (136/180)	127	136
	gap = ∞ (0/180)	0	0
	geom. time (75)	87.78	−19.66%
	geom. gap (53)	0.01	−45.38%
	geom. nodes (127)	137.02	0.00%
MINLPLib	solved (100/168)	99	100
	gap = ∞ (5/168)	6	5
	geom. time (20)	51.81	−28.31%
	geom. gap (64)	0.65	−18.03%
	geom. nodes (99)	110.06	0.00%

(a) Warm generation of J-sets.

	Baseline	Warm start
solved (131/180)	127	129
gap = ∞ (0/180)	0	0
geom. time (69)	79.53	−26.82%
geom. gap (55)	0.01	+56.17%
geom. nodes (125)	136.70	+1.05%
solved (100/168)	99	100
gap = ∞ (6/168)	6	7
geom. time (20)	51.66	−27.91%
geom. gap (63)	0.55	+10.40%
geom. nodes (99)	110.06	+1.10%

(b) Warm start on LP solver.

Table 2.3: Impact of warm generation of J-sets and warm start on LP Solver.

We can see in Table 2.3a that the warm generation of J-sets notably improves the performance of `RAPOSa`, which is also convincingly illustrated by the performance profiles

[8] As we have already mentioned, we have seen in our numerical experiments that the performance of `RAPOSa` is not very sensitive to the chosen local NLP solver.

in Figure 2.7 in Appendix 2.A.

2.5.2 Warm Start on LP Solver

As mentioned before, only a small number of bound-factor constraints are different between the child node and its father node. Because of this, it could be beneficial to feed the linear solver with information regarding the solution of the father problem (optimal solution and optimal basis).

Table 2.3b shows the impact of warm start with respect to the baseline configuration.[9] Differently from the preceding enhancements, the results are somewhat divided now. Warm start reduces the running time in solved instances but, at the same time, the gap in the unsolved ones seems to deteriorate. This suggests that warm start may perform better in relatively easy instances, but not so well in instances that were not solved within the time limit.

2.5.3 Products of Constraint Factors and Bounding Factors

In this section we explore the impact of strengthening the constraints of the original problem with degree less than δ by multiplying them by appropriately chosen bounding factors or variables. Especial care must be taken when choosing these products. The baseline includes the J-sets' enhancement and, hence, we do not want to include products that might increase the number of RLT variables in the resulting relaxations, since this might lead to a large increase in the number of bound-factor constraints.

In order to avoid the above problem, our implementation of this enhancement proceeds as follows. Given a constraint of degree less than δ, we first identify what combinations of bounding factors might be used to multiply the original constraint so that i) no new RLT variables are needed and ii) the resulting degree is at most δ. We then restrict our attention to the combinations that involve more bounding factors and distinguish between two strategies:

More in common. Each inequality constraint is multiplied by bound-factor constraints that involve as many variables already present in the constraint as possible. Similarly, equality constraints are multiplied by as many variables present in the constraint as possible.

Less in common. Each inequality constraint is multiplied by bound-factor constraints that involve as many variables not present in the constraint as possible. Similarly,

[9] Currently, `RAPOSa` only supports warm start when run with `Gurobi` as linear solver.

equality constraints are multiplied by as many variables not present in the constraint as possible.

In both approaches priority is given to variables with a higher density in the original problem (present in more monomials), which showed to be a good strategy in some preliminary experiments. Importantly, we create as many new constraints as constraints we had in the original problem. Alternatively, it would be worth studying more aggressive strategies under which multiple different combinations of bounding factors or variables are considered for each constraint.

Note that the new constraints we are adding are valid cuts so the algorithm converges due to Theorem 1.8.

Data Set		Baseline	More in common	Less in common
DS	solved (130/180)	127	130	130
	gap = ∞ (0/180)	0	0	0
	geom. time (68)	77.29	−5.66%	−6.75%
	geom. gap (53)	0.01	−8.84%	−10.01%
	geom. nodes (127)	137.02	−2.90%	−3.26%
MINLPLib	solved (100/168)	99	99	99
	gap = ∞ (6/168)	6	6	6
	geom. time (21)	46.17	+1.70%	−8.67%
	geom. gap (64)	0.49	+2.42%	+7.10%
	geom. nodes (98)	102.41	−0.42%	−1.90%

Table 2.4: Impact of products of constraint factors and bounding factors.

Table 2.4 shows that the "More in common" approach performs slightly better in DS and slightly worse in MINLPLib. Interestingly, the "Less in common" approach seems to lead to slight improvements in both data sets, so it may be worth to study its impact when combined with other enhancements.

2.5.4 Branching Rule

In the baseline configuration, we follow the approach in Sherali and Tuncbilek (1992) and choose the branching variable involved in a maximal violation of RLT identities. More precisely, given a solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ of the linear relaxation, we branch on a variable $i \in \arg\max_{j \in N} \theta_j$, where θ_j is defined as:

$$\theta_j = \max_{J \subset N^\delta, 1 \leq |J| \leq \delta - 1} \{|\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J|\}. \tag{2.1}$$

In a more recent paper, Dalkiran and Sherali (2016) apply a slightly more sophisticated rule for variable selection, in which the maximums in the above equation are replaced by

sums and also the violations associated to each variable are weighted by the minimum distance of the current value of the variable to its lower and upper bound. Here we follow a similar approach and study different rules, where θ_j is of the form:

$$\theta_j = \sum_{J \subset N^\delta, 1 \leq |J| \leq \delta - 1} w(j, J) \, |\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J|, \tag{2.2}$$

where the sums might be replaced by maximums as in the original approach and $w(j, J)$ represent weights that may depend on the variable and monomial at hand. We have studied a wide variety of selections for these weights and the ones that have delivered the best results are the following ones:

Constant weights. $w(j, J) = 1$ for each j and J. This corresponds with the baseline configuration when maximums are taken in Equation (2.2), thus recovering Equation (2.1). Otherwise, if sums are considered we get the rule named "Sum" in Table 2.5.

Coefficients. $w(j, J) \equiv w(J)$ corresponds with the sum of the absolute values of the coefficients of monomial J in the problem.

Dual values. $w(j, J)$ corresponds with the sum of the absolute values of the dual values associated with the constraints of the problem containing J.

Variable range. Defining the range of a variable as the difference between its upper and lower bounds, $w(j, J) \equiv w(j)$ is taken as the quotient between the range of the variable at the current node and its range at the root node. Thus, variables whose range has been reduced less are given a higher priority.

Variable density. $w(j, J) \equiv w(j)$ is taken to be proportional to the total number of monomials in which variable j appears in the problem, so that more "active" variables are given a higher priority.

It is important to highlight that weights $w(j, J)$ are always greater than zero, except in two cases. The first one is if a variable x_j has a lower bound equal to its upper bound. In that case, for the strategy "Variable range", $w(j, J) = 0$. However, if that is the case, Lemma 1.4 implies that $\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J = 0$ for each $J \subset N^\delta$. The second one is when for a certain J, dual values associated to the constraints of the problem containing J are zero. Therefore, Theorem 1.7 cannot be used to ensure the convergence of the algorithm when this branching rule is used. Another important point is that, although "Dual values" is the only rule in which weights $w(j, J)$ can be zero, we can also have numerical issues

when weights are close to zero. To avoid this, we did the following: if the corresponding branching rule (except "Baseline" and "Sum") indicates that $\theta_j = 0$ (or very close to zero) for each $j \in N$, then we use "Sum" rule to select the branching variable in that iteration. Using this strategy, we can ensure the convergence of the algorithm according to Theorem 1.7 for each of the six branching rules.

Data Set		Baseline	Sum	Coefficients	Dual values	Var. range	Var. density
DS	solved (138/180)	127	135	136	136	137	134
	gap = ∞ (0/180)	0	0	0	0	0	0
	geom. time (82)	71.77	−13.07%	−10.04%	+6.61%	−14.89%	−2.09%
	geom. gap (57)	0.01	−23.80%	−21.88%	−18.96%	−24.62%	−21.20%
	geom. nodes (123)	141.76	−13.19%	−9.41%	−0.70%	−12.24%	−1.37%
MINLPLib	solved (108/168)	99	105	105	104	104	104
	gap = ∞ (5/168)	6	5	6	6	6	5
	geom. time (31)	63.04	−50.23%	−46.09%	−52.45%	−69.41%	−49.26%
	geom. gap (66)	0.55	−54.24%	−43.47%	−25.79%	−52.54%	−50.72%
	geom. nodes (97)	95.79	−29.18%	−28.95%	−34.42%	−39.53%	−27.84%

Table 2.5: Impact of branching rules.

Table 2.5 contains the results for the different rules we have just described. Except for the baseline, which uses the maximum as in Equation (2.1), all other rules use sums as in Equation (2.2); the reason being that rules based on sums have shown to be remarkably superior to their "maximum" counterparts. In particular, we can see in the first two columns of the table that, just by replacing the maximum with the sum in the original branching rule, the geometric means of the computing time and gap get divided by two in MINLPLib. In general, all rules based on sums perform notably better than the original one. Arguably "Sum" and "Var. range" are the two most competitive ones and, by looking at the performance profiles in Figure 2.10 in Appendix 2.A, it seems that "Var. range" is slightly superior, which goes along the lines of the approach taken in Dalkiran and Sherali (2016).[10]

2.5.5 Bound Tightening

We study the effect of different bound tightening strategies. More precisely, Table 2.6 represents the following approaches, along with some natural combinations of them:

OBBT root node. OBBT is run on the linear relaxation at the root node. OBBT is quite time consuming, so we limit its available time so that it does not use more than

[10] In Dalkiran and Sherali (2016) their weights were of the form $w(j, J) = \min\{\bar{u}_j - \bar{x}_j, \bar{x}_j - \bar{l}_j\}$, which we tried as well, but they were outperformed by the ones shown in Table 2.5. Further, Dalkiran and Sherali (2016) also included addends of the form $\left|\prod_{k\in J\cup\{j\}} \bar{x}_k - \bar{x}_j \bar{X}_J\right|$. In our experiments we observed no benefit from the inclusion of these terms.

20% of the total time available to `RAPOSa`. Since this sometimes implies that bound tightening is not applied to all variables, we prioritize tightening the upper bounds and also prioritize variables with larger ranges.

Linear FBBT. FBBT is run at all nodes on the linearized constraints of the original problem.[11]

Nonlinear FBBT. FBBT is run at all nodes on the original nonlinear constraints.

Note that bound tightening techniques do not compromise convergence of the algorithm, since they only tight variable bounds, without deleting any point from the feasible region, that is, they are valid cuts. Thus, Theorem 1.8 can be used in this case.

Data Set		Baseline	OBBT root node	Lin FBBT all nodes	Nonlin FBBT all nodes	OBBT + Lin FBBT	OBBT + Nonlin FBBT
DS	solved (133/180)	127	126	126	127	127	124
	gap = ∞ (0/180)	0	0	0	0	0	0
	geom. time (74)	74.99	−2.42%	−9.65%	−11.62%	−6.62%	−10.82%
	geom. gap (67)	0.01	+3.43%	+13.83%	+12.92%	+10.22%	+10.10%
	geom. nodes (113)	143.65	−27.20%	−13.46%	−14.54%	−33.16%	−33.97%
	BT geom. time (180)	0.00	1.19	1.53	0.97	3.27	2.64
MINLPLib	solved (110/168)	99	97	106	109	102	108
	gap = ∞ (5/168)	6	5	5	5	5	5
	geom. time (33)	69.81	−28.74%	−74.71%	−86.39%	−77.86%	−89.59%
	geom. gap (67)	0.50	−33.71%	−66.87%	−79.73%	−68.63%	−81.17%
	geom. nodes (96)	117.66	−48.57%	−39.47%	−55.74%	−59.57%	−73.72%
	BT geom. time (168)	0.00	0.38	0.33	0.26	0.77	0.62

Table 2.6: Impact of bound tightening.

The results in Table 2.6 show a mild improvement on DS and a very large impact on MINLPLib. The fact that bound tightening has a relatively small impact on DS was expected, since the generation procedure for the random instances in this data set already leads to relatively tight bounds. We can see that the nonlinear FBBT is superior to both the linear FBBT and the OBBT. Overall, the best configuration is the one combining OBBT at the root node with nonlinear FBBT at all nodes, which is a standard approach in well-established global solvers. We also ran some tests combining OBBT with the execution of both FBBT schemes at all nodes but, while this led to a reduction on the number of explored nodes, this reduction did not compensate for the additional computational overhead of running two FBBT schemes at every node. Additionally, we also checked if it

[11] Our `C++` FBBT implementation builds upon the following Python implementation `https://github.com/Pyomo/pyomo/tree/main/pyomo/contrib/fbbt`, which is part of Pyomo's environment (Hart et al., 2011). We also run some computational experiments using the bound tightening procedures included in `Couenne`, but the results were slightly worse.

could be beneficial to run FBBT approaches only at prespecified depths of the branch-and-bound tree, such as running it at nodes whose depth is a multiple of 10, but we observed a detrimental effect on performance with these approaches.

2.5.6 SDP Cuts

The last enhancement we study is the introduction of SDP cuts to tighten the linear relaxations. As discussed in Section 2.2.1, the main choice of this approach is the vector $\boldsymbol{y}$ that is then used to define matrix $\boldsymbol{M} = [\boldsymbol{y}\boldsymbol{y}^{\mathsf{T}}]$. Importantly, the resulting cuts involve products of the different components of $\boldsymbol{y}$ and, since we are using J-sets, we should carefully define vector $\boldsymbol{y}$ so that the resulting cuts do not lead to the inclusion of new RLT variables (which in turn would require to include additional bound-factor constraints) and increase the solving time of the linear relaxations.

To minimize the impact of the above problem we proceed as follows. Note that J-sets correspond with maximal monomials with respect to set inclusion. Then, given a maximal monomial J, we define a vector $(x_j)_{j\in J}$ composed of the variables included in monomial J. Using these vectors ensures that the set of maximal monomials does not increase significantly after introducing SDP cuts.[12]. We are now ready to fully describe the different approaches we have studied regarding SDP cuts, which are partially inspired in the comprehensive study developed in Sherali et al. (2012a), to which the interested reader is referred for a deeper discussion and motivation.

First, for each J-set we consider three possibilities to define vector $\boldsymbol{y}$: i) taking $\boldsymbol{y}^1 = (x_j)_{j\in J}$, ii) expanding it to vector $\boldsymbol{y}^2 = (1, (x_j)_{j\in J})$, and iii) expanding it to a vector of the form $\boldsymbol{y}^3 = (1, (x_j)_{j\in J}, \ldots)$ in which, if possible, products of the variables in J are added while keeping under control the new RLT variables required by these additional products and without getting any element in $\boldsymbol{M}$ with a degree larger than δ.

For each of the above three possible definitions of the $\boldsymbol{y}$ vector, we proceed as follows: i) by default, SDP cuts are applied in all nodes and they are inherited "forever", ii) in order to save computational time in the computation of the $\boldsymbol{\alpha}$ vectors, the corresponding $\bar{\boldsymbol{M}}_L$ matrix is divided in 10×10 overlapping submatrices (each matrix shares its first 5 rows with the preceding one), iii) for each eigenvector with a negative eigenvalue, we add the corresponding cut, and iv) the procedure is repeated for each and every maximal monomial.

[12] If a maximal monomial corresponds with RLT variable X_{123}, when defining matrix $\boldsymbol{M}$ from vector (x_1, x_2, x_3) we get X_{11}, X_{22}, and X_{33} from the diagonal elements of $\boldsymbol{M}_L$. If these RLT variables were not present in the original problem, they will be added in our SDP-cuts approach. Yet, in our analysis we observed no significant impact from adding these very specific monomials.

Note that SDP cuts are valid cuts. Indeed, regardless the strategy followed, for each $\bar{\boldsymbol{x}}$ feasible in ($PP(\Omega)$), the matrix $\bar{\boldsymbol{M}}_L$ is positive semidefinite if $\bar{X}_J = \prod_{j\in J} \bar{x}_j$ for each $J \subset N^\delta$, because $\bar{\boldsymbol{M}}_L$ would be the product of a vector by itself. Therefore, the corresponding SDP cut is verified by any $(\bar{\boldsymbol{x}}, E(\bar{\boldsymbol{x}}))$ with $\bar{\boldsymbol{x}}$ feasible in ($PP(\Omega)$). Thus, we can add SDP cuts to the linear relaxations without compromising the convergence of the algorithm due to Theorem 1.8.

Data Set		Baseline	$\boldsymbol{y}^1$	$\boldsymbol{y}^2$	$\boldsymbol{y}^3$	$\boldsymbol{y}^2$ Inh-1	$\boldsymbol{y}^2$ Inh-2
DS	solved (147/180)	127	123	138	134	133	136
	gap = ∞ (0/180)	0	0	0	0	0	0
	geom. time (116)	35.08	+80.18%	+35.08%	+190.18%	+48.87%	+24.70%
	geom. gap (74)	0.01	−0.71%	−38.93%	−26.71%	−24.91%	−37.43%
	geom. nodes (106)	119.87	+3.30%	−18.65%	−35.28%	−2.98%	−16.64%
	SDP geom. time (180)	0.00	0.33	0.36	3.65	0.56	0.44
MINLPLib	solved (102/168)	99	99	102	102	100	102
	gap = ∞ (6/168)	6	7	7	6	6	6
	geom. time (27)	21.95	−5.70%	−35.03%	−33.40%	−21.98%	−25.33%
	geom. gap (64)	0.50	−0.71%	−17.66%	−29.64%	−26.86%	−26.83%
	geom. nodes (98)	102.09	−10.72%	−18.51%	−19.38%	−15.16%	−16.63%
	SDP geom. time (168)	0.00	0.13	0.17	0.17	0.20	0.19

Table 2.7: Impact of SDP cuts.

Table 2.7 shows the results of approaches $\boldsymbol{y}^1$, $\boldsymbol{y}^2$, and $\boldsymbol{y}^3$. All of them seem to improve the performance of the algorithm, except for the running times in DS, with $\boldsymbol{y}^2$ being the best of the three. On the other hand, vectors $\boldsymbol{y}^2$ and $\boldsymbol{y}^3$ perform very similarly in MINLPLib, the reason being that most problems in this data set are quadratic and these two vectors, by construction, coincide for quadratic (and for cubic) problems.

Given the above results, we carried out some additional experimets with vector $\boldsymbol{y}^2$. For instance, the last two columns in Table 2.7 represent the results when considering that cuts are only inherited to child nodes (Inh-1) and that they are also inherited to grandchildren (Inh-2). The performance of the latter is comparable with the one with full inheritance, but no significant gain is observed. Additionally, we also studied the impact of running cycles of the form "solve → add cuts → solve → add cuts..." at each node before continuing with the branching, but in our experiments they had a detrimental effect. Similarly, we also tested configurations in which cuts were added only in nodes at prespecified depths of the branch-and-bound tree, such as nodes whose depth is a multiple of 10, but performance also worsened.

2.5.7 Combining Different Enhancements: RAPOSa's Best Configuration(s)

In this section we study the impact of combining all the enhancements discussed so far in a new version of `RAPOSa`. Further, in order to get a clearer impact on the individual impact of each enhancement, we also study the performance of this new version of `RAPOSa` when the different enhancements are dropped one by one. We believe this analysis is a good complement to the one developed in the preceding sections, where the impact of each enhancement was individually assessed with respect to the baseline version (the one incorporating just the J-sets and the auxiliary nonlinear solver). This new version of `RAPOSa` is defined by taking the best configuration for each individual enhancement. The J-sets, the auxiliary local NLP solver, the warm generation of J-sets, and the warm start on the LP solver are all incorporated. Regarding the other four enhancements, we proceed as follows:

Products of constraint factors and bounding factors. We take the "less in common" approach.

Branching rule. We consider the rule based on variable ranges.

Bound tightening. We consider OBBT at the root node and nonlinear FBBT at all nodes.

SDP cuts. We consider the approach with vector $\boldsymbol{y}^2$, with cuts being generated in all nodes and inherited forever.

Since the number of enhancements is relatively large, we have split the results in Table 2.8 into two blocks, always taking the configuration with all the enhancements, named "All", as the reference one. Further, we have 8 additional columns, each of them corresponding to the results obtained when an individual enhancement is dropped. The results seem to confirm the findings in the previous sections. Both the use of J-sets and of an auxiliary nonlinear solver have a dramatic impact in the performance of the RLT-based algorithm, regardless of the data set. The next most important enhancement is bound tightening, especially in MINLPLib. Also, the impact of the branching rule is quite noticeable and, again, more significant in MINLPLib. The technical enhancement about efficient coding, the warm generation of J-sets, also has a clearly positive impact on performance on both data sets.

The impact of the remaining three enhancements is somewhat mixed. Warm starting the linear solver seems to be beneficial in MINLPLib, but not in DS. The situation gets

Data Set		All	No J-sets	No nonlinear solver	No warm gen. J-sets	No warm start
DS	solved (154/180)	133	47	99	133	151
	gap $= \infty$ (0/180)	0	0	81	0	0
	geom. time (147)	13.847	+2482.72%	+135.22%	+9.51%	−17.4%
	geom. gap (138)	0.002	+724.71%	$>$ +1e5%	+5.53%	−40.1%
	geom. nodes (42)	13.550	+120.17%	+328.89%	0.00%	0.00%
MINLPLib	solved (121/161)	112	93	108	110	111
	gap $= \infty$ (2/161)	2	6	53	2	2
	geom. time (52)	6.824	+1521.31%	+55.63%	+12.86%	+4.24%
	geom. gap (73)	0.031	+2341.28%	$>$ +1e5%	+23.75%	+13.02%
	geom. nodes (86)	21.990	+56.13%	+164.28%	0.00%	+0.77%

Data Set		All	No products	No branching rule	No bound tightening	No SDP cuts
DS	solved (154/180)	133	130	131	131	130
	gap $= \infty$ (0/180)	0	0	0	0	0
	geom. time (147)	13.847	+3.48%	+9.00%	+9.80%	−27.69%
	geom. gap (138)	0.002	+1.92%	+5.73%	+6.61%	−6.98%
	geom. nodes (42)	13.550	0.00%	+6.16%	+59.21%	+1.06%
MINLPLib	solved (121/161)	112	113	110	103	116
	gap $= \infty$ (2/161)	2	2	3	3	2
	geom. time (52)	6.824	−5.47%	+12.86%	+205.92%	−34.82%
	geom. gap (73)	0.031	−1.05%	+54.45%	+327.02%	−32.71%
	geom. nodes (86)	21.990	+0.57%	+17.33%	+140.53%	−13.83%

Table 2.8: Individual impact of the different enhancements.

reversed for the products of constraint factors and bounding factors, which slightly improve performance in DS but slightly reduce it in MINLPLib. Finally, we have that SDP cuts, when added on top of the other enhancements, seem to deteriorate performance. This is somewhat surprising and is definitely a direction for further research. Given the promising behaviour of SDP cuts in Table 2.7, it would be important to understand why they have a substantial negative impact when combined with the other enhancements.

In Figure 2.2 we represent the performance profiles associated to all the configurations we have just discussed, with the exception of the configurations without J-sets and without nonlinear solver since, given their particularly bad performance, would distort the resulting plots. The performance profiles confirm what we have already seen in Table 2.8. The version without warm start of the LP solver is the best one in terms of optimality gap of the difficult instances in DS, whereas the version without SDP cuts is the best one in the other three performance profiles. The superior performance of this version is especially good in MINLPLib. The performance profiles also show that, setting aside the enhancements involving J-sets and the auxiliary nonlinear solver, the highest impact comes from the bound tightening, especially in MINLPLib, where the configuration without this enhancement falls clearly behind the rest.

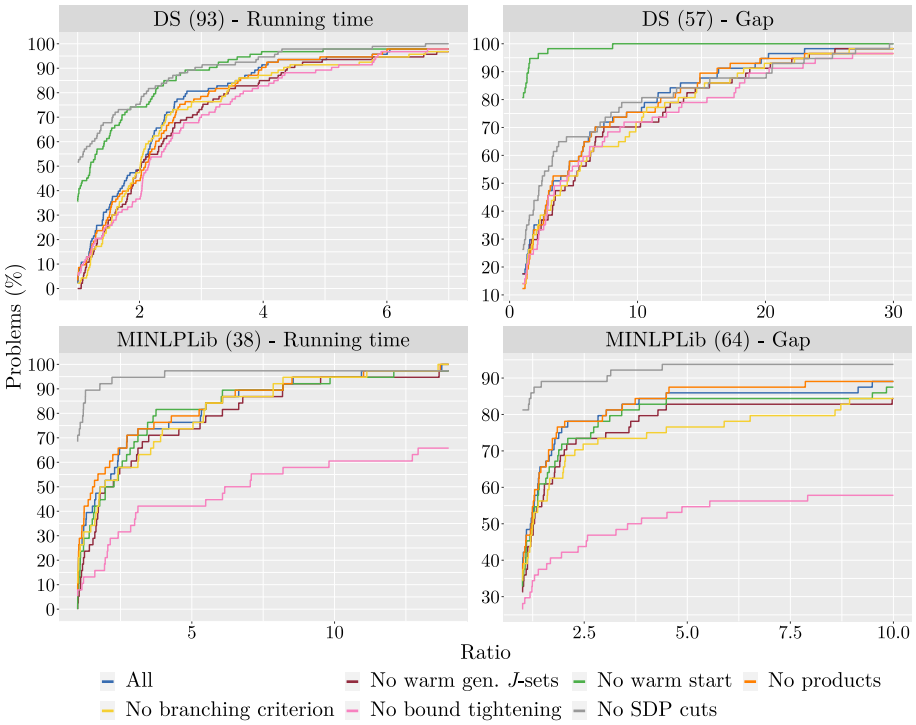


Figure 2.2: Performance profiles of the impact of the different enhancements.

2.6 Overall Performance of RAPOSa and Comparison with Other Solvers

In this section we present a comparison between `RAPOSa`, `BARON`, `Couenne`, and `SCIP` on the instances of DS and MINLPLib. Ideally, we would have liked to include `RLT-POS` (Dalkiran and Sherali, 2016) in the analysis, since its underlying RLT implementation has been one of the main sources of inspiration for `RAPOSa`. Unfortunately, `RLT-POS` is not publicly available. Instead, we present in Table 2.9 a comparison of the enhancements included in both `C++` implementations of the RLT scheme, which shows that none of them dominates the other in terms of enhancements.

Enhancement → ↓ RLT version	J-sets	NLP Solver	Warm Gen. J-sets	Warm Start	Products Cons.&B. Factors	Branch Crit.	Bound Tight	SDP Cuts	Reduced RLT
`RLT-POS`	YES	YES	NO	NO	NO	YES	NO	YES	YES
`RAPOSa`	YES	YES	YES	YES	YES	YES	YES	YES	NO

Table 2.9: Comparison of the functionalities of `RLT-POS` and `RAPOSa`.

The "Reduced RLT" enhancement consists of a series of RLT-based reformulations introduced in Sherali et al. (2012b) for polynomial optimization problems containing linear

equality constraints. The idea is to rely on the basis of the associated linear systems to reduce the size of the linear relaxations. Different variants of this approach are implemented in `RLT-POS`. One additional difference between `RLT-POS` and `RAPOSa` is that the former contains a heuristic which, depending on some underlying features of the problem at hand such as the degree, the density, and the number of equality constraints, chooses a specific configuration of the different features. The computational study in Dalkiran and Sherali (2016) compares `RLT-POS` with `BARON`, `Couenne`, and also with the SDP based solver SparsePOP (Waki et al., 2008). The latter turned out to be the least competitive, whereas the performance of `RLT-POS` was notably superior to `Couenne` and slightly superior to `BARON` in DS. These results are comparable to the ones we report below for `RAPOSa`. As far as the computational study is concerned, one of the additions of the current paper is that the analysis is also developed for MINLPLib, which contains a wide variety of instances coming from real applications, and not just randomly generated instances as DS.[13]

We move now to the comparison, for the instances in both DS and MINLPLib, of `RAPOSa` with three of the most popular solvers for finding global optima of nonlinear programming problems: `BARON`, `Couenne`, and `SCIP`. All solvers have been run taking as stopping criterion that the relative or absolute gap is below the threshold 0.001 and with a time limit of 1 hour in each instance. `RAPOSa` was run with two different configurations: i) the baseline version in Section 2.5 (with J-sets and nonlinear solver) and ii) the version that looked superior from the analysis in Section 2.5.7 (all the enhancements except the use of SDP cuts). It is worth mentioning the auxiliary solvers used by each of the global solvers involved in the comparison. All solvers use `Ipopt` as the auxiliary nonlinear solver. Regarding the linear solver, `RAPOSa` uses `Gurobi`, whereas `BARON`, `Couenne`, and `SCIP` use `Clp`. Given the superior performance of `Gurobi` with respect to `Clp` reported in Section 2.4.2, it may be that the linear solver is giving a slight edge to `RAPOSa`. Yet, this is not straightforward to assess, since the solution of linear subproblems is not equally critical for each solver, since `BARON` and `SCIP`, for instance, heavily rely on nonlinear (convex) relaxations whereas all relaxations solved by `RAPOSa` and `Couenne` are indeed linear.

Table 2.10 contains two summaries of results, one for each data set. First, we can see that, in DS, both configurations of `RAPOSa` solved more problems than the other solvers. For this data set, the version of `RAPOSa` with no SDP cuts clearly dominates all others, not only in the number of solved problems, but also in running times and optimality gaps. At the other end, we see that `SCIP` falls clearly behind `BARON` and `Couenne` in DS. Regarding MINLPLib, `SCIP` is the solver that performs best. The behaviour of the best version

[13] With respect to Dalkiran and Sherali (2016), also the thorough analysis enhancement by enhancement in Section 2.4 and Section 2.5 represents a novel contribution.

Data Set		No SDP cuts	BARON 21.1.13	Couenne 0.5.7	SCIP 7.0.2	Baseline
DS	solved (166/180)	156	145	135	104	152
	geom. time (117)	68.664	+25.24%	+54.09%	+704.90%	+116.52%
	geom. gap (82)	0.002	+423.22%	+473.36%	+20444.16%	+22.21%
MINLPLib	solved (130/168)	119	109	117	126	102
	geom. time (50)	25.435	−66.66%	−62.84%	−56.69%	+1764.75%
	geom. gap (85)	0.035	+551.46%	+44.29%	−33.98%	+724.12%

Table 2.10: Comparative between different configurations of RAPOSa and other solvers.

of RAPOSa is comparable to that of BARON and Couenne, falling behind in running times but being superior in optimality gaps. This suggests that RAPOSa may be particularly effective for difficult instances, a hypothesis that the explore more deeply in the performance profiles below. It is worth noting that BARON, Couenne, and SCIP have been tested for years on MINLPLib instances and, thus, some of these solvers' enhancements may have been designed to address weaknesses identified on them. Last, we can also compare the performance of the version of RAPOSa with all enhancements except SDP cuts with the version with just J-sets and the auxiliary nonlinear solver. The aggregate improvement in performance is remarkable, especially in MINLPLib.

Figure 2.3 contains the performance profiles associated to the results reported in Table 2.10, which confirm the previous conclusions. It is worth emphasizing once again the significant gain in performance exhibited by RAPOSa after the inclusion of the enhancements discussed in this paper. This is especially relevant in MINLPLib since, with these changes, RAPOSa goes from being the worst solver to being highly competitive with the state-of-the-art solvers.

The results in Table 2.10 and Figure 2.3 both suggest that the performance of RAPOSa may be particularly good in the most difficult instances of both data sets, since it is more competitive on the optimality gaps in instances not solved by all instances than on the running times solved by all instances. Indeed, for DS, even the baseline version of RAPOSa dominates BARON, Couenne, and SCIP in terms of optimality gap. To further explore this insight, in Figure 2.4 we represent the performance profiles associated to the instances that no solver managed to solve within the time limit of one hour. They seem to confirm that RAPOSa becomes more and more competitive as the difficulty of the instances to be solved increases (at least in the two data sets under study).

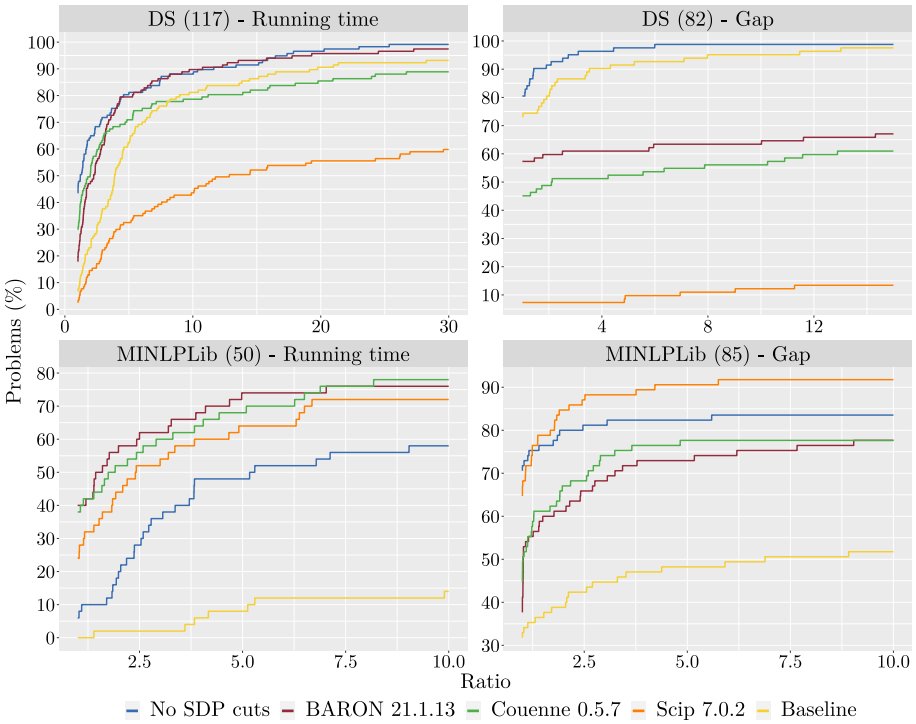


Figure 2.3: Performance profiles to compare `RAPOSa` with other solvers.

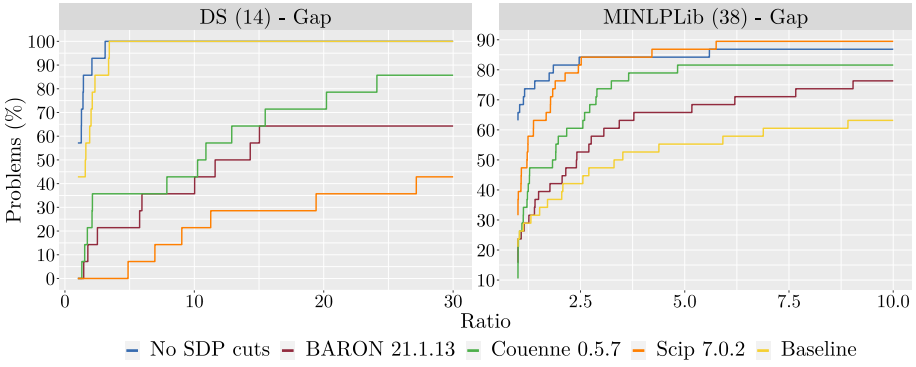


Figure 2.4: Performance profiles to compare `RAPOSa` with other solvers in the most difficult instances (no solver solved them in one hour).

2.7 Conclusions

In this chapter we have introduced `RAPOSa`, a new global optimization solver specifically designed for polynomial programming problems with box-constrained variables. We have thoroughly analysed the impact of different enhancements of the underlying RLT-based algorithm on the performance of the solver. In particular, our findings provide one more piece of evidence of the relevance of bound-tightening techniques to reduce the search space

in branch-and-bound algorithms.

In Section 2.6 we compared the performance of `RAPOSa` with three state-of-the-art solvers and the results are already very promising, since `RAPOSa`'s has proven to be competitive with all of them. Yet, given that `RAPOSa` is still a newborn, it has a great potential for improvement and the analyses in Section 2.4 and Section 2.5 already highlight some promising directions. We conclude with a brief outline of a few of them:

Bound tightening. Given the impact on performance of bound-tightening techniques, it may be interesting to equip `RAPOSa` with more sophisticated version of both FBBT and OBBT routines, such as the ones described in Belotti (2013) and Gleixner et al. (2017). Chapter 5 explores bound tightening in more detail.

SDP cuts and general cut management. As we already mentioned in Section 2.5.7, a clear direction for future research is to get deeper into the analysis of SDP cuts. Given their promising behaviour in Table 2.7, it would be important to understand why it has a negative impact, in MINLPLib, when combined with the other enhancements. Further, implementing a general and flexible cut management scheme might help to improve the overall performance of the RLT-based algorithm and might also be helpful to correct the undesirable behaviour of the SDP cuts. Chapter 4 goes beyond SDP cuts and deals directly with SDP and SOCP constraints.

Learning to branch. There is an intensive research on the use of machine learning techniques to improve the performance of branch-and-bound algorithms, especially in integer programming (Khalil et al., 2016; Lodi and Zarpellon, 2017; Balcan et al., 2018). The integration of the insights from this research into `RAPOSa`, along with some ideas that may be specific to polynomial optimization, is definitely worth pursuing. Chapter 3 illustrates the use of machine learning techniques over a portfolio of branching rules.

Integer problems. A natural avenue for `RAPOSa` is to extend its branch-and-bound scheme to allow for integer programming problems, using the result from Theorem 1.9.

2.A Appendix

In this Appendix we present all performance profiles associated with the computational analysis of the different enhancements of the basic RLT-based algorithm discussed in Section 2.4 and Section 2.5.

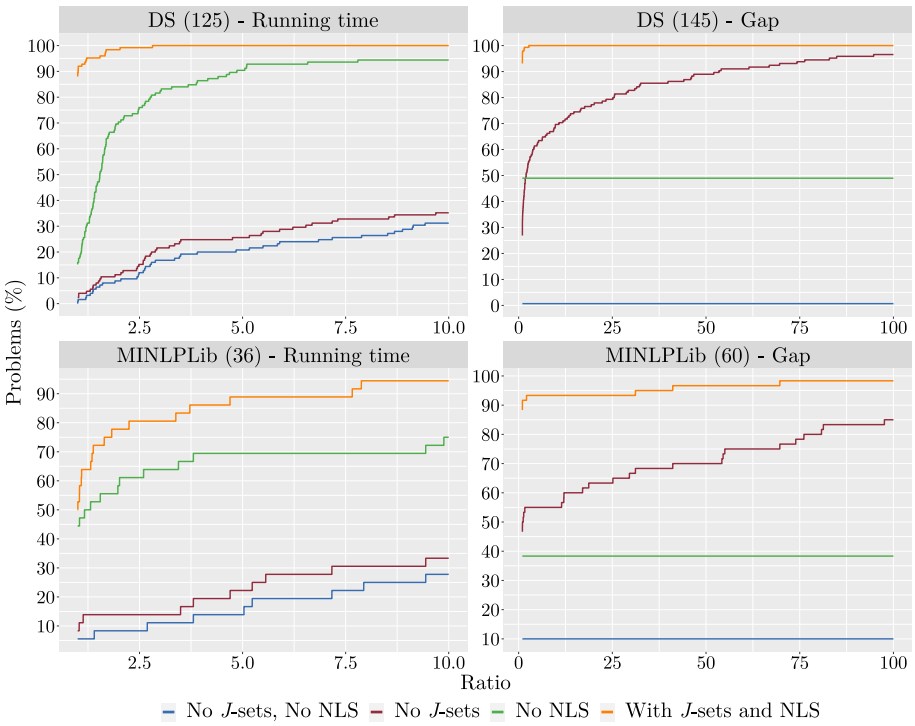


Figure 2.5: Performance profiles of J-sets and auxiliary nonlinear solver.

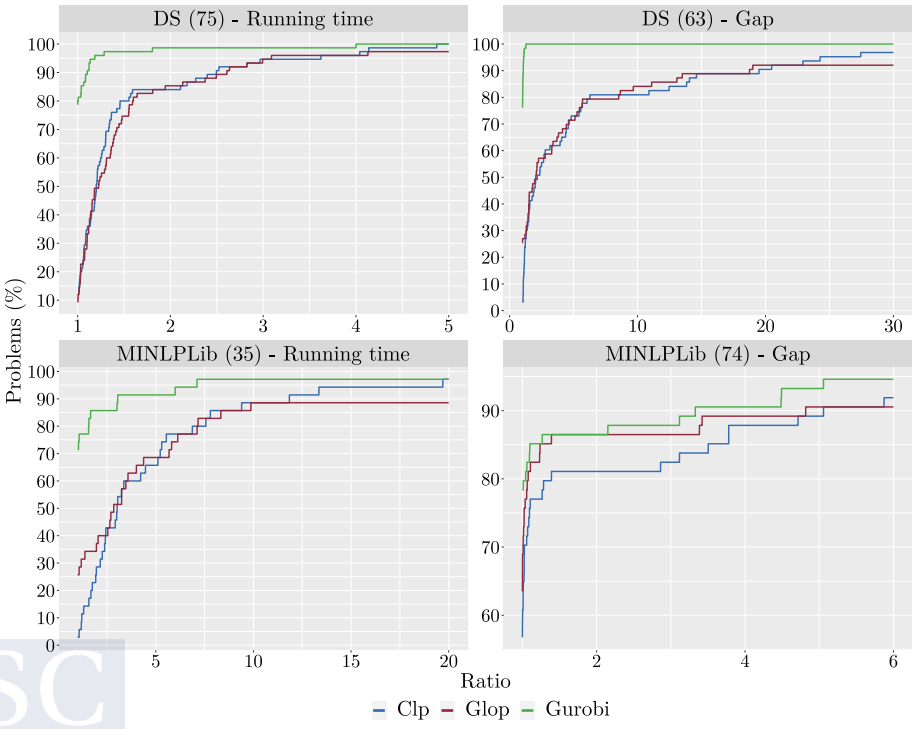


Figure 2.6: Performance profiles of different linear solvers.

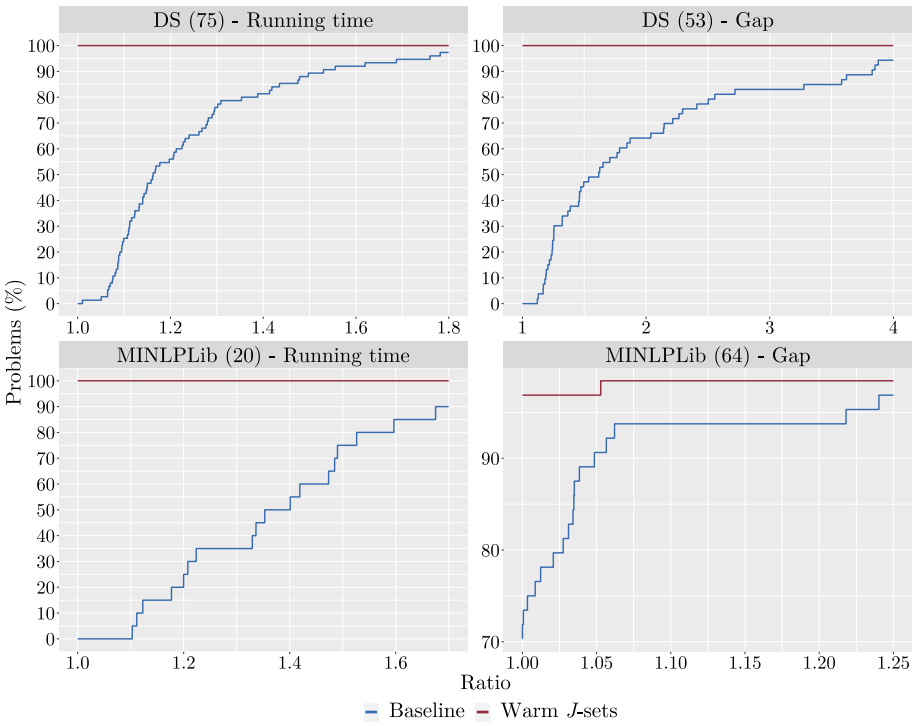


Figure 2.7: Performance profiles of warm generation of J-Sets.

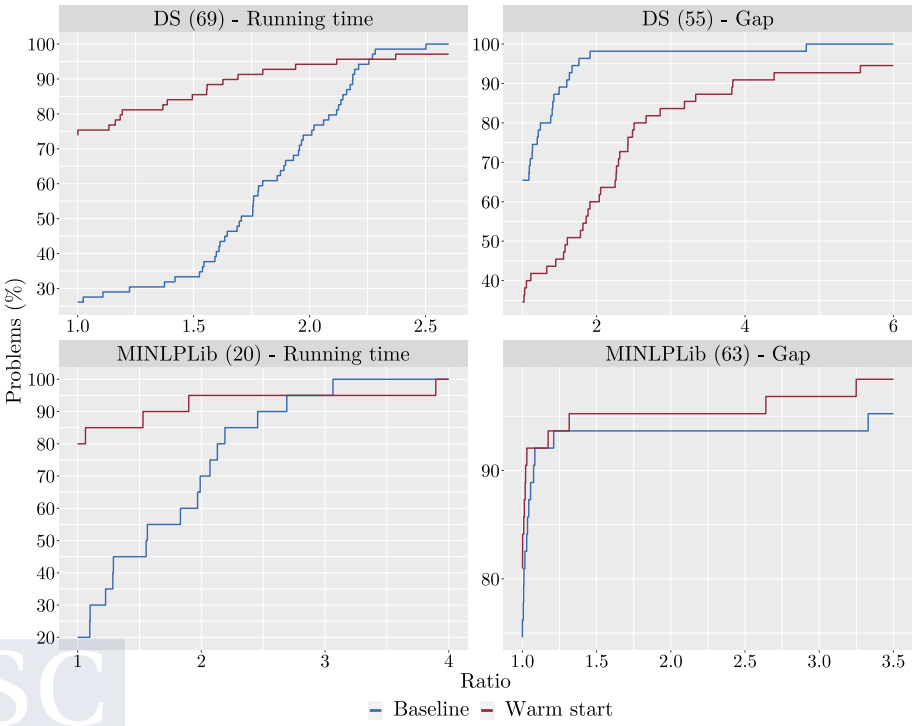


Figure 2.8: Performance profiles of warm start on LP solver.

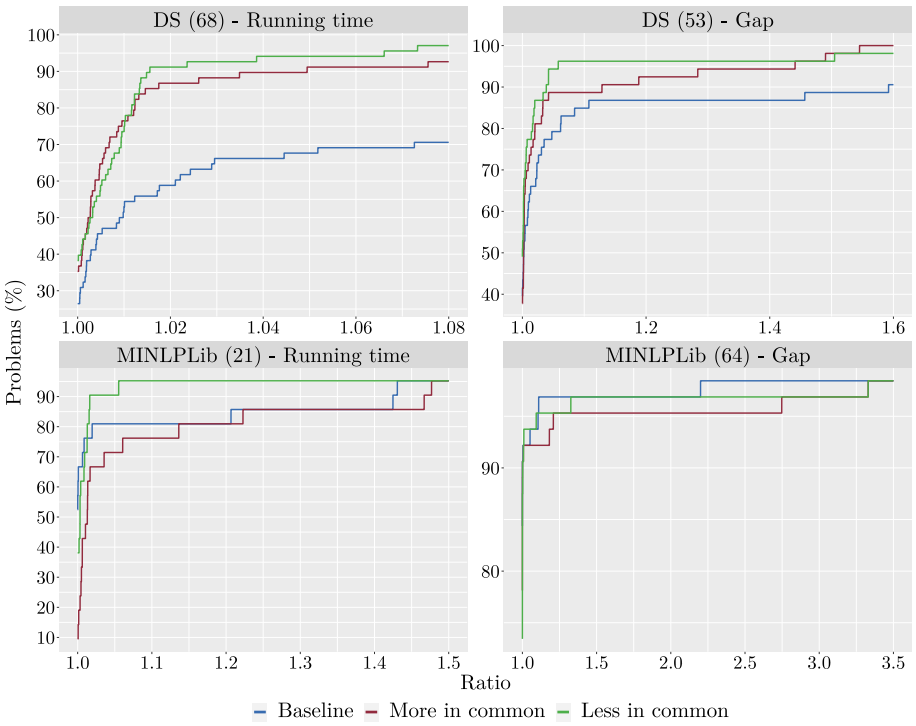


Figure 2.9: Performance profiles of products of constraint factors and bounding factors.

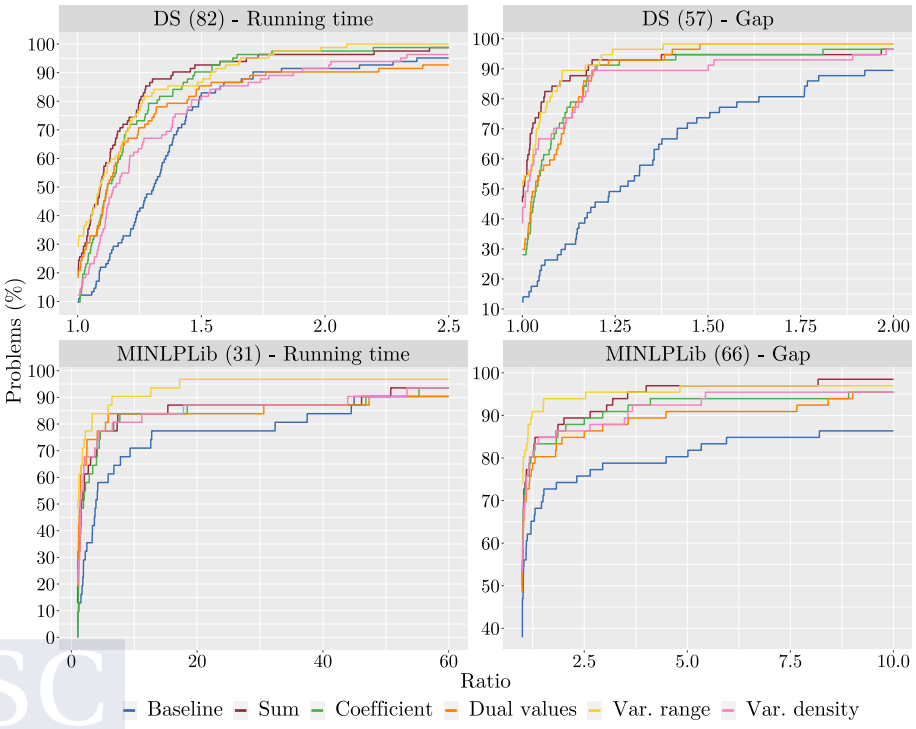


Figure 2.10: Performance profiles of branching rules.

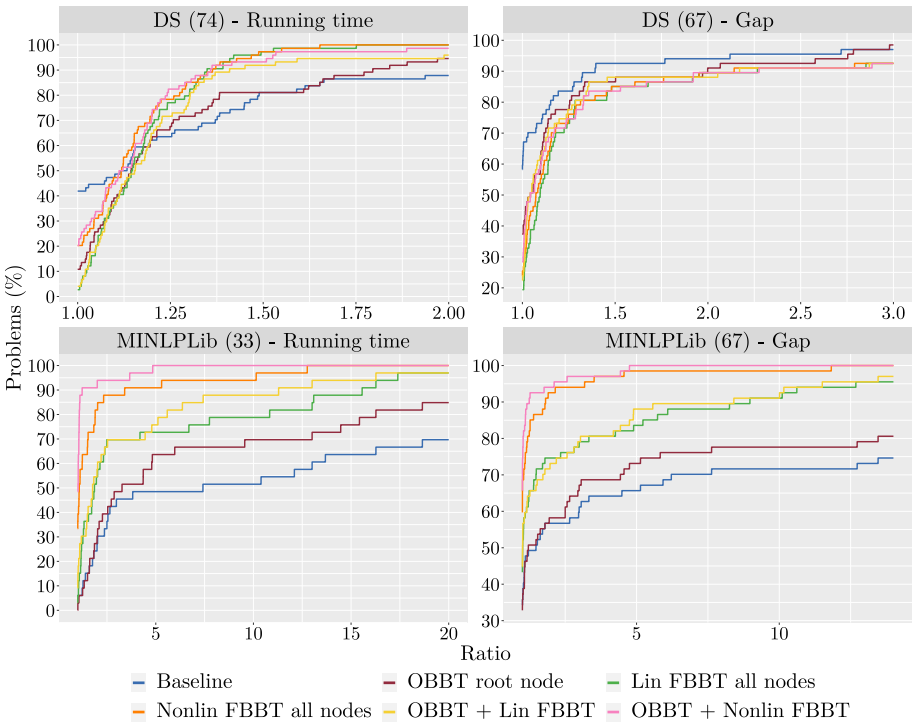


Figure 2.11: Performance profiles of bound tightening.

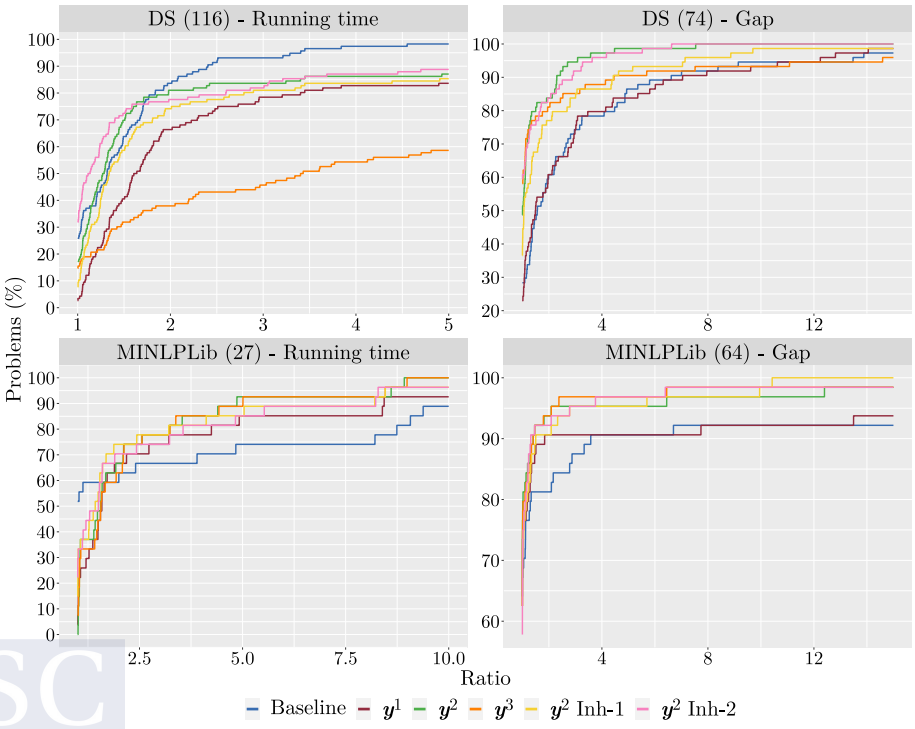


Figure 2.12: Performance profiles of SDP cuts.

Bibliography

Achterberg, T., T. Koch, and A. Martin (2005): "Branching rules revisited," *Operations Research Letters*, 33, 42–54.

Backer, B. D., F. Didier, and E. Guére (2015): "Glop: an open-source linear programming solver," in *22nd International Symposium on Mathematical Programming*.

Balcan, M., T. Dick, T. Sandholm, and E. Vitercik (2018): "Learning to branch," arXiv.

Belotti, P. (2013): "Bound reduction using pairs of linear inequalities," *Journal of Global Optimization*, 56, 787–819.

Belotti, P., S. Cafieri, J. Lee, and L. Liberti (2012): "On feasibility-based bounds tightening," Optimization Online.

Belotti, P., J. Lee, L. Liberti, F. Margot, and A. Wächter (2009): "Branching and bounds tightening techniques for non-convex MINLP," *Optimization Methods and Software*, 24, 597–634.

Bestuzheva, K., M. Besançon, W. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S. J. Maher, F. Matter, E. Mühmer, B. Müller, M. E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, and J. Witzig (2021): "The SCIP optimization suite 8.0," Optimization Online.

Bussieck, M. R., A. S. Drud, and A. Meeraus (2003): "MINLPLib-A collection of test models for mixed-integer nonlinear programming," *INFORMS Journal on Computing*, 15, 114–119.

Byrd, R. H., J. Nocedal, and R. A. Waltz (2006): "Knitro: an integrated package for nonlinear optimization," *Large-Scale Nonlinear Optimization*, 83, 35–59.

Czyzyk, J., M. P. Mesnier, and J. J. Moré (1998): "The NEOS server," *IEEE Journal on Computational Science and Engineering*, 5, 68–75.

DALKIRAN, E. AND H. D. SHERALI (2013): "Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality," *Journal of Global Optimization*, 57, 1147–1172.

——— (2016): "RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems," *Mathematical Programming Computation*, 8, 337–375.

DOLAN, E. D. AND J. J. MORÉ (2002): "Benchmarking optimization software with performance profiles," *Mathematical Programming*, 91, 201–213.

DRUD, A. (1985): "CONOPT: a GRG code for large sparse dynamic nonlinear optimization problems," *Mathematical Programming*, 31, 153–191.

FORREST, J., S. VIGERSKE, T. RALPHS, L. HAFER, H. G. SANTOS, M. SALTZMAN, B. KRISTJANSSON, A. KING, AND S. BRITO (2020): *Clp: version 1.17.6*.

FOURER, R., D. M. GAY, AND B. W. KERNIGHAN (1990): "A modelling language for mathematical programming," *Management Science*, 36, 519–554.

GAY, D. M. (1997): "Hooking your solver to AMPL," Bell Laboratories.

——— (2005): "Writing. nl files," Sandia National Laboratories.

GLEIXNER, A. M., T. BERTHOLD, B. MÜLLER, AND S. WELTGE (2017): "Three enhancements for optimization-based bound tightening," *Journal of Global Optimization*, 67, 731–757.

GONZÁLEZ-RODRÍGUEZ, B., J. OSSORIO-CASTILLO, J. GONZÁLEZ-DÍAZ, Á. M. GONZÁLEZ-RUEDA, D. R. PENAS, AND D. RODRÍGUEZ-MARTÍNEZ (2022): "Computational advances in polynomial optimization: RAPOSa, a freely available global solver," *Journal of Global Optimization*, In press.

GOULD, N. AND J. SCOTT (2016): "A note on performance profiles for benchmarking software," *ACM Transactions on Mathematical Software*, 43, 1–5.

GUROBI OPTIMIZATION, LLC (2022): *Gurobi optimizer reference manual*.

HART, W. E., J. WATSON, AND D. L. WOODRUFF (2011): "Pyomo: modelling and solving mathematical programs in Python," *Mathematical Programming Computation*, 3, 219–260.

Khalil, E. B., P. Le Bodic, L. Song, G. L. Nemhauser, and B. Dilkina (2016): "Learning to branch in mixed-integer programming," *Proceedings of the AAAI Conference on Artificial Intelligence*, 30.

Lodi, A. and G. Zarpellon (2017): "On learning and branching: a survey," *TOP*, 25, 207–236.

Morrison, D. R., S. H. Jacobson, J. J. Sauppe, and E. C. Sewell (2016): "Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning," *Discrete Optimization*, 19, 79–102.

Murtagh, B. A. and M. A. Saunders (1978): "Large-scale linearly constrained optimization," *Mathematical Programming*, 14, 41–72.

Perron, L. and V. Furnon (2022): *OR-Tools 9.3*.

Puranik, Y. and N. V. Sahinidis (2017): "Domain reduction techniques for global NLP and MINLP optimization," *Constraints*, 22, 338–376.

Sahinidis, N. V. (2021): *BARON 21.1.13: global optimization of mixed-integer nonlinear programs*.

Sherali, H. D., E. Dalkiran, and J. Desai (2012a): "Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts," *Computational Optimization and Applications*, 52, 483–506.

Sherali, H. D., E. Dalkiran, and L. Liberti (2012b): "Reduced RLT representations for nonconvex polynomial programming problems," *Journal of Global Optimization*, 52, 447–469.

Sherali, H. D. and C. H. Tuncbilek (1992): "A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique," *Journal of Global Optimization*, 2, 101–112.

Tawarmalani, M. and N. V. Sahinidis (2005): "A polyhedral branch-and-cut approach to global optimization," *Mathematical Programming*, 103, 225–249.

Waki, H., S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto (2008): "SparsePOP — a sparse semidefinite programming relaxation of polynomial optimization problems," *ACM Transactions on Mathematical Software*, 35, 1–13.

WÄCHTER, A. AND L. T. BIEGLER (2006): "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 106, 25–57.

Chapter 3

Learning for Spatial Branching

3.1 Introduction

The contents of this chapter are based on Ghaddar et al. (2022). In recent years, there has been a substantial rise on the use of statistical learning techniques to improve the performance of branch-and-bound schemes. Most of the focus so far has been set on mixed-integer linear programming (MILP) problems and, in particular, on learning to choose the branching variable (see Lodi and Zarpellon (2017) and Bengio et al. (2021) and references therein). Since branch-and-bound schemes are also at the core of most global optimization algorithms for solving nonlinear and nonconvex optimization problems, it is natural to wonder to what extent the insights gained for (discrete) MILP problems can be successfully transferred to the (continuous) nonlinear setting. To this end, in this chapter we study variable selection for spatial branching in polynomial programming problems.

A key factor of the efficiency of state-of-the-art global solvers in nonlinear optimization is the design of the branch-and-bound algorithm and, in particular, the branching rules. Inspired by the recent literature on "learning to branch", we explore ways to harness learning-based approaches to improve the performance of the branch-and-bound search for polynomial optimization and, more specifically, we focus on variable selection in spatial branching. Among the various approaches that have been proposed for solving problems in polynomial optimization, here we build upon the branch-and-bound scheme embedded in the RLT-based algorithm introduced by Sherali and Tuncbilek (1992) and explained in detail in Chapter 1. Although we have framed our contribution in the context of the RLT-based algorithm for polynomial optimization, the insights are extensible to any branch-and-bound technique designed to find global optima for any class of NLP problems.

Machine learning (ML) and the neighbouring field of statistical learning have both been effectively leveraged in several areas to develop algorithms that learn from observing the performance of different tasks. While mathematical optimization intrinsically lies at the core of statistical learning methods, recent years have seen a rise of research in the opposite direction: learning and predicting the best optimization approaches in different

settings. In particular, an active area of research concerns the use of ML techniques in the algorithmic design of optimization solvers. A substantial part of the associated literature, to which this chapter belongs, focuses on improving the performance of branch-and-bound schemes.

Because of the wide-ranging applications and the ease of use of state-of-the-art solvers, mixed-integer linear programming represents an excellent testing ground for the aforementioned research. Further, since these problems are solved using branch-and-bound techniques, the richness of the data generated in the process naturally lends itself to learning. As already mentioned, variable selection in branch-and-bound schemes is probably the topic that has attracted the most attention so far and its outstanding importance for the performance of such schemes has long been recognized.

The integration of learning in the branching decisions can involve different paradigms and approaches. Strong branching has proven to be the variable selection rule that leads to smallest trees (Linderoth and Savelsbergh, 1999), but the associated computational costs make its overall performance not competitive with more sophisticated rules such as reliability branching (Achterberg et al., 2005). This motivates the research in Khalil et al. (2016) and Alvarez et al. (2017), where the authors devise learning approaches to approximate strong branching without the associated computational overhead, and test their approaches on the MIPLIB instances. Another natural learning approach related to branching is taken in Di Liberto et al. (2016) where the authors, inspired by the portfolio approaches to algorithm selection (Gomes and Selman, 2001; Leyton-Brown et al., 2003), train a clustering-based classifier to switch between branching rules and again test their approach on MIPLIB instances. Because of the richness of the different learning paradigms, novel approaches in this area are constantly appearing such as the use of neural networks in Gasse et al. (2019). They approximate strong branching by encoding the branching policies into a graph-convolutional neural network which allows exploiting the natural bipartite graph representation of MILP problems, thereby reducing the amount of manual feature engineering. Then, they test their approach on four benchmarks of specific combinatorial optimization problems.

Although some work has been done on using statistical learning for NLP, this area is still not as mature as the one for MILP. For instance, for nonconvex quadratic programming problems with box constraints, Baltean-Lugojan et al. (2019) try to learn effective linear outer-approximations of semidefinite constraints. To generate the corresponding cuts, a neural network is used to select the most promising submatrices without the computational burden of solving semidefinite programming problems. For convex quadratic problems, Bonami et al. (2022) use ML to decide between applying branch and bound on the quadratic

problem and applying it on an equivalent MILP reformulation.

Despite the extensive work done on learning to branch in MILP problems, it is an issue not yet well understood and a very active area of research. The understanding of variable selection is even more limited in spatial branching for NLPs, where there has not been much research in the topic (an exception is Belotti et al. (2009)), much less in the learning context.

Our research aims to bridge this gap by proposing a statistical learning framework for spatial branching. We start by noticing that even relatively similar branching rules exhibit a lot of variability in their relative performance to one another across instances of different data sets. This is not something specific to this setting since, as argued in Di Liberto et al. (2016) and references therein, work in portfolio algorithms has shown that there is often no single approach that performs best on all instances. Thus, it is important to know to what extent one can learn which rule will perform best on a given instance and this is the main objective of this chapter. We follow an algorithm selection approach in the spirit of the one used in Di Liberto et al. (2016) for MILP problems, but our learning takes place in a regression framework instead of a clustering one. In the present work, learning is done offline and the branching rule for a new instance is selected based on its underlying features. Given the promising results we obtain, the proposed approach can be seen as a first step towards more sophisticated (possibly online) statistical learning schemes for variable selection in NLP problems. Given that the learning takes place offline, our approach entails no computational overhead once the solving process starts, so there is no node-efficiency *vs.* time-efficiency trade-off. We develop our analysis for polynomial programming problems by building upon the global solver `RAPOSa` introduced in Chapter 2. We perform our numerical experiments using both sets of instances introduced in Section 2.3.2 and adding a new one: QPLIB instances (Furini et al., 2018), which illustrate that our approach can be successfully applied to general polynomial programming problems without focusing on a particular class of problems or structure.

The contribution of this chapter is threefold: (1) developing a framework for learning to select the most efficient branching rule in nonlinear optimization problems, (2) defining graph-based features and branching rules that significantly contribute to the outcome of the statistical learning process, and (3) introducing a new performance measure which, by combining information related to both running time and optimality gap, allows for a smooth comparison in sets of instances of varying structure and difficulty.

The remainder of this chapter is organized as follows. In Section 3.2, we present the set of branching rules for algorithm selection. In Section 3.3, we describe the learning methodology, motivate and define the novel key performance indicator (KPI), and describe

the features used for learning. In Section 3.4, we show the performance of the learning approach. Finally, conclusions and future research directions are drawn in Section 3.5.

3.2 Motivation and Problem Description

As we have discussed, the main goal of this chapter is to provide a first study of the impact that statistical learning may have in the context of spatial branching. In this section we describe some graphs and graph-related features for polynomial optimization problems, and the branching rules that constitute our portfolio for algorithm selection.

3.2.1 Graphs Associated to Polynomial Optimization Problems

The relevance of graph representations of general optimization problems has long been recognized and, as discussed in Bienstock and Muñoz (2018) and Faenza et al. (2022), polynomial optimization problems are not an exception. Our interest on these graphs is twofold: first, as a means to define new branching rules and, second, to define features that can be taken as input by the statistical learning process.

To each problem of the form $PP(\Omega)$, one can associate different graphs that capture some characteristics of its structure. We introduce two different graphs, both based on the concept of intersection graphs as defined in Fulkerson and Gross (1965). The first one is the *variables intersection graph*, VIG, analogous to the one used in Bienstock and Muñoz (2018). It is built by assigning a vertex to each variable and connecting two vertices if the corresponding variables appear together in some monomial of $PP(\Omega)$. The second one is the *constraints-monomials intersection graph*, CMIG, where we have one vertex for each monomial, one vertex for each constraint, and one for the objective function. We then connect one vertex associated to a monomial with one associated to a constraint (or the objective function) if the monomial appears in the constraint (or objective function).

Once the graphs are defined, we are interested in parameters that summarize some properties of their underlying structure. Below we describe several standard parameters that we use in different parts of our analysis (for further details refer, for instance, to Latora et al. (2017)).

- **Density or edge density**. Given a graph G, the edge density of G is defined as the number of edges in G divided by the number of edges in the complete graph with the same number of nodes.
- **Treewidth**. Given a graph G, the treewidth is the smallest width of any tree-decomposition of G (Robertson and Seymour, 1984). It gives a measure of how close

a graph is to a tree. Trees have treewidth 1.

- **Modularity**. Given a division of a graph G into communities, the modularity of the division is a measure of the strength of that division, (*i.e.*, graphs with high modularity have dense connections between the nodes within communities but sparse connections between nodes in different communities). The modularity of G is the maximum modularity over all the possible divisions into communities of the graph. Since the exact computation of this parameter is a hard computational problem, we use the greedy algorithm in Clauset et al. (2004) to compute an approximate value.

- **Transitivity**. The transitivity of a graph G is the ratio between the number of triangles (subgraphs that are complete graphs of order 3) and the number of triads in G.

- **Eigenvector centrality or eigencentrality**. For each node in a graph, its centrality score is the corresponding value in the first eigenvector of the graph adjacency matrix.

We now move to the definition of different branching rules, two of which heavily rely on the eigencentralities of the nodes associated with the variables of the problem.

3.2.2 Branching Rules

State-of-the art solvers for nonlinear optimization build upon two main themes to define their variable selection rules: i) adaptations of MILP methods such as strong branching and, most notably, reliability branching (Achterberg et al., 2005), as described in Belotti et al. (2009) and ii) contributions of the variables to the violations of nonlinear terms using variants of the so-called violation transfer (Tawarmalani and Sahinidis, 2004; Belotti et al., 2009). In the specific context of the RLT-based algorithm, past literature has mainly focused on violations of the RLT-defining identities. In the seminal paper by Sherali and Tuncbilek (1992), starting from an optimal solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ at a node in the branch-and-bound tree, the violation of each variable x_j, where $j \in N$, in $PP(\Omega)$ is computed according to the following expression:

$$\theta_j = \max_{J \subset N^\delta, 1 \leq |J| \leq \delta - 1} \{|\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J|\}. \tag{3.1}$$

Then, the variable that maximizes this value is chosen for branching. In Chapter 2 new branching rules were introduced, relying on variations of (3.1) where the maxima are replaced with sums. In particular, in Section 2.5.4 we defined various branching rules based

on the formula:

$$\theta_j = \sum_{J \subset N^\delta, 1 \leq |J| \leq \delta - 1} w(j, J) |\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J| \tag{3.2}$$

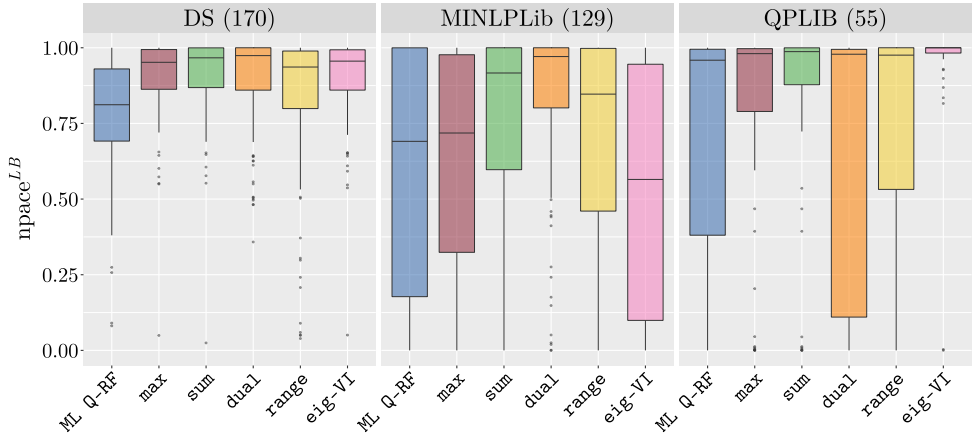
where $w(j, J)$ are the weights associated with the violations. In order to define our portfolio of branching rules we complement the original formulation in (3.1) with various rules from Section 2.5.4 defined by using different weights inside the sum, which can be seen in Table 3.1. Moreover, two new branching rules related to graphs measures are part of our portfolio.

Branching rule	**Definition**
Maximum (`max`)	Eq. (3.1).
Sum (`sum`)	Eq. (3.2), with $w(j, J) = 1$ for each j and J.
Dual (`dual`)	Eq. (3.2), with $w(j, J)$ defined as the sum of the absolute values of the shadow prices of the problem constraints containing $J \cup \{j\}$.[1]
Range (`range`)	Eq. (3.2), with $w(j, J) = \frac{\min\{\bar{u}_j - \bar{x}_j, \bar{x}_j - \bar{l}_j\}}{u_j - l_j}$, where u_j and l_j are the upper and lower bound of x_j at the root node, *i.e.*, $PP(\Omega)$, and $\bar{u}_j$ and $\bar{l}_j$ are the bounds at the current node.
Eigencentrality VIG (`eig-VI`)	Eq. (3.2), with $w(j, J)$ defined as the eigencentrality of x_j's node in VIG.
Eigencentrality CMIG (`eig-CMI`)	Eq. (3.2), with $w(j, J)$ defined as the eigencentrality of x_j's node in CMIG (0 if x_j never appears alone in a monomial).

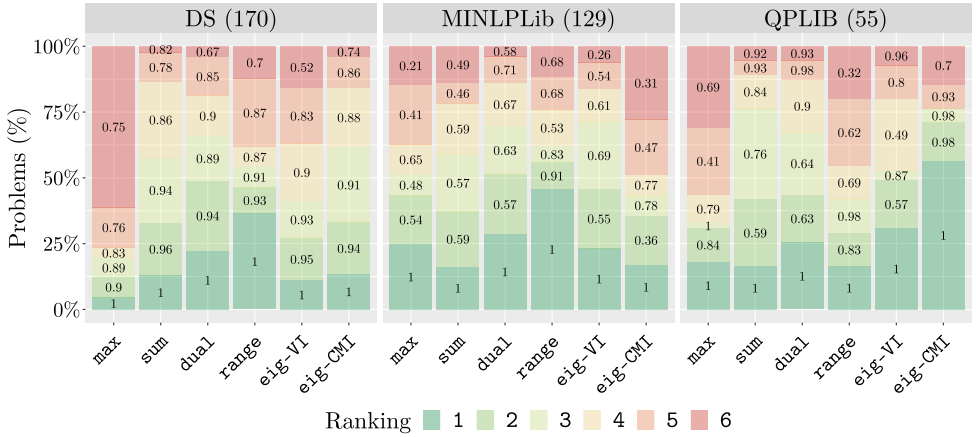
[1] In Section 2.5.4 we considered constraints containing J, instead of $J \cup \{j\}$, but further analysis showed that considering constraints containing $J \cup \{j\}$ throws better results.

Table 3.1: Definition of the branching rules.

Figure 3.1 shows the results of a preliminary computational experiment to assess the performance of the different branching rules. More precisely, we used three different sets of instances (refer to Section 3.4.1 for details): DS (Dalkiran and Sherali, 2016), MINLPLib (Bussieck et al., 2003), and QPLIB (Furini et al., 2018). For our comparison of the branching rules, we use a KPI whose values are normalized to $[0, 1]$ (details are given in Section 3.3.2). The closer the normalized KPI is to 1 for a given branching rule, the better the performance of that rule. Figure 3.1a shows, for the problems in each of the three sets of instances, the boxplots of this normalized KPI using each of the described branching rules. As expected, no branching rule outperforms the rest in the different data sets. While the `sum` rule performs reasonably well in the DS instances, it does not for the ones in MINLPLib. In QPLIB, the `eig-CMI` rule clearly outperforms all other rules, but it would be a poor choice for many of the MINLPLib instances. These observations already hint at the promising potential of the integration of statistical learning techniques in the branching rule selection process. In Figure 3.1b we rank, for each problem, the branching



(a) Boxplots of the normalized KPI for each branching rule.



(b) Each branching rule ranked from 1 (best) to 6 (worst), according to the normalized KPI. Stacked bar graphs represent percentage of problems in each rank position per rule. Numbers inside bars indicate the average value of the normalized KPI for the instances in each rank position.

Figure 3.1: Comparative analysis of the performance of the different branching rules.

rules from 1 (best) to 6 (worst), according to the normalized KPI. As we observe from the learning process, it is important not only to choose the best possible branching rule, but also to ensure that, in case of mistake, the selected rule performs reasonably well. For instance, for those MINLPLib instances where `max` is not the best one, its performance is on average significantly worse than other methods. In those cases, it would be preferable to choose, for instance, the `range` rule, even when it is not the best possible choice.

Finally, it is important to highlight that the two new branching rules based on graph measures could compromise the convergence of the algorithm. Indeed, eigencentrality could be zero for some variables, hence Theorem 1.7 would not ensure convergence of the algorithm. However, we followed the same strategy of Section 2.5.4 in order to guarantee

the convergence of the algorithm.

3.3 Proposed Methodology

3.3.1 Overview of the Learning Framework

Our learning framework is related to the approach for switching heuristics in Di Liberto et al. (2016), where the authors use a cluster-based method to learn to choose from a portfolio of branching rules in MILP problems. In this work, we develop a different, regression-based, statistical learning technique. The main elements of our setting, some of which are novel with respect to past literature, are the following:

- **Spatial branching**. We separate from previous literature by studying branching variable selection in nonlinear problems.
- **Algorithm portfolio**. The goal is learning to choose from the branching rules defined in Section 3.2.2.
- **Data collection**. Learning is performed on a rich set of very diverse instances (see Section 3.4.1). All instances are solved with all branching rules, gathering all the relevant data on their performance. Thus, all the learning takes place offline, with no computational overhead when solving new instances (beyond the computation of the features of the given instance).
- **Graph-based branching rules and features**. A novel part of our approach is the use of graphs to define both branching rules and features (see Tables 3.1 and 3.2, respectively). Remarkably, both of them turn out to play an important role in the results.
- **Statistical learning technique**. It consists of running individual regressions for each branching rule, with the goal of predicting its relative performance with respect to the remaining ones. Then, given a new instance, the rule with the best predicted relative performance is chosen.
- **KPI.** We present a novel KPI, which we refer to as *pace*, and that is discussed in detail in Section 3.3.2. The pace allows to jointly study the performance in instances solved by all methods (where running time is normally used) and instances not solved by at least one method (where optimality gap is normally used).

- **Learning on static features.** Learning uses instance-specific features, with no dynamic information being gathered on a node-by-node basis along the branch-and-bound tree. Thus, the resulting learned branching rule is static, in the sense that it selects the same branching rule in each and every node of the tree associated to a given instance. This is different from most of the literature in MILP problems where dynamic features are often included (see, for instance, Di Liberto et al. (2016), Khalil et al. (2016), and Alvarez et al. (2017)).

It is worth noting that, despite the variability in performance illustrated in Figure 3.1, all the rules in our portfolio revolve around the same baseline computation (violations of RLT-defining identities). This common core probably limits to what extent they can complement each other and, as a consequence, may result in learned rules with less potential to significantly outperform the original ones. Similarly, the choice to perform statistical learning only with static features, may result in learning a less effective branching rule. Because of the two aforementioned points, our analysis provides some sort of lower bound on the "potential for learning" in spatial branching. One of our main contributions is to show that, even under these "limitations", there is much potential to improve the overall performance of the branch-and-bound algorithm by conducting offline learning on the branching rule.

3.3.2 A Novel Key Performance Indicator

Carrying out a fair comparison of the performance of different algorithms on instances of varying difficulty often involves some challenges. Ideally, one would like to have a KPI that allows to compare the performance on all instances together, but the most widely used KPIs, running time and optimality gap, fail to do so. To illustrate this, let's define the following subsets of the set of instances:

- $A =$ "Instances solved by all algorithms within the time limit".
- $B =$ "Instances solved by some but not all algorithms within the time limit".
- $C =$ "Instances not solved by any algorithms within the time limit".

The running time is a good KPI in A and uninformative in C, whereas the optimality gap is a good KPI in C and uninformative in A. Further, neither the time nor the gap allows to differentiate between algorithms for each instance in B, since the running time does not distinguish between algorithms that have reached the time limit (but with different gaps) and the optimality gap does not distinguish between algorithms that have solved

the given instance (but with different running time). Because of this, it is customary to split the performance analysis separating the problems in $A \cup B$ and C or A and $C \cup B$. For instance, in Alvarez et al. (2017) they "separated the problems that were solved by all methods from the problems that were not solved by at least one of the compared methods". An important limitation of such approaches is that the sets A, B, and C depend on the algorithms to be compared, so the results may substantially change if a new algorithm is added to the study or an existing one is removed. In Gupta et al. (2020), when discussing limitations of their work stemming from the $\mathcal{NP}$-hard nature of MILP solving, the authors acknowledge that "one can consider the primal-dual bound gap after a time limit as an evaluation metric for the bigger instances, but this is misaligned with the solving time objective".

We propose a novel approach to define KPIs that allows to jointly compare all instances while being equivalent to the running time in A, mimicking the optimality gap in C, and being able to completely differentiate between successful and time-limiting algorithms in B. Thus, this new family of KPIs overcomes all the issues mentioned above. First, recall the standard definition of the optimality gap at the end of the algorithm, $OG^{\texttt{end}}$, as a function of the lower and upper bounds (LB and UB, respectively):

$$OG^{\texttt{end}} = \frac{UB^{\texttt{end}} - LB^{\texttt{end}}}{|UB^{\texttt{end}}| + \varepsilon},$$

where ε is a small constant needed to avoid divisions by zero. We can now define a KPI based on the pace at which the optimality gap is closed during the execution:

$$\text{pace}^{OG} = \frac{\text{time}}{OG^{\texttt{root}} - OG^{\texttt{end}} + \varepsilon}.$$

Note that, in our setting, $OG^{\texttt{root}}$ is common for each branching rule. So defined, pace^{OG} represents the time needed to improve the optimality gap in one unit. Suppose now that we are comparing two algorithms according to their paces pace_1^{OG} and pace_2^{OG}:

- In set A, $OG^{\texttt{end}}$ is the same for each rule. Hence, $\text{pace}_1^{OG} \,/\, \text{pace}_2^{OG} = \text{time}_1/\text{time}_2$.
- In set C, the running time is the same for each rule. Hence, $\text{pace}_1^{OG} \,/\, \text{pace}_2^{OG}$ is of the form $(OG_2^{\texttt{end}} + K)/(OG_1^{\texttt{end}} + K)$, where K is a constant term.
- In set B, pace^{OG} may recognize the difference between two rules that have solved an instance if they have different running times and, similarly, between two rules that have not solved an instance if they have different gaps. Further, as desired, rules that have solved an instance will have a better pace^{OG} than those that have not.

Given the above properties, pace^{OG} stands out as natural candidate for a streamlined comparison of branching rules, jointly for all instances. Unfortunately, although branch-and-bound schemes always compute a lower bound at the root node, an initial upper bound is often not available. Yet, since the main goal of the branching rules is often to deliver a faster increase of the lower bounds, it is natural to define a variant of pace^{OG} that is based on the pace at which the lower bound increases:

$$\text{pace}^{LB} = \frac{\text{time}}{LB^{\texttt{end}} - LB^{\texttt{root}} + \varepsilon}. \tag{3.3}$$

Note that pace^{LB} is always well defined and measures the amount of time needed to improve the lower bound in one unit. This new KPI, pace^{LB}, guides both the learning process described in Section 3.3.4 and the illustration of the results in Section 3.4.

3.3.3 Features

A key element of our approach is the identification and selection of input variables (features) that are able to explain and predict the behaviour of the aforementioned KPI for each branching rule. Following some of the ideas in Di Liberto et al. (2016), Khalil et al. (2016), and Alvarez et al. (2017) for MILP and our knowledge on polynomial optimization problems, we have considered 34 features. They are summarized in Table 3.2. Features are categorized depending on whether they represent characteristics of the variables, constraints, monomials, coefficients, or other attributes of the polynomial optimization problems. Additionally, we have considered some novel features related to the graph representations of $PP(\Omega)$, VIG and CMIG, as described in Section 3.2.1. Many of the chosen features consist of counts or statistical measures (such as average, median, and variance) of the characteristics of the optimization problem. These are *static* features, as they only depend on the formulation of the original problem. They represent global information of the problem instance, without any node-specific information that changes throughout the branch-and-bound tree.

3.3.4 Quantile Regression Learning Techniques

In supervised learning, data consists of input–output pairs. Given an optimization problem of the form $PP(\Omega)$ we use, as input, the information from the features vector. Regarding the output, we compute for each of the branching rules described in Section 3.2.2 the so-called pace^{LB}, that is, the ratio between the running time and the improvement in the lower bound. It is convenient to normalize the values of pace^{LB} to $[0, 1]$ by dividing the best pace among all rules, *i.e.*, the smallest, by the pace of each rule. We refer to this

Variables	No. of variables, variance of the density of the variables[1] Average/median/variance of the ranges of the variables[2] Average/variance of the no. of appearances of each variable[3] Pct. of variables not present in any monomial with deg. greater than 1 Pct. of variables not present in any monomial with deg. greater than 2
Constraints	No. of constraints, Pct. of equality/linear/quadratic constraints
Monomials	No. of monomials Pct. of linear/quadratic monomials Pct. of linear/quadratic RLT variables Average pct. of monomials in each constraint and in the obj. fun.[4]
Coefficients	Average/variance of the coefficients
Other	Degree and density of $PP(\Omega)$[5] No. of variables divided by no. of constrains/degree No. of RLT variables/monomials divided by no. of constrains
Graphs	Density, modularity, treewidth, and transitivity of VIG and CMIG[6]

[1] The density of a variable is the number of different monomials in which it appears divided by the number of different monomials in the problem.
[2] The range of a variable is the difference between its upper and its lower bound.
[3] For each variable we compute in how many constraints (and objective function) that variable is present.
[4] For each constraint (and objective function) we compute the percentage of monomials present in it with respect to the total number of monomials of $PP(\Omega)$.
[5] The density of $PP(\Omega)$ is its number of different monomials divided by total number of monomials a problem with the same degree and number of variables might have.
[6] We do not include transitivity for CMIG since, given the structure of the graph, its value is always 0.

Table 3.2: List of features used by the different learning techniques.

normalized pace as npace^{LB}. Thus, the closer this value is to one for a given branching rule, the better the performance of that rule. Note that the pace^{LB} of a given branching rule is independent of the other rules, but its npace^{LB} is not.

While our ultimate goal is the selection of the best branching rule, our variables of interest are quantitative outputs. Therefore, we address the problem as a regression problem. The task is to learn, for each rule, a real-valued function that models and predicts npace^{LB}. Then, given an instance, the rule that corresponds to the highest predicted normalized pace is selected.

The primary goal of classical regression models is to estimate the conditional mean of a response variable, $y \in \mathbb{R}$, given the features vector $\boldsymbol{x} \in \mathbb{R}^d$. Let y denote the npace^{LB} of a given rule. In the preliminary computational experiment in Figure 3.1a we observe an asymmetric behaviour of y (negative skewness), as well as the presence of outliers. This makes conventional regression models based on the conditional mean unsuitable. In this context, quantile regression is presented as a more appropriate methodology, since

it does not make any assumptions about the distribution of the response variable and is more robust to the presence of outliers. Quantile regression aims to estimate $Q_\tau(\boldsymbol{x})$, the τ-conditional quantile of y, with $\tau \in (0,1)$. More precisely, $Q_\tau(\boldsymbol{x}) = \inf\{y : \ F(y|\boldsymbol{x}) \geq \tau\}$, with $F(y|\boldsymbol{x})$ being the conditional cumulative distribution function of y. Thus, the focus is on the conditional distribution of the response variable rather than just on its conditional mean, $\mu(\boldsymbol{x}) = E(y|\boldsymbol{x})$. In our context it might be relevant to find out, for example, that for certain instances a given rule exceeds, with high probability, a given value of y. We refer to Koenker (2005) and Koenker et al. (2017) for a detailed overview of quantile regression.

There are several quantile regression methods in the statistics and ML literature. We focus on nonparametric methods, which allow for more flexible specifications of the relation between the response and the explanatory variables than their parametric counterparts. We present below a quick overview of the three main methods used in our analysis.

Quantile Generalized Additive Models.

Generalized additive models assume that the conditional mean of the response has an additive structure such as $\mu(\boldsymbol{x}) = \sum_{j=1}^{m} f_j(\boldsymbol{x})$, where the m additive terms represent predictors. The f_j's are nonparametric functions which can be modelled using an expansion of basis functions; spline basis functions in our case. Hence, we get a flexible and interpretable statistical method, capable of characterizing nonlinear regression effects. This leaves behind the traditional linear model that fails to represent the real effects of the inputs, which are normally nonlinear. For detailed information of generalized additive models, we refer to Hastie et al. (2009). Quantile generalized additive models, see Fasiolo et al. (2021a), extend the previous idea and assume that the τ-conditional quantile of y is given as a linear combination of unknown smooth functions of the features vector. Thus, it uses a very flexible technique that combines the ideas of quantile regression and generalized additive models.

Stochastic Gradient Boosting for Quantile Regression.

Boosting is a supervised learning technique designed for classification problems, but also applicable to regression. The basic idea of this ensemble method is to create a highly accurate prediction rule as a weighted combination of weak learners predictions. In this work we focus on the stochastic gradient boosting algorithm, introduced in Friedman (2002). The stochastic gradient boosting algorithm is a modification of the gradient boosting algorithm by Friedman (2001), that improves the performance by incorporating randomness in the procedure. More specifically, a subsample of the training data is randomly selected at each

iteration, and used to fit the base learner (tree).

We use stochastic gradient boosting for quantile regression, which generalizes the described method to the context of quantile regression by using an asymmetrically weighted absolute loss function (Kriegler and Berk, 2010).

Quantile Regression Forests.

Random forests are supervised learning algorithms that can be used both in classification and regression problems. They were introduced in Breiman (2001) as ensemble methods that grow several individual decision trees and aggregate them to make a single prediction. For regression, they give an approximation of the conditional mean of a response variable. Quantile regression forests, introduced in Meinshausen (2006), are a generalization of random forests that compute an estimation of the conditional distribution of the response variable by taking into account all the observations in every leaf of every tree and not just their average. One of the advantages of random forests, as well as quantile random forests, is that we can use the complete data set, without randomly splitting the data into training and data set, to evaluate the model using the out-of-bag predictions.

3.4 Computational Results

3.4.1 Experimental Setup

We use three different sets of instances to test the performance of the learning approach on a variety of instances with different characteristics. The first and second one are the sets of instances introduced in Section 2.3.2 and the third data set comes from another well-known benchmark, QPLIB[1] (Furini et al., 2018), a library of quadratic programming instances, for which we made a selection analogous to the one made for MINLPLib, resulting in a total of 63 instances. Moreover, we have disregarded instances solved at the root node, since the branching rule plays no role in them. We have also discarded unconstrained problems, as some of the features have the number of constraints in the denominator. Our final data set consists of a total of 354 instances.

All the executions reported in this chapter have been performed on the supercomputer Finisterrae II, at Galicia Supercomputing Centre (CESGA). Specifically, we used computational nodes powered with 2 deca-core Intel Haswell 2680v3 CPUs with 128GB of RAM and 1TB of hard drive.

[1] Instances from QPLIB can be downloaded at https://raposa.usc.es/files/QPLIB-TS.zip.

The branching rules defined in Section 3.2.2 have been coded in `RAPOSa` (González-Rodríguez et al., 2022), the state-of-the-art solver introduced in Chapter 2. The analyses are performed using a configuration of `RAPOSa` with a minimal number of enhancements from Chapter 2 (calls to an NLP local solver and J-sets). Each instance of our final data set was run with six different configurations of `RAPOSa`, based on the six different branching rules defined in Section 3.2.2. The time limit of each execution was set to one hour.

We follow the standard learning procedure. We randomly split the complete set of instances into two disjoint sets: the training set (70%) and the data set (30%). With the objective of obtaining a better performance, we gather the instances into "families" related to the groupings defined in the corresponding libraries, where those belonging to the same group share similar characteristics. The within-family proportion of instances is maintained through the splitting process. Moreover, in order to gain robustness, we construct 10 partitions of the data set into training and test data and report aggregate results over all the partitions.

As discussed in Section 3.3.4, when performing quantile regression, one has to specify the τ-conditional quantile to be estimated. Due to the negative asymmetry present in the KPI (see Figure 3.1), we have opted for values below the median ($\tau \leq 0.5$), where more differentiated behaviours can be observed between the branching rules and where it is more relevant to make a good selection. Although the different values evaluated lead to similar results, the best ones were obtained with $\tau = 0.3$, which is therefore the value used for the numerical analysis.

We have performed the learning process in R programming language (R Core Team, 2022). We have used the libraries "qgam" (Fasiolo et al., 2021b), "gbm" (Greenwell et al., 2020), and "ranger" (Wright and Ziegler, 2017) for quantile generalized additive models, stochastic gradient boosting for quantile regression, and quantile regression forests, respectively.

3.4.2 Numerical Results and Analysis

This section presents the numerical results that allow to evaluate the performance of the obtained ML-based branching rules. The objective is to show the impact of the ML approach when embedded within `RAPOSa` and also to provide further evidence of its effectiveness across various sets of instances. As a first step we compare the three ML-based rules obtained with the learning methodologies described in Section 3.3.4, that is, quantile generalized additive models (Q-GAM), stochastic gradient boosting for quantile regression (SGB-QR), and quantile regression forests (Q-RF). As `RAPOSa` includes several

enhancements (see Chapter 2), we provide results of the ML-based branching for different versions of `RAPOSa`. In this chapter, we consider as the baseline version, the baseline version in Chapter 2 including the enhancements from Section 2.5.1 and Section 2.5.3. Furthermore, we provide results for this baseline version with warm start on LP solver (Section 2.5.2) and with bound tightening (Section 2.5.5).

	Baseline	**WS**	**WS + BT**
Best branching rule (across all instances)	13.741	12.773	5.578
ML-based branching rule:			
Q-GAM	9.097	8.915	5.145
SGB-QR	9.155	8.723	5.024
Q-RF	8.884	8.570	5.035
Optimal rule (instance by instance)	7.498	7.058	4.460
Improvement after learning (with Q-RF)	35.3%	32.9%	9.7%
Optimal improvement (upper bound for learning)	45.4%	44.7%	20.0%

Table 3.3: Performance of ML-based rules according to pace^{LB} (average over data sets).

Table 3.3 presents the results for the pace^{LB},[2] where the reported number corresponds to the average, across the 10 data sets, of the geometric mean of pace^{LB} in the instances of each data set. Similarly to what we did in the numerical analysis in Chapter 2 for time and for gap, in this table and the following ones we have discarded those instances that were solved by all configurations in less than 5 seconds and those instances for which no configuration could return a gap after the time limit.[3] The best branching rule is the original branching rule where the aforementioned average is smaller, that is, the one performing best. For the baseline configuration, the `range` is the best branching rule while for the other two setups, the `dual` is the best branching rule. The optimal rule corresponds with choosing, for each instance, the original rule that performs best. We can see that the three learning approaches Q-GAM, SGB-QR, and Q-RF have similar performance. Since Q-RF performs slightly better, hereafter we use it as the reference ML-based rule. For the baseline configuration, compared to selecting the best branching rule, Q-RF improves the pace by 35.3%. Importantly, this is not far from the improvement obtained by the optimal rule, 45.4%, which is an upper bound for our algorithm selection approach. For the WS + BT, the improvement with Q-RF is 9.7% and the optimal one is 20.0%. This is due to the fact that bound tightening techniques already improve the bounds significantly, reducing the search space and, therefore, reducing as well the room for improvement using different branching rules. Interestingly, the WS enhancement improves the KPI from 13.741 to

[2] We take $\varepsilon = 0.001$ in Equation (3.3).

[3] The resulting number of instances after this filtering is 257.

12.773, which is 7.0%. This value is lower than using Q-RF, which implies that given the choice between applying warm starting or a machine-learning branching approach, the latter may be better (and requiring no computational overhead). Table 3.4 shows that, when using out-of-bag predictions for the Q-RF methodology, we get similar results (the reported values correspond to the geometric means of pace^{LB} across all the instances of the data set and the best original rule was always the `dual`).

	Baseline	**WS**	**WS + BT**
Best branching rule (across all instances)	12.402	11.490	5.083
Out-of-bag Q-RF	7.467	7.699	4.636
Optimal rule (instance by instance)	6.129	5.786	4.140
Improvement after learning (with Q-RF)	39.8%	33.0%	8.8%
Optimal improvement (upper bound for learning)	50.6%	49.6%	18.6%

Table 3.4: Performance of ML-based rules according to pace^{LB} (out-of-bag).

In order to get additional insights from the above results, we now explore in depth the performance of the ML-based branching rule Q-RF in the baseline setup, building upon the out-of-bag predictions.[4] In Figure 3.2, the complete set of instances for the three libraries is used (total of 354 instances). The boxplots show that npace^{LB} is significantly closer to 1 for Q-RF than for any of the original rules. For the bar chart, it also validates that Q-RF is outperforming the other branching rules as it is ranked number 1 (best) in terms of the pace more often (38.98% compared to range which is 36.72%). The numbers inside the bars indicate average value of npace^{LB} for the problems in each rank position. Thus, even when Q-RF is ranked 6th (worst) its average normalized pace value is 0.7 which is also better than the corresponding value for each of the other six branching rules (ranging from 0.43-0.69). In particular, the plots in Figure 3.2 show that, although the learned rule chooses the best rule in less than 50% of the instances, it consistently chooses rules whose performance is very close to that of the best rule. This last observation also allows to understand why the improvement after learning with Q-RF, reported in Tables 3.3 and 3.4, is close to the optimal improvement.

In Figure 3.3 we separate the results in Figure 3.2 for the three sets of instances, DS, MINLPLib, and QPLIB. The number of instances within each library is indicated in parenthesis. There are some differences across data sets. For instance, the box plots show little variation in npace^{LB} across the branching rules for DS set compared to the other two sets. `max` is the worst one, while the other rules are close in terms of normalized pace.

[4] We have also run the corresponding analysis for WS and WS+BT configurations, obtaining qualitatively similar results.

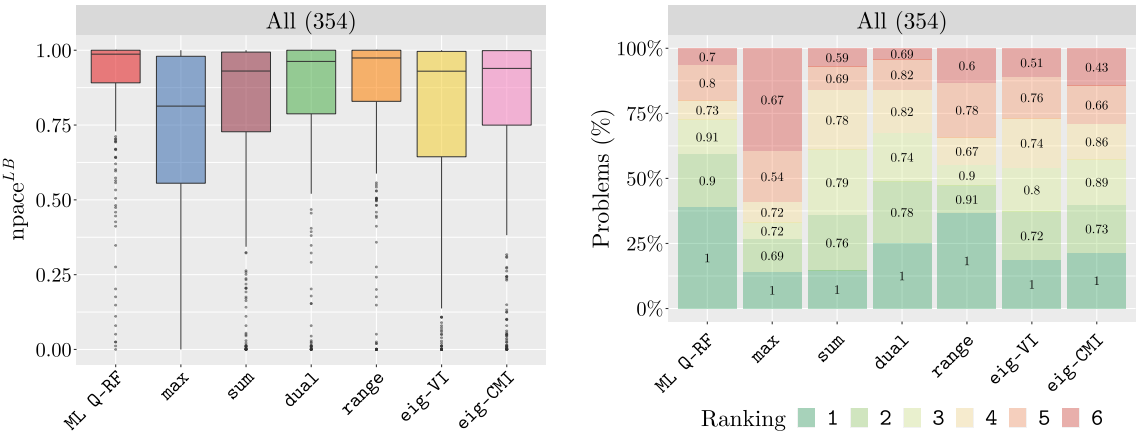
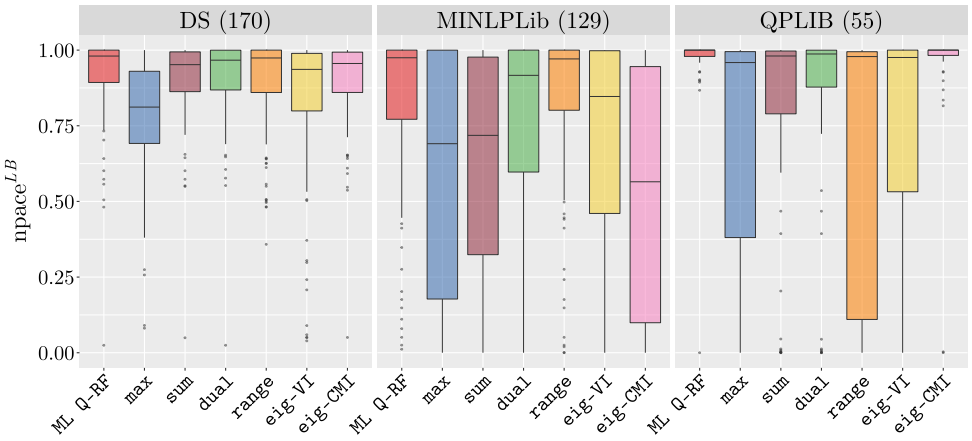


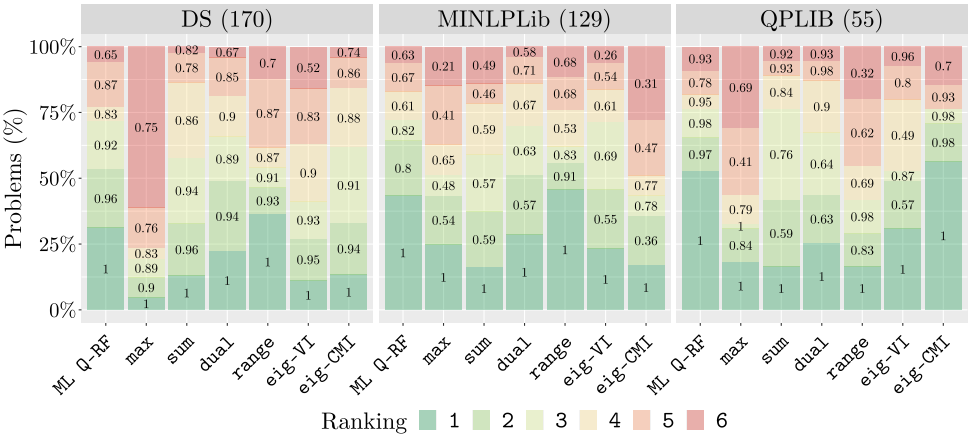
Figure 3.2: Comparative analysis of the performance of the different branching rules, according to npace^{LB}. Left, boxplots for each rule. Right, each rule ranked from 1 (best) to 6 (worst).

For MINLPLib, `range` performs slightly better than `ML Q-RF` as the number of times that `range` ranked first is 45.73% compared to 43.41% for `ML Q-RF`. For QPLIB, both `ML Q-RF` and `eig-CMI` strongly dominate in terms of performance, ranking first 52.72% and 56.36% of the instances, respectively. Note that for this set, the `range` rule, which performed very well for MINLPLib, is the worst branching rule for QPLIB. In particular, it is ranked 6th in around 20% of the instances with an average npace^{LB} of 0.32 in them, while the other rules, when ranked 6th, have an average npace^{LB} ranging from 0.69 to 0.96. Therefore, the ML-based rule seems to understand which are the features within and across data sets that drive the performance of the original rules.

Next, we further explore to what extent the ML-based rule manages to learn the patterns behind optimal rule selection. Figure 3.4 presents a comparison of the frequencies with which the original rules are chosen by the ML-based rule and the optimal rule. The behaviour of the ML-based branching rule strongly resembles that of the theoretical optimal branching rule in all three data sets. For DS the dominant rule is `range`, with `dual` and `eig-CMI` also playing an important role. For MINLPLib, `range` dominates again and, surprisingly, `max` is chosen quite often and so is again `dual`. For QPLIB, `eig-CMI` is chosen for almost 50% of the instances for both the ML-based rule and the optimal rule. Interestingly, in this last set of instances, `range` is rarely selected as the rule for branching. Recall that our offline learning is performed jointly on all sets of instances, so the results in Figure 3.4, showing that the ML-based rule correctly adapts to the instances in DS, MINLPLib, and QPLIB, are a sign of the quality of the learning process and of its potential to learn from highly heterogeneous instances.



(a) Boxplots of npace^{LB} for each branching rule.



(b) Each branching rule ranked from 1 (best) to 6 (worst), according to npace^{LB}.

Figure 3.3: Comparative analysis of the performance of the different branching rules.

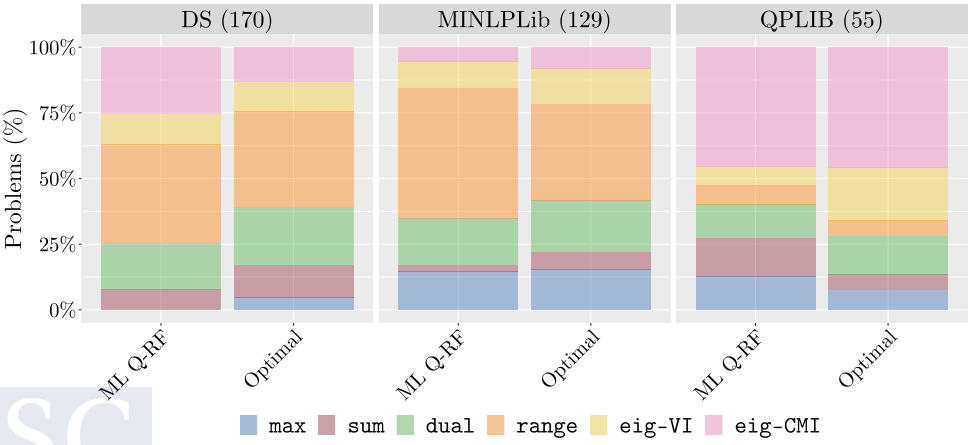


Figure 3.4: Percentage of times that each branching rule is selected by `ML Q-RF` and the optimal branching rule, according to npace^{LB}.

One advantage of the statistical learning techniques used in our analysis is the interpretability. In particular, (quantile) random forests allow to assign scores to the different features and, hence, understand the ones that are the most important for the associated predictions. Since our Q-RF builds upon six independent regressions, one for each of the original branching rules, we have six different scores for each explanatory feature. Figure 3.5 presents, for each of the six regressions, the feature importance scores of the top 15 features overall. A first observation is that the novel graph-related features are important, as three of them appear in the top 10: CMIG.mod, VIG.mod, and VIG.Dens. Further, CMIG.mod, the modularity on CMIG, is in first position. Different features have different importance score for learning different branching rules. In particular, graph-related features are important for the corresponding graph-based rules: VIG.mod and VIG.Dens are particularly important for `eig-VI` and CMIG.mod score is very high for `eig-CMI`. Additionally, the most important features for learning `range` and `dual` branching rules are those related to the ranges of variables.

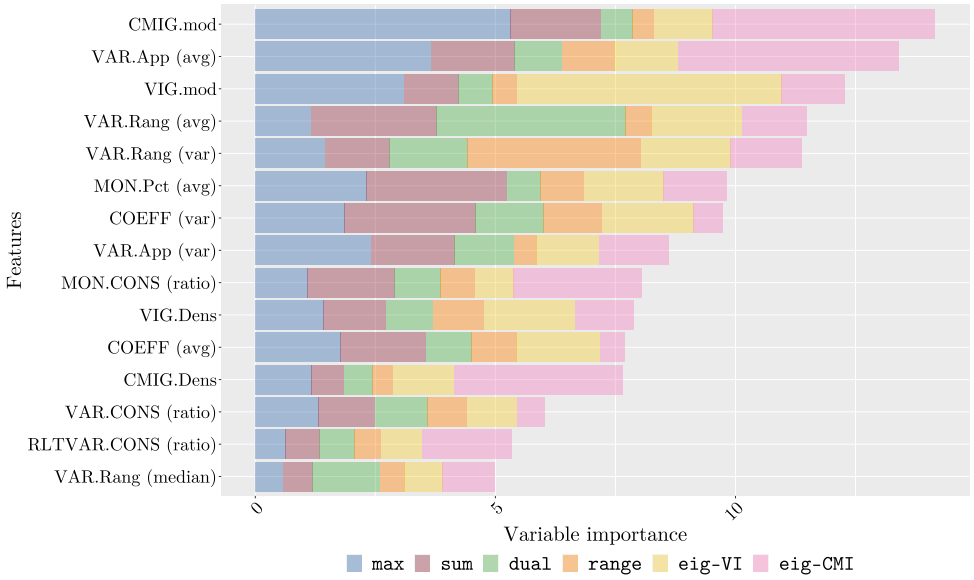


Figure 3.5: For each branching rule, variable importance score of the features obtained using `ML Q-RF`.

To conclude, we compare the performance of the different branching rules by showing the corresponding performance profiles (Dolan and Moré, 2002). In Figure 3.6 we present the performance profiles for the three sets of instances together and individually. Similarly to what we did in the computational analysis from Chapter 2 and in Table 3.3 and Table 3.4, we have discarded those instances that were solved by all configurations in less than 5 seconds and those instances for which no configuration could return a gap after the time

limit of one hour. Consistently with all the results we have already reported so far, `ML Q-RF` performs noticeably better than all the original branching rules. From the DS data set plot, `ML Q-RF` performs slightly better than 4 of the 6 branching rules (while `max` and `eig-VI` are significantly worse). For MINLPLib data set, `ML Q-RF` is clearly the best and `range` is in second position. For QPLIB data set, `ML Q-RF` is slightly better than `eig-CMI` and dominates the other 5 branching rules. Therefore, despite having that the original rules performing best vary significantly for the different sets of instances, `ML Q-RF` manages to slightly outperform the best rules in each individual set and solidly outperforms all the original rules overall.

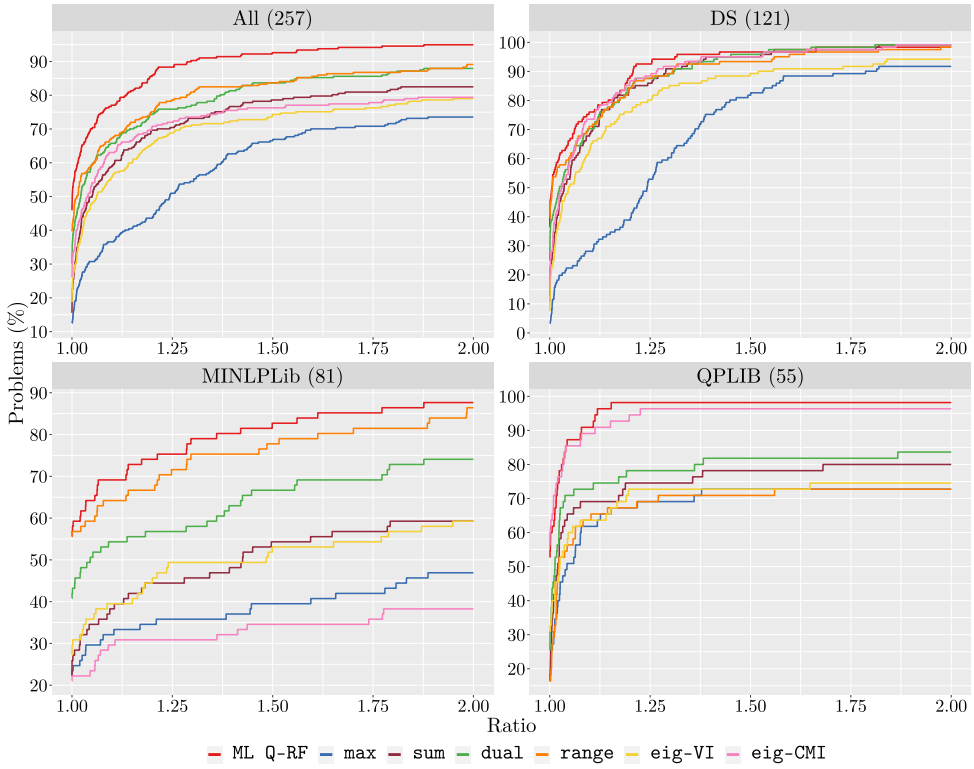


Figure 3.6: Performance profiles of pace^{LB} for each branching rule.

3.5 Conclusions and Future Research

We have presented a framework for learning rules for spatial branching and shown in different numerical experiments that the resulting gains in performance can be significant (up to 30%). These experiments have been performed jointly on a rich set of diverse instances taken from standard NLP libraries and from past literature. Interestingly, we

have also shown that the interpretability of our framework allows to understand the importance of the different explanatory features for the resulting ML-based rule. This kind of understanding, which may shed light for the feature-design of future contributions, has been overlooked by most of the past literature on "learning to branch".

As we briefly discussed in Section 3.3.1, the choice to perform statistical learning only with static features limits the potential of our learning setting. Arguably, neglecting node by node information and limiting the scope of the algorithm selection to be instance specific, instead of node specific, may result in learning a less effective branching rule. Yet, given that this work is the first contribution on "learning to branch" in the nonlinear setting, we believe that the static learning setting provides a natural starting point. First, since it involves no computational overhead for the computation of node-specific features, the differences in performance of the learned rule with respect to the original ones directly reflects the net effect of the learning process. Second, learning static rules allows for a clean definition of the "optimal" rule for each instance (this would not be entirely straightforward with dynamic rules, where the selection of the best "myopic" rule in each node might not lead to the best performance on the given instance). This is relevant since, once we pin down what can be achieved with perfect learning, one can easily assess the quality of the learning behind the rules resulting from our approach. Third, static learning provides a benchmark on which to assess the performance of more advanced dynamic techniques and, in particular, to evaluate to what extent the online computational overhead of node-specific features is compensated by the reduction on the tree size.

Given the above discussion, a prominent direction for future research is the incorporation of dynamic features and study the performance of the resulting rules taking the approach developed in this chapter as a benchmark. Additionally, future research should aim at having richer portfolios of branching rules, incorporating for instance rules that build upon reliability branching or violation transfer (see, for instance, Tawarmalani and Sahinidis (2004); Achterberg et al. (2005); Belotti et al. (2009)).

BIBLIOGRAPHY

ACHTERBERG, T., T. KOCH, AND A. MARTIN (2005): "Branching rules revisited," *Operations Research Letters*, 33, 42–54.

Alvarez, A. M., Q. Louveaux, and L. Wehenkel (2017): "A machine learning-based approximation of strong branching," *INFORMS Journal on Computing*, 29, 185–195.

Baltean-Lugojan, R., P. Bonami, R. Misener, and A. Tramontani (2019): "Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks," Optimization Online.

Belotti, P., J. Lee, L. Liberti, F. Margot, and A. Wächter (2009): "Branching and bounds tightening techniques for non-convex MINLP," *Optimization Methods and Software*, 24, 597–634.

Bengio, Y., A. Lodi, and A. Prouvost (2021): "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 290, 405–421.

Bienstock, D. and G. Muñoz (2018): "LP formulations for polynomial optimization problems," *SIAM Journal on Optimization*, 28, 1121–1150.

Bonami, P., A. Lodi, and G. Zarpellon (2022): "A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX," *Operations Research*, In press.

Breiman, L. (2001): "Random forests," *Machine Learning*, 45, 5–32.

Bussieck, M. R., A. S. Drud, and A. Meeraus (2003): "MINLPLib-A collection of test models for mixed-integer nonlinear programming," *INFORMS Journal on Computing*, 15, 114–119.

Clauset, A., M. E. J. Newman, and C. Moore (2004): "Finding community structure in very large networks," *Physical Review E*, 70, 66–111.

Dalkiran, E. and H. D. Sherali (2016): "RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems," *Mathematical Programming Computation*, 8, 337–375.

Di Liberto, G., S. Kadioglu, K. Leo, and Y. Malitsky (2016): "DASH: dynamic approach for switching heuristics," *European Journal of Operational Research*, 248, 943–953.

Dolan, E. D. and J. J. Moré (2002): "Benchmarking optimization software with performance profiles," *Mathematical Programming*, 91, 201–213.

FAENZA, Y., G. MUÑOZ, AND S. POKUTTA (2022): "New limits of treewidth-based tractability in optimization," *Mathematical Programming*, 191, 559–594.

FASIOLO, M., S. N. WOOD, M. ZAFFRAN, R. NEDELLEC, AND Y. GOUDE (2021a): "Fast calibrated additive quantile regression," *Journal of the American Statistical Association*, 116, 1402–1412.

——— (2021b): "Qgam: bayesian nonparametric nuantile negression modeling in R," *Journal of Statistical Software*, 100, 1–31.

FRIEDMAN, J. H. (2001): "Greedy function approximation: a gradient boosting machine," *The Annals of Statistics*, 29, 1189–1232.

——— (2002): "Stochastic gradient boosting," *Computational Statistics & Data Analysis*, 38, 367–378.

FULKERSON, D. R. AND O. A. GROSS (1965): "Incidence matrices and interval graphs," *Pacific Journal of Mathematics*, 15, 835–855.

FURINI, F., E. TRAVERSI, P. BELOTTI, A. FRANGIONI, A. GLEIXNER, N. GOULD, L. LIBERTI, A. LODI, R. MISENER, H. MITTELMANN, N. V. SAHINIDIS, S. VIGERSKE, AND A. WIEGELE (2018): "QPLIB: a library of quadratic programming instances," *Mathematical Programming Computation*, 11, 237–265.

GASSE, M., D. CHÉTELAT, N. FERRONI, L. CHARLIN, AND A. LODI (2019): "Exact combinatorial optimization with graph convolutional neural networks," arXiv.

GHADDAR, B., I. GÓMEZ-CASARES, J. GONZÁLEZ-DÍAZ, B. GONZÁLEZ-RODRÍGUEZ, B. PATEIRO-LÓPEZ, AND S. RODRÍGUEZ-BALLESTEROS (2022): "Learning for spatial branching: an algorithm selection approach," arXiv.

GOMES, C. P. AND B. SELMAN (2001): "Algorithm portfolios," *Artificial Intelligence*, 126, 43–62.

GONZÁLEZ-RODRÍGUEZ, B., J. OSSORIO-CASTILLO, J. GONZÁLEZ-DÍAZ, Á. M. GONZÁLEZ-RUEDA, D. R. PENAS, AND D. RODRÍGUEZ-MARTÍNEZ (2022): "Computational advances in polynomial optimization: RAPOSa, a freely available global solver," *Journal of Global Optimization*, In press.

GREENWELL, B., B. BOEHMKE, AND J. CUNNINGHAM (2020): *Gbm: generalized boosted regression models*, R package version 2.1.8.

Gupta, P., M. Gasse, E. B. Khalil, M. P. Kumar, A. Lodi, and Y. Bengio (2020): "Hybrid models for learning to branch," *Proceedings of the International Conference on Neural Information Processing System*, 34, 18087–18097.

Hastie, T., R. Tibshirani, and J. Friedman (2009): *The elements of statistical learning*, Springer New York.

Khalil, E. B., P. Le Bodic, L. Song, G. L. Nemhauser, and B. Dilkina (2016): "Learning to branch in mixed-integer programming," *Proceedings of the AAAI Conference on Artificial Intelligence*, 30.

Koenker, R. (2005): *Quantile regression*, Cambridge University Press.

Koenker, R., V. Chernozhukov, X. He, and L. Peng (2017): *Handbook of quantile regression*, Chapman and Hall/CRC.

Kriegler, B. and R. Berk (2010): "Small area estimation of the homeless in Los Angeles: an application of cost-sensitive stochastic gradient boosting," *The Annals of Applied Statistics*, 4, 1234–1255.

Latora, V., V. Nicosia, and G. Russo (2017): *Complex networks: principles, methods and applications*, Cambridge University Press.

Leyton-Brown, K., E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham (2003): "A portfolio approach to algorithm select," *Proceedings of the International Joint Conference on Artificial Intelligence*, 18, 1542–1543.

Linderoth, J. T. and M. W. P. Savelsbergh (1999): "A computational study of search strategies for mixed-integer programming," *INFORMS Journal on Computing*, 11, 173–187.

Lodi, A. and G. Zarpellon (2017): "On learning and branching: a survey," *TOP*, 25, 207–236.

Meinshausen, N. (2006): "Quantile regression forests," *Journal of Machine Learning Research*, 7, 983–999.

R Core Team (2022): *R: a language and environment for statistical computing.*

Robertson, N. and P. D. Seymour (1984): "Graph minors. III. Planar tree-width," *Journal of Combinatorial Theory, Series B*, 36, 49–64.

SHERALI, H. D. AND C. H. TUNCBILEK (1992): "A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique," *Journal of Global Optimization*, 2, 101–112.

TAWARMALANI, M. AND N. V. SAHINIDIS (2004): "Global optimization of mixed-integer nonlinear programs: a theoretical and computational study," *Mathematical Programming*, 99, 563–591.

WRIGHT, M. N. AND A. ZIEGLER (2017): "Ranger: a fast implementation of random forests for high dimensional data in C++ and R," *Journal of Statistical Software*, 77, 1–17.

Chapter 4

Enhancing RLT with Conic Constraints

4.1 Introduction

The contents of this chapter are based on González-Rodríguez et al. (2022a). The large volume of theoretical and computational research on conic optimization has led to important advances over the last few years in the efficiency and robustness of the associated algorithmic procedures to solve them. Leading state-of-the-art mixed-integer linear programming (MILP) solvers such as `Gurobi` (Gurobi Optimization, LLC, 2022), `CPLEX` (IBM ILOG, 2022), and `Xpress` (FICO, 2022) have recently added functionalities that allow to efficiently solve second-order cone programming (SOCP) problems and, further, `Mosek` (Mosek ApS, 2022) has positioned itself as a reliable solver for general semidefinite programming (SDP) problems.

Importantly, despite the convex nature of conic optimization problems, they are also proving to be a powerful tool in the design of branch-and-bound algorithms for general nonconvex mixed-integer nonlinear programming (MINLP) problems and, especially, polynomial optimization problems. The developments of the last several decades have shown that conic optimization is a central tool in addressing nonconvexities. These advances have been more prominent in the case of polynomial optimization problems, and even more so in the particular case of (quadratically-constrained) quadratic optimization problems, where a variety of convex relaxations have been thoroughly studied (Shor, 1987; Ghaddar et al., 2011b; Burer and Ye, 2020; Bonami et al., 2019; Elloumi and Lambert, 2019). These conic-based relaxations include semidefinite programming and second-order cone programming as their powerhouses. They often provide fast and guaranteed approaches for computing bounds on the global value of the nonconvex optimization problem at hand. For instance, Lasserre (2001) introduced semidefinite relaxations corresponding to liftings of the polynomial programs into higher dimensions. The construction is motivated by results related to representations of nonnegative polynomials as sum-of-squares (Parrilo, 2003) and the dual theory of moments. This led to the development of systematic approaches for solving polynomial optimization problems to global optimality, the main limitation of

these approaches currently being that they are computationally prohibitive in general.

Nowadays, state-of-the-art solvers tackle MINLP problems and, in particular, polynomial optimization ones through different relaxations to be solved at the nodes of the branch-and-bound tree. A crucial direction of current research focuses on integrating tighter and tighter relaxations while preserving good computational properties, with relaxations that build upon different families of conic constraints becoming increasingly important. Probably the most common approach comes from the identification of linear SDP-based cuts that tighten the relaxations as the algorithm progresses (Sherali et al., 2012; Baltean-Lugojan et al., 2019; González-Rodríguez et al., 2022b). In this work we are interested in the alternative approach of directly adding SDP constraints to the relaxations as in Burer and Vandenbussche (2008) and Buchheim and Wiegele (2013). Probably the main reason why this approach has received less attention is that the resulting algorithms need to rely on SDP solvers which, until recently, were probably not reliable enough and too expensive computationally in some instances. One of the primary goals of this chapter is to show that the situation has changed, and that general branch-and-bound schemes based on the solution of an SDP at each and every node of the branch-and-bound tree can be very competitive and even superior to previous approaches.

In order to achieve the above goal, in this chapter we introduce and compare linear SDP-based constraints as well as SOCP and SDP constraints. Importantly, their definition ensures that they are efficient in the sense of preserving the size and sparsity of the RLT-based relaxation for polynomial optimization problems introduced in Sherali and Tuncbilek (1992) and discussed in detail in Chapter 1. We consider a total of nine different versions of constraints to be added: one based on linear SDP-based cuts, four based on SOCP constraints, and four based on SDP constraints. These conic constraints are then integrated into the polynomial optimization solver `RAPOSa`, introduced in Chapter 2. As a second step, we then develop thorough computational studies on well-established benchmarks including randomly generated instances from Dalkiran and Sherali (2016), MINLPLib instances (Bussieck et al., 2003), and QPLIB instances (Furini et al., 2018), which provide a wide variety of classes of polynomial optimization problems. One of the main findings is that, in around 50% of the instances, the best performance is achieved by one of the versions explicitly incorporating SOCP and SDP conic constraints. The remaining 50% is split quite evenly between the baseline RLT and a version of it incorporating SDP-based linear cuts. Importantly, the analysis also allowed to identify particular classes of problems where one specific family of SOCP/SDP conic constraints is consistently superior to the linear versions.

To the best of our knowledge, our contribution represents one of very few implementa-

tions of branch-and-bound schemes with conic relaxations for broad classes of problems, with the added value of the generality of the resulting scheme, since it can be applied to any given polynomial optimization problem. The most related approaches are Burer and Vandenbussche (2008) for nonconvex quadratic problems with linear constraints and Buchheim and Wiegele (2013) for unconstrained mixed-integer quadratic problems. In Burer and Vandenbussche (2008), the authors present a computational analysis in which they compare the relative performance of different SDP relaxations, with the main highlight being that "only a small number of nodes are required" to fully solve the problems at hand.

Buchheim and Wiegele (2013) introduce a new branch-and-bound algorithm, `Q-MIST` (Quadratic Mixed-Integer Semidefinite programming Technique), and develop a thorough computational analysis in which they show that, for a wide variety of families of instances within the scope of their algorithm, `Q-MIST` notably outperforms `Couenne` (Belotti et al., 2009). It is worth noting that, if looking at specific classes of optimization problems, then one can find additional successful implementations of branch-and-bound algorithms with the inclusion of tailor-made conic constraints, such as Ghaddar and Jabr (2019) for optimal power flow problems and combinatorial optimization problems, Krislock et al. (2017) and Rendl et al. (2010) for maximum cut problems, Piccialli et al. (2022) for minimum sum-of-squares clustering, and Ghaddar et al. (2011a) for maximum k-cut problem.

Last, but not least, in our computational experiments we also observe that there is a lot of variability in the best performing version of conic constraints for the different instances, with each version beating the rest for a nonnegligible number of instances. This observation motivates the last contribution of this work, in which we exploit this variability by learning to choose the best version among our portfolio of different constraints. Building upon the framework in Chapter 3, we show that the resulting machine learning version significantly outperforms each and every one of the underlying versions. This last contribution naturally fits into the rapidly emerging strand of research on "learning to optimize", whose advances are nicely presented in the survey papers Lodi and Zarpellon (2017) and Bengio et al. (2021). The closest approach to this last part of our contribution is Baltean-Lugojan et al. (2019), in which deep neural networks are used to rank SDP-based cuts for quadratic problems. Then, only the top scoring cuts are added, aiming to obtain a good balance between the tightness and the complexity of the relaxations. A common feature of the SDP-based cuts used in Baltean-Lugojan et al. (2019) and those in this work is that they are designed with a big emphasis in sparsity considerations (number of nonzeros in the constraints), with the goal of obtaining computationally efficient relaxations. From the point of view of the learning process the approaches are quite different, since they focus on one type of constraints (the SDP-based cuts) and they want to learn the best SDP-based

cuts to add at a given node, whereas we want learn to choose for any given instance which conic constraints to include in the relaxations.

The contribution of this work can be summarized as follows. First, we define and show the potential of different SOCP and SDP strengthenings of the classic RLT relaxations for general polynomial optimization problems in a branch-and-bound scheme. Second, we design a statistical learning approach to learn the best strengthtening to use on a given instance and obtain promising results for the resulting algorithm.

The remainder of this chapter is organized as follows. In Section 4.2 we present the different families of conic constraints that will be integrated within the baseline RLT implementation. In Section 4.3 we present a first series of computational results. Then, in Section 4.4 we show how the conic constraints can be further exploited withing a machine learning framework. Finally, we conclude in Section 4.5.

4.2 Conic Enhancements of RLT

The analysis developed in this chapter builds upon the theoretical results from Dalkiran and Sherali (2013) and, therefore, only the bound-factor constraints associated to J-sets are incorporated to the linear relaxation of $(PP(\Omega))$. The ensuing relaxations are less tight but on the other hand, they are smaller in size and, hence, faster to solve. Note that the use of J-sets reduces not only the number of constraints but, more importantly, also the number of RLT variables. As it can be seen in Dalkiran and Sherali (2013) and in Section 2.4.1, the performance of the RLT-based algorithm is clearly superior when the J-set approach is followed.

As already discussed in Section 4.1, the use of semidefinite programming to improve the performance of branch-and-bound schemes is not new, and Baltean-Lugojan et al. (2019) provides a thorough and up-to-date review of the field. Typically, the goal is to rely on semidefinite programming to tighten the relaxations of the original nonconvex optimization problems targeted by a branch-and-bound algorithm. We start this section by reviewing the main ingredient of such approaches and then present various families of SDP-driven constraints that can be incorporated into the RLT relaxations in an efficient way. In particular, they should preserve the underlying dimensionality and sparsity of the problem, which is crucial for these approaches to be competitive.

The main ingredient that has to be specified is the matrix or matrices on which positive semidefiniteness is to be imposed. To each (multi-)set of variables $\{x_j\}_{j\in J}$, with $J \subset N^\delta$, one can associate a vector $\boldsymbol{y}_J = (x_j)_{j\in J}$ (resulting from the concatenation of all the variables in J, including repetitions). To any such vector one can associate matrix $\boldsymbol{M} = [\boldsymbol{y}\boldsymbol{y}^\top]$, which

is trivially positive semidefinite. Now, let $\boldsymbol{M}_L = [\boldsymbol{y}\boldsymbol{y}^\intercal]_L$ be the matrix obtained when each monomial in $\boldsymbol{M}$ is replaced by the corresponding RLT variable in the lifted space. The constraint $\boldsymbol{M}_L \succcurlyeq 0$ is a valid cut because it never removes feasible solutions of ($PP(\Omega)$) and so it does not compromise convergence of the RLT-based algorithm to a global optimum as we mentioned in Section 2.5.6. In practice, vectors of the form $(1, (x_j)_{j\in J})$ are often preferred, since they result in $\boldsymbol{M}_L$ matrices containing also variables in the original space and not only RLT variables, and lead to tighter relaxations. In Sherali et al. (2012), for instance, the authors discuss different ways of defining $\boldsymbol{y}$ for a given J. Specifically, they mainly work with $\boldsymbol{y}^1 = (x_j)_{j\in J}$, $\boldsymbol{y}^2 = (1, (x_j)_{j\in J})$, and $\boldsymbol{y}^3 = (1, (x_j)_{j\in J}, \ldots)$, defined by concatenating also monomials of degree greater than one, while ensuring that no monomial in the resulting $\boldsymbol{M}$ has degree larger than δ, the degree of ($PP(\Omega)$).

We now move to the definition of the specific SDP-driven constraints for the RLT-based algorithm that constitute the subject of study in this chapter.

To illustrate these kinds of constraints, we use the following polynomial programming problem:

$$\begin{aligned} \text{minimize} \quad & x_1^2 + x_2^2 + x_1x_2x_3 \\ \text{subject to} \quad & x_1x_2 + x_1x_4 \geq 1 \\ & 1 \leq x_1 \leq 10 \\ & 0 \leq x_2 \leq 8 \\ & 0 \leq x_3 \leq 15 \\ & 0 \leq x_4 \leq 7. \end{aligned} \tag{4.1}$$

Note that the monomials with degree greater than one are $\{1,1\}$,$\{2,2\}$, $\{1,2,3\}$, $\{1,2\}$, and $\{1,4\}$. Since $\{1,2\}$ is included in $\{1,2,3\}$, it is removed. Therefore, the J-sets are $\{1,1\}$, $\{2,2\}$, $\{1,2,3\}$, and $\{1,4\}$.

4.2.1 Linear SDP-Based Constraints

In Sherali et al. (2012), the authors associate linear cuts to the constraints of the form $\boldsymbol{M}_L \succcurlyeq 0$ as follows. Given the solution at a given relaxation, they evaluate matrix $\boldsymbol{M}_L$ at that particular point, obtaining matrix $\bar{\boldsymbol{M}}_L$. Then, they evaluate the positive semidefiniteness of $\bar{\boldsymbol{M}}_L$. More precisely, they develop different methods to look for an eigenvector $\boldsymbol{\alpha}$ associated to a negative eigenvalue of $\bar{\boldsymbol{M}}_L$. For each such vector, one can add the valid cut $\boldsymbol{\alpha}^\intercal \boldsymbol{M}_L \boldsymbol{\alpha} \geq 0$ to the linear relaxation to separate the current solution. A potential drawback of these cuts is that they may be very "dense", in the sense of involving

a large number of variables, which may increase the solving time of the relaxations. Thus, as already discussed in Sherali et al. (2012), it is important to carefully chose the sets J used to define the $\boldsymbol{M}_L$ matrices.

The sparsity of the cuts is particularly important if the RLT-based algorithm is being run with the J-set approach (Section 2.2.1) since, in general, the resulting cuts might involve monomials not contained in any J-set. This would require to include additional RLT variables in the relaxations (and the corresponding bound-factor constraints), increasing the size and potentially reducing the sparsity of the relaxations. Here we follow Section 2.5.6, where we only consider $\boldsymbol{y}$ vectors associated with J-sets of ($PP(\Omega)$). If vectors $\boldsymbol{y}^1$ or $\boldsymbol{y}^2$ are used, the resulting cuts preserve well the size of the relaxations and its sparsity. In particular, in analysis from Section 2.5.6, the SDP cuts based on $\boldsymbol{y}^2$ are the ones delivering the best results and so we use them in this chapter as well. The above discussion regarding the adequacy of building constraints based on J-sets, in order to obtain sparser constraints (number of nonzero coefficients) and to preserve the size (number of variables) of the resulting relaxations, also applies to the conic constraints defined in the following which also build upon J-sets.

4.2.2 SDP Constraints

We next describe two approaches to tighten the classic RLT relaxations by directly adding semidefinite constraints. They just differ in the matrices on which positive semidefiniteness is imposed.

Approach 1. For each J-set J, $\boldsymbol{y}^1$ is used to define the $\boldsymbol{M}_L$ matrix and the constraint $\boldsymbol{M}_L \succcurlyeq 0$. Note that, whenever J contains only one variable x_i (possibly multiple times), this would result in a trivial constraint and, therefore, these constraints are disregarded with one exception: if $|J| = 2$, then $\boldsymbol{y}^1$ is replaced with $(1, x_i)$. With this exception, this approach is mathematically equivalent for quadratic problems to the SOCP approach we present in Section 4.2.3 below (see Proposition 4.1).

Approach 2. For each J-set J, $\boldsymbol{y}^2$ is used to define the $\boldsymbol{M}_L$ matrix and the constraint $\boldsymbol{M}_L \succcurlyeq 0$.

Preliminary analysis has shown that constraints building upon $\boldsymbol{y}^3$, although they lead to tighter relaxations, generate significantly bigger matrices and increase the complexity of the resulting SDP problems.

To provide an example of both approaches, consider the polynomial optimization problem in (4.1). Then, the semidefinite constraints added with the above approaches are

the following ones:

Approach 1

$$\begin{pmatrix} 1 & x_1 \\ x_1 & X_{11} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} 1 & x_2 \\ x_2 & X_{22} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} X_{11} & X_{12} & X_{13} \\ X_{12} & X_{22} & X_{23} \\ X_{13} & X_{23} & X_{33} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} X_{11} & X_{14} \\ X_{14} & X_{44} \end{pmatrix} \succcurlyeq 0.$$

Approach 2

$$\begin{pmatrix} 1 & x_1 & x_1 \\ x_1 & X_{11} & X_{11} \\ x_1 & X_{11} & X_{11} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} 1 & x_2 & x_2 \\ x_2 & X_{22} & X_{22} \\ x_2 & X_{22} & X_{22} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} 1 & x_1 & x_2 & x_3 \\ x_1 & X_{11} & X_{12} & X_{13} \\ x_2 & X_{12} & X_{22} & X_{23} \\ x_3 & X_{13} & X_{23} & X_{33} \end{pmatrix} \succcurlyeq 0, \begin{pmatrix} 1 & x_1 & x_4 \\ x_1 & X_{11} & X_{14} \\ x_4 & X_{14} & X_{44} \end{pmatrix} \succcurlyeq 0.$$

4.2.3 SOCP Constraints

We now describe the second-order cone constraints which, with respect to the SDP ones, lead to looser relaxations but, on the other hand, can be solved more efficiently by state-of-the-art optimization solvers. For each J-set J and each pair of variables present in J, $x_i \neq x_j$, we define the following second-order cone constraint:

$$\frac{X_{ii} + X_{jj}}{2} \geq \left\| \begin{pmatrix} X_{ij} \\ \frac{X_{ii} - X_{jj}}{2} \end{pmatrix} \right\|_2. \tag{4.2}$$

We argue now why these constraints are valid cuts. First, constraint (4.2) can be rewritten as $X_{ii}X_{jj} \geq X_{ij}^2$. Now, given a solution of a linear relaxation satisfying $X_J = \prod_{j \in J} x_j$ for each $J \subset N^\delta$, the above condition reduces to $x_i x_i x_j x_j \geq x_i x_j x_i x_j$, which is trivially true.

Note that constraints in (4.2) are trivially true if $i = j$ and, hence, whenever we have a variable x_i appearing twice or more in J, we instead add the second-order constraint

$$\frac{1 + X_{ii}}{2} \geq \left\| \begin{pmatrix} x_i \\ \frac{1 - X_{ii}}{2} \end{pmatrix} \right\|_2, \tag{4.3}$$

which is equivalent to $X_{ii} \geq x_i x_i$ and, for solutions satisfying $X_J = \prod_{j \in J} x_j$ for each $J \subset N^\delta$, is again trivially true. Thus, Theorem 1.8 guarantees the convergence of the resulting algorithm.

Consider again the polynomial optimization problem in (4.1). Then, the SOCP constraints added are the following ones:

$$\frac{1 + X_{11}}{2} \geq \left\| \begin{pmatrix} x_1 \\ \frac{1 - X_{11}}{2} \end{pmatrix} \right\|_2, \frac{1 + X_{22}}{2} \geq \left\| \begin{pmatrix} x_2 \\ \frac{1 - X_{22}}{2} \end{pmatrix} \right\|_2, \frac{X_{11} + X_{22}}{2} \geq \left\| \begin{pmatrix} X_{12} \\ \frac{X_{11} - X_{22}}{2} \end{pmatrix} \right\|_2,$$

$$\frac{X_{11}+X_{33}}{2} \geq \left\| \begin{pmatrix} X_{13} \\ \frac{X_{11}-X_{33}}{2} \end{pmatrix} \right\|_2, \quad \frac{X_{22}+X_{33}}{2} \geq \left\| \begin{pmatrix} X_{23} \\ \frac{X_{22}-X_{33}}{2} \end{pmatrix} \right\|_2, \quad \frac{X_{11}+X_{44}}{2} \geq \left\| \begin{pmatrix} X_{14} \\ \frac{X_{11}-X_{44}}{2} \end{pmatrix} \right\|_2.$$

Now, we present a result which is well known, as can be seen for instance in Kuang et al. (2017). However, we include the proof here to make this thesis as self-contained as possible.

Proposition 4.1. *Approach 1 from Section 4.2.2 and SOCP approach from this section are equivalent for quadratic problems.*

Proof. The proof of the proposition is reduced to prove that

$$\boldsymbol{M} = \begin{pmatrix} M_{11} & M_{12} \\ M_{12} & M_{22} \end{pmatrix} \succcurlyeq 0 \Leftrightarrow \frac{M_{11}+M_{22}}{2} \geq \left\| \begin{pmatrix} M_{12} \\ \frac{M_{11}-M_{22}}{2} \end{pmatrix} \right\|_2, \tag{4.4}$$

taking into account that $M_{11}, M_{12}, M_{22} \geq 0$.

Indeed, the left side of (4.4) is equivalent to the claim that the eigenvalues of matrix $\boldsymbol{M}$ are nonnegative. The characteristic polynomial of $\boldsymbol{M}$ is $p_{\boldsymbol{M}}(t) = t^2 - (M_{11}+M_{22})t + M_{11}M_{22} - M_{12}^2$. Therefore, the eigenvalues of matrix $\boldsymbol{M}$ are given by

$$\lambda = \frac{M_{11}+M_{22} \pm \sqrt{(M_{11}-M_{22})^2 + 4M_{12}^2}}{2}.$$

Thus, both eigenvalues are nonnegative if and only if $M_{11}+M_{22} \geq \sqrt{(M_{11}-M_{22})^2+4M_{12}^2}$, which is equivalent to $M_{11}M_{22} \geq M_{12}^2$. After some straightforward algebra, we can see that the condition in the right side of (4.4) is equivalent to $M_{11}M_{22} \geq M_{12}^2$. □

4.2.4 Binding SOCP and SDP Constraints

Since solving SOCP or SDP problems is usually more time-consuming than solving linear programming problems, we define a new approach in order to reduce the time needed for solving the resulting RLT relaxation with SOCP or SDP constraints. This consists of checking which conic constraints (second-order cone or semidefinite) are binding after solving the first relaxation, *i.e.*, this is done only once, at the root node. Thereafter, only these binding constraints are used to tighten the future linear relaxations. This approach significantly reduces the number of second-order cone or semidefinite constraints in the relaxations and, although these new relaxations are not as tight, one might expect that the binding constraints at the root node tend to be the most important ones in subsequent relaxations, at least in the first phase of the algorithm. We assess the trade-off between the

difficulty of solving the relaxations and how tight they are in the computational analysis in Section 4.3.

4.3 Computational Results

4.3.1 Testing Environment

All the computational analysis reported in this chapter have been performed on the supercomputer Finisterrae III, provided by Galicia Supercomputing Centre (CESGA). Specifically, we use computational nodes powered with 32 cores Intel Xeon Ice Lake 8352Y CPUs with 256GB of RAM connected through an Infiniband HDR network, and 1TB of SSD.

Regarding the data sets, we use the same ones as in Chapter 3: DS (Dalkiran and Sherali, 2016), MINLPLib (Bussieck et al., 2003), and QPLIB (Furini et al., 2018), resulting in a total of 409 instances.[1] We develop our analysis by building upon the global solver for polynomial optimization problems `RAPOSa` introduced in Chapter 2.[2] Regarding the auxiliary solvers, `RAPOSa` uses i) `Gurobi` for the linear relaxations ii) `Gurobi` and `Mosek` for the SOCP relaxations, and iii) `Mosek` for the SDP relaxations. The main objective of the thorough numerical analysis developed in this and in the following section is to assess the performance of different SOCP/SDP conic-driven versions of `RAPOSa` with respect to two more traditional ones: basic RLT and RLT with linear SDP-based cuts. More precisely, the full set of ten different versions is as follows:

- `RLT`: standard RLT-based algorithm (with J-sets).
- `SDP cuts`: linear SDP-based cuts added to the RLT relaxation.
- `SOCP`^{G}: SOCP constraints added to the RLT relaxation and solved with `Gurobi`.
- `SOCP`^{G,B}: same as above, but using only constraints that were binding at the root node.
- `SOCP`^{M}: SOCP constraints added to the RLT relaxation and solved with `Mosek`.
- `SOCP`^{M,B}: same as above, but using only constraints that were binding at the root node.

[1] In this case, the only instances we have discarded are two instances from MINLPLib because none of the `RAPOSa`'s configurations could solve the relaxation at the root node in the time limit.

[2] As in Chapter 3, we use the baseline version from Chapter 2 including the enhancements from Section 2.5.1 and Section 2.5.3.

- `SDP`^{1}: SDP constraints added to the RLT relaxation following Approach 1 and solved with `Mosek`.
- `SDP`^{1,B}: same as above, but using only constraints that were binding at the root node.
- `SDP`^{2}: SDP constraints added to the RLT relaxation following Approach 2 and solved with `Mosek`.
- `SDP`^{2,B}: same as above, but using only constraints that were binding at the root node.

For each instance and each one of the above options, we run `RAPOSa` with a time limit of one hour.

4.3.2 Numerical Results and Analysis

The main goal of the numerical analysis in this section is to show the potential of SOCP/SDP conic-constraints to improve upon the performance of more classic implementations of the RLT-based algorithm, such as `RLT` and `SDP cuts`. The measures used to evaluate the performance of the different versions of `RAPOSa` are pace^{LB} and npace^{LB}, the two KPIs introduced in Chapter 3 that capture the pace at which a given algorithm closes the gap or, more precisely, the pace at which it increases the lower bound along the branch-and-bound tree.

As thoroughly discussed in Chapter 3, pace^{LB} and npace^{LB} are natural measures that allow to compare the performance of different solvers/algorithms on all the instances of a data set at once, regardless of their difficulty and of how many versions of the underlying solver/algorithm have solved them to optimality. This is different from more common approaches, where the running time is used to evaluate performance on instances solved by all versions, the optimality gap is used for those instances solved by none, and where some decision has to be made regarding those instances solved by some but not all of the versions of the solver/algorithm.

Figure 4.1 shows, for the different sets of problems, the percentage of instances in which each one of the ten versions is the best one. We can see that, quite consistently across the three data sets, a version with either SOCP or SDP constraints is the best one in around 50% of the instances. This is in itself one of the main highlights of this work: RLT versions incorporating nonlinear SOCP/SDP constraints can improve the performance of RLT-based algorithms in half of the instances of the sets of problems under consideration.

Figure 4.1 contains more valuable information. First, the versions based on SOCP constraints come out on top much more often than those based on SDP constraints. Regarding the solvers for SOCP versions, it seems that `Gurobi` performs notably better in

Figure 4.1: Percentage of times that each version delivers the best performance, according to npace^{LB}.

MINLPLib instances, whereas `Mosek` is overwhelmingly better on DS and also comes out on top on QPLIB. When comparing binding versions with their nonbinding counterparts we can see that, for SDP versions, `SDP`^{1,B} and `SDP`^{2,B} seem to come out on top significantly more often than `SDP`^{1} and `SDP`^{2}, respectively. In the case of SOCP versions, there is no clear winner between the binding and nonbinding versions. Finally, regarding the RLT versions without conic constraints, `RLT` and `SDP cuts`, we can see that each of them turns out to be the best option in around 25% of the instances, with `SDP cuts` looking preferable in DS and QPLIB, whereas `RLT` comes out on top three times as much in MINLPLib.

Importantly, Figure 4.1 and the preceding discussion show that there is a lot of variability, with all 10 versions showing up as the best choice for a nonnegligible percentage of the instances. Further, this variability also follows different patterns for the different sets of problems, which motivates the approach taken in Section 4.4 below, where we use machine learning techniques to try to learn to choose in advance the most promising RLT version for a given instance.

A natural question given the results in Figure 4.1 is whether or not there are specific

subclasses of instances in the different data sets where a certain RLT version is noticeably dominant. A deeper analysis of the results shows that this is indeed the case, as presented in Figure 4.2. First, when looking at instances in DS with high density (larger than 0.5) we can see that the versions relying on SOCP constraints and `Mosek` as a solver, $\texttt{SOCP}^{M}$ and $\texttt{SOCP}^{M,B}$, are the best performing ones in around 70% of the instances; `RLT` goes down to around 10%, `SDP cuts` to around 5%, and $\texttt{SDP}^{2}$ and $\texttt{SDP}^{2,B}$ split the remaining 15%. Second, we selected from MINLPLib set all instances whose name contains the word "water", which are problems related to the design of water networks (Castro and Teles, 2013; Teles et al., 2012) and wastewater treatment systems (Castro et al., 2009, 2007). We have that in 15 out of the 28 resulting instances, the best option is one of the following SDP-based ones: $\texttt{SDP}^{1,B}$, $\texttt{SDP}^{2}$, and $\texttt{SDP}^{2,B}$. Again, the instances in which `RLT` or `SDP cuts` are the best option are less than 20%. The reasons behind the notoriously good performance of the versions based on second order cone constraints for high-density problems in DS and of those based on positive semidefinite constraints for "water"-related instances in MINLPLib are definitely worth studying more deeply.

Figure 4.2: Percentage of times that each version delivers the best performance, according to npace^{LB}, for two specific subclasses of problems.

Despite the results shown in Figure 4.1 and Figure 4.2, it is important to ensure that the variability is not spurious. For instance, it might be that all RLT versions perform very similarly to one another, which would turn most of the above discussion meaningless. In Table 4.1 we present a concise summary of the geometric mean of pace^{LB} for the complete set of instances and for the two special subclasses identified above. The first row measures the performance of `RLT` while the second row measures the performance of a hypothetical RLT version capable of choosing the best performing RLT version in each and every instance. The first column shows that, on aggregate on the whole set of instances, this hypothetical and optimal version would divide the pace by two (50.5% improvement),

a substantial improvement. This improvement is much more pronounced for high-density problems in DS, in which the pace gets divided by four (74.4% improvement), and even more so for "water"-related instances in MINLPLib where the pace becomes more than ten times smaller (92.5% improvement).

	All	DS high density	water
RLT	12.69	0.82	33994.03
Optimal version	6.28	0.21	2546.88
Improvement	50.5%	74.4%	92.5%

Table 4.1: Performance of optimal version according to pace^{LB}.

4.4 Machine Learning for Different Conic Constraints

In this section, we want to exploit the wide variability in the performance of the different RLT versions shown in the previous section to try to learn in advance which one should be chosen for a given instance. The goal is to design a machine learning procedure that can be trained based upon the different features of the instances and then choose the most promising RLT version when confronted with a new instance. The performance of the hypothetical "optimal version" in Table 4.1 represents an upper bound on the potential improvement attained by such a machine learning version.

We follow the framework in Chapter 3, where we use statistical learning techniques to improve the performance of `RAPOSa` by learning to choose between different branching rules. Here we use exactly the same learning framework: learning is jointly performed on all instances and we use the same features and the same KPI (npace^{LB}) for the instances. Yet, instead of trying to select the best branching rule for a given problem, the goal now is to select the best RLT version in our portfolio. The results we present below testify the robustness of the statistical learning approach in Chapter 3 since, with no customization at all, it delivers remarkable improvements in an unrelated learning task (branching rule versus conic approach).

Table 4.2 shows the remarkable improvement obtained with the machine learning version of RLT that chooses, for each given instance, the most promising version of the 10-version portfolio.[3] Considering all instances, this new version improves 32.9% with respect to `RLT`, being the upper bound for learning 50.5%. In instances with high densities

[3]The results we report do not distinguish between training and test data since, as discussed in Chapter 3, quantile random forests allow to evaluate the model on all instances using the out-of-bag predictions (no need to split between training and test for the reporting).

	All	DS high density	water
RLT	12.69	0.82	33994.03
ML-based version	8.51	0.25	14727.22
Optimal version	6.28	0.21	2546.88
Improvement after learning	32.9%	69.5%	56.7%
Optimal improvement	50.5%	74.4%	92.5%

Table 4.2: Performance of ML-based version according to pace^{LB}.

from DS data set, it improves a remarkable 69.5% out of the best achieved 74.4%, and in "water"-related instances in MINLPLib it improves 56.7% out of 92.5%. It is worth noting that the size of these improvements is comparable, and even superior, to those obtained in Chapter 3 when this learning scheme was introduced to learn to choose between branching rules, which shows the robustness of the proposed approach.

Figure 4.3 represents, side by side, how often each of the ten RLT versions is selected by the ML version and by the optimal one. We can see that the behaviour of the former mimics quite well that of the latter in the three sets of instances. Given that the learning is conducted jointly on the whole set of instances, the fact that the ML version adapts to the instances in DS, MINLPLib, and QPLIB, is reassuring about the quality of the learning process. In particular, the SOCP versions with `Mosek` are primarily chosen for DS data set and the SDP versions are mainly chosen for QPLIB data set, the one in which they are optimal more often. In Figure 4.4 we further explore the behaviour for the high-density problems in DS and for "water"-related instances in MINLPLib. We see that, again, the ML version mimics the patterns of the optimal version. The dominant version in DS, $\texttt{SOCP}^M$, is chosen in almost 75% of the instances. Similarly, the three SDP dominant versions in "water"-related instances are chosen almost 50% of the time. The dominant version in DS, $\texttt{SOCP}^M$, is chosen in almost 75% of the instances.

Figure 4.5 presents boxplots summarizing the performance according to npace^{LB} for all instances and for each individual data set. Recall that, by definition, values close to 1 mean that the corresponding version is almost the best one, whereas values close to 0 mean that its pace is much worse than the best one. We can see that, although `RLT` and `SDP cuts` versions are, on aggregate, the best ones in all three data sets, they are significantly outperformed by `ML Q-RF` in the three of them. The improvement is particularly noticeable in DS, where `ML Q-RF` is, by far, better than all underlying versions. This observation is further reinforced by the performance profiles (Dolan and Moré (2002)) represented in Figure 4.6. Again, `ML Q-RF` clearly outperforms all others, especially in DS instances.

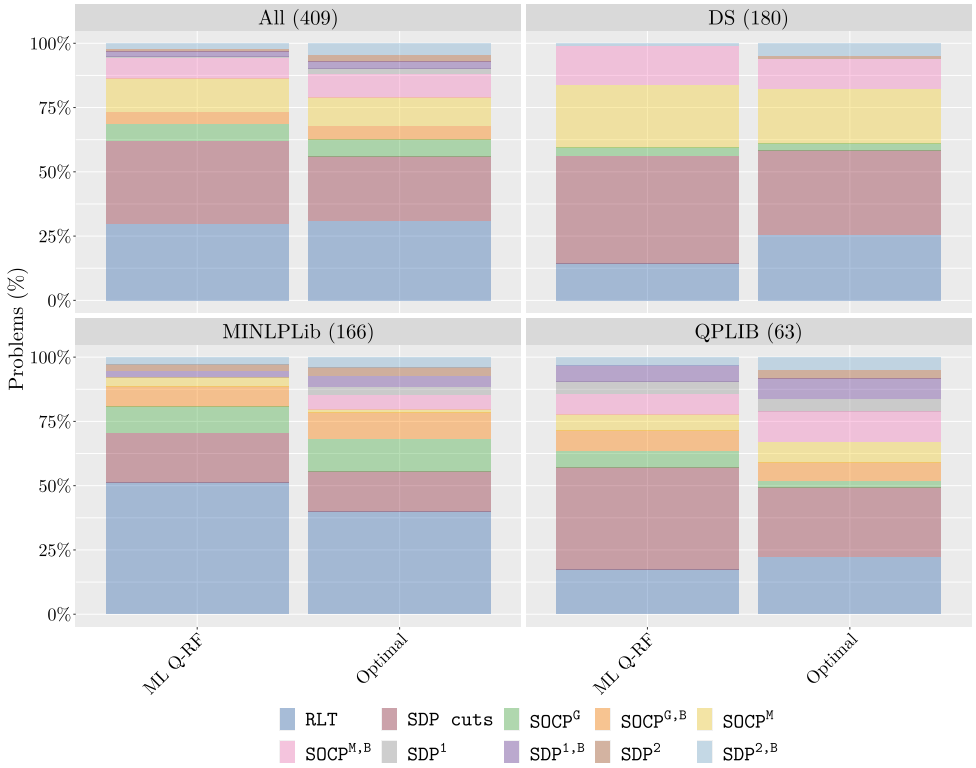


Figure 4.3: Percentage of times that each version is selected by `ML Q-RF` and the optimal version, according to npace^{LB}.

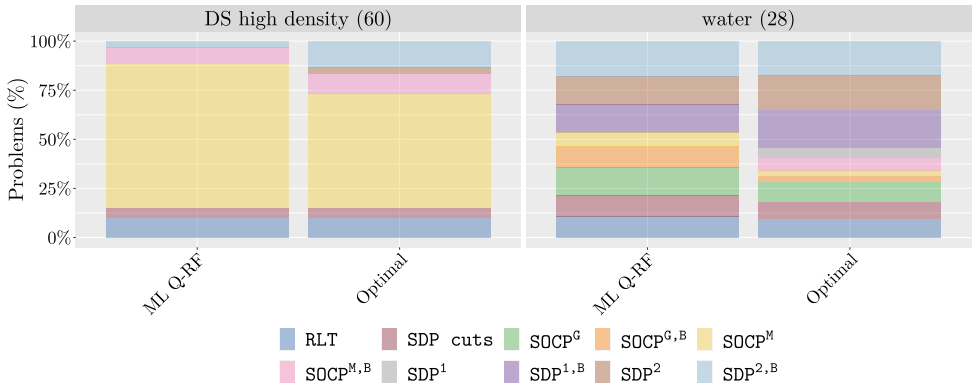


Figure 4.4: Percentage of times that each version is selected by `ML Q-RF` and the optimal version, according to npace^{LB}, for two specific subclasses of problem.

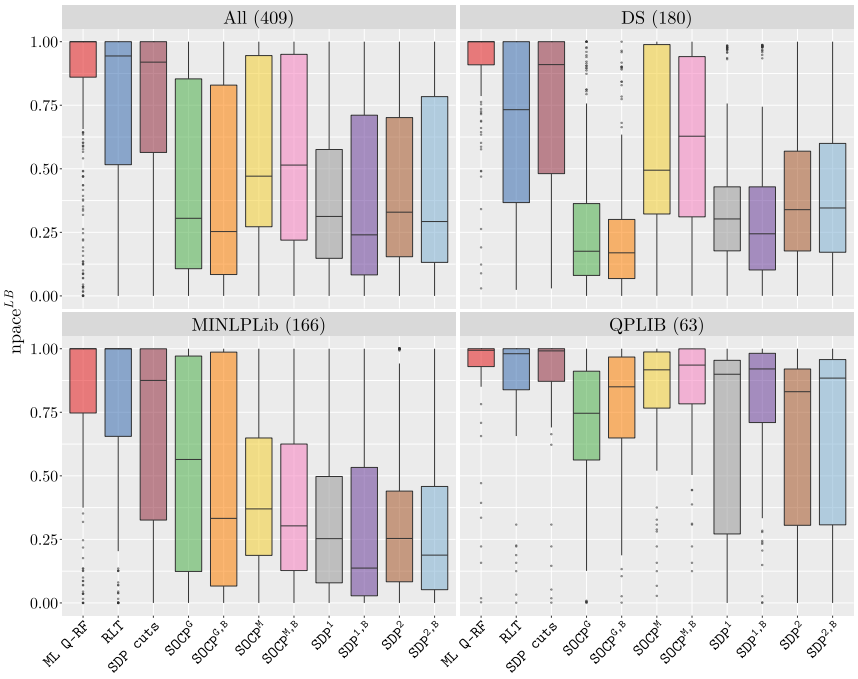


Figure 4.5: Boxplot of npace^{LB} for each version.

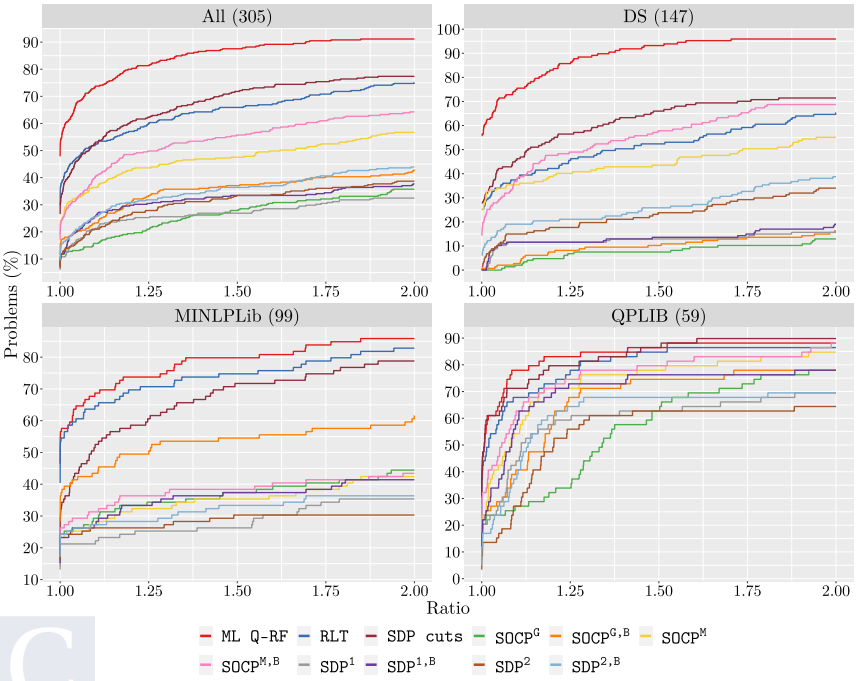


Figure 4.6: Performance profiles of pace^{LB} for each version.

4.5 Conclusions and Future Work

The main contribution of this chapter is to show that the solution to global optimality via branch-and-bound schemes of nonconvex optimization problems and, in particular, polynomial optimization ones, can benefit from tightening the underlying relaxations with conic constraints. We explore different families of such constraints, building upon either second-order cones or positive semidefiniteness. We also show that the potential of these conic constraints can be successfully exploited by embedding them into a learning framework. The main goal is to predict which is the most promising type of constraints to add to the RLT relaxation at each node of the underlying branch-and-bound algorithm when confronted with a new instance. The results in Section 3.4 show that the versions with SOCP/SDP conic constraints deliver consistently good results for instances in specific subclasses of problems: high-density problems in DS and of those based on positive semidefinite constraints for "water"-related instances in MINLPLib.

As a future step, one may wonder to what extent one might get an even superior performance if the learning analysis was further specialized for the current setting: for example, including features capturing some "conic" characteristics of the polynomial optimization problems and fine-tuning the regression techniques. Furthermore, an important direction for future research is to improve the understanding on the structure of these problems and the specificities that lead to the superior performance of SOCP and SDP constraints, respectively. Another direction is to investigate the number of SOCP and SDP constraints added at different nodes of the branch-and-bound tree. Additionally, we aim to extend this framework for various relaxations of polynomial optimization problems besides the RLT-based algorithm.

Bibliography

Baltean-Lugojan, R., P. Bonami, R. Misener, and A. Tramontani (2019): "Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks," Optimization Online.

Belotti, P., J. Lee, L. Liberti, F. Margot, and A. Wächter (2009): "Branching and bounds tightening techniques for non-convex MINLP," *Optimization Methods and Software*, 24, 597–634.

BENGIO, Y., A. LODI, AND A. PROUVOST (2021): "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 290, 405–421.

BONAMI, P., A. LODI, J. SCHWEIGER, AND A. TRAMONTANI (2019): "Solving quadratic programming by cutting planes," *SIAM Journal on Optimization*, 29, 1076–1105.

BUCHHEIM, C. AND A. WIEGELE (2013): "Semidefinite relaxations for non-convex quadratic mixed-integer programming," *Mathematical Programming*, 141, 435–452.

BURER, S. AND D. VANDENBUSSCHE (2008): "A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations," *Mathematical Programming*, 113, 259–282.

BURER, S. AND Y. YE (2020): "Exact semidefinite formulations for a class of (random and non-random) nonconvex quadratic programs," *Mathematical Programming*, 181, 1–17.

BUSSIECK, M. R., A. S. DRUD, AND A. MEERAUS (2003): "MINLPLib-A collection of test models for mixed-integer nonlinear programming," *INFORMS Journal on Computing*, 15, 114–119.

CASTRO, P. M., H. A. MATOS, AND A. Q. NOVAIS (2007): "An efficient heuristic procedure for the optimal design of wastewater treatment systems," *Resources, Conservation and Recycling*, 50, 158–185.

CASTRO, P. M. AND J. P. TELES (2013): "Comparison of global optimization algorithms for the design of water-using networks," *Computers & Chemical Engineering*, 52, 249–261.

CASTRO, P. M., J. P. TELES, AND A. Q. NOVAIS (2009): "Linear program-based algorithm for the optimal design of wastewater treatment systems," *Clean Technologies and Environmental Policy*, 11, 83–93.

DALKIRAN, E. AND H. D. SHERALI (2013): "Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality," *Journal of Global Optimization*, 57, 1147–1172.

——— (2016): "RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems," *Mathematical Programming Computation*, 8, 337–375.

DOLAN, E. D. AND J. J. MORÉ (2002): "Benchmarking optimization software with performance profiles," *Mathematical Programming*, 91, 201–213.

Elloumi, S. and A. Lambert (2019): "Global solution of non-convex quadratically constrained quadratic programs," *Optimization Methods and Software*, 34, 98–114.

FICO (2022): *FICO Xpress optimizer reference manual*.

Furini, F., E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. V. Sahinidis, S. Vigerske, and A. Wiegele (2018): "QPLIB: a library of quadratic programming instances," *Mathematical Programming Computation*, 11, 237–265.

Ghaddar, B., M. F. Anjos, and F. Liers (2011a): "A branch-and-cut algorithm based on semidefinite programming for the minimum k-partition problem," *Annals of Operations Research*, 188, 155–174.

Ghaddar, B. and R. A. Jabr (2019): "Power transmission network expansion planning: a semidefinite programming branch-and-bound approach," *European Journal of Operational Research*, 274, 837–844.

Ghaddar, B., J. C. Vera, and M. F. Anjos (2011b): "Second-order cone relaxations for binary quadratic polynomial programs," *SIAM Journal on Optimization*, 21, 391–414.

González-Rodríguez, B., R. Alvite-Pazó, S. Alvite-Pazó, B. Ghaddar, and J. González-Díaz (2022a): "Polynomial optimization: enhancing RLT relaxations with conic constraints," arXiv.

González-Rodríguez, B., J. Ossorio-Castillo, J. González-Díaz, Á. M. González-Rueda, D. R. Penas, and D. Rodríguez-Martínez (2022b): "Computational advances in polynomial optimization: RAPOSa, a freely available global solver," *Journal of Global Optimization*, In press.

Gurobi Optimization, LLC (2022): *Gurobi optimizer reference manual*.

IBM ILOG (2022): *CPLEX optimization studio CPLEX user's manual*.

Krislock, N., J. Malick, and F. Roupin (2017): "BiqCrunch: a semidefinite branch-and-bound method for solving binary quadratic problems," *ACM Transactions on Mathematical Software*, 43, 1–23.

Kuang, X., B. Ghaddar, J. Naoum-Sawaya, and L. F. Zuluaga (2017): "Alternative LP and SOCP hierarchies for ACOPF problems," *IEEE Transactions on Power Systems*, 32, 2828–2836.

LASSERRE, J. B. (2001): "Global optimization with polynomials and the problem of moments," *SIAM Journal on Optimization*, 11, 796–817.

LODI, A. AND G. ZARPELLON (2017): "On learning and branching: a survey," *TOP*, 25, 207–236.

MOSEK APS (2022): *Introducing the MOSEK optimization suite 9.3.20.*

PARRILO, P. A. (2003): "Semidefinite programming relaxations for semialgebraic problems," *Mathematical Programming*, 96, 293–320.

PICCIALLI, V., A. M. SUDOSO, AND A. WIEGELE (2022): "SOS-SDP: an exact solver for minimum sum-of-squares clustering," *INFORMS Journal on Computing*, 34, 2144–2162.

RENDL, F., G. RINALDI, AND A. WIEGELE (2010): "Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations," *Mathematical Programming*, 121, 307–335.

SHERALI, H. D., E. DALKIRAN, AND J. DESAI (2012): "Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts," *Computational Optimization and Applications*, 52, 483–506.

SHERALI, H. D. AND C. H. TUNCBILEK (1992): "A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique," *Journal of Global Optimization*, 2, 101–112.

SHOR, N. Z. (1987): "An approach to obtaining global extremums in polynomial mathematical programming problems," *Cybernetics*, 23, 695–700.

TELES, J. P., P. M. CASTRO, AND H. A. MATOS (2012): "Global optimization of water networks design using multiparametric disaggregation," *Computers & Chemical Engineering*, 40, 132–147.

Chapter 5

Enhancing Bound Tightening with Conic Constraints

5.1 Introduction

Bound tightening techniques are at the core of most global optimization algorithms for nonlinear problems. Previous literature has shown that its impact in the performance of optimization algorithms is considerable. For instance, in Chapter 2 we evaluate the performance of different enhancements of the RLT-based algorithm, and bound tightening techniques proved to be the ones that had the biggest impact.

Belotti et al. (2009) discuss different bound tightening techniques for solving nonlinear programming problems. They also discuss different strategies in order to decide at which points of the algorithm it is better to apply bound tightening. Moreover, they mention how to apply techniques of interval arithmetic to do feasibility-based bound tightening. Belotti et al. (2012) present a method for finding the limit point of the feasibility-based bound tightening sequence, because it might happen that running time might be infinite in some cases. Puranik and Sahinidis (2017) also analyse bound tightening techniques together with constraint propagation and interval arithmetic.

Belotti (2013) proposes improving bound tightening using convex combinations of the constraints of the problem. Gleixner et al. (2017) introduce some enhancements for feasibility-based bound tightening, using Lagrangian duality to obtain useful information from optimization-based bound tightening that can help to improve feasibility-based bound tightening. Ryoo and Sahinidis (1996) present range reduction techniques and some valid inequalities that can be used to reduce the size of the search space of the branch-and-bound tree. They rely on duality theory to systematically tight variable bounds (and also right-hand side of the problem constraints).

Furthermore, not only theoretical studies on how to improve boundary adjustment have been carried out, but also efforts have been made to adapt these techniques to some classes of problems with practical applications. Particularly, alternating current optimal power

flow (ACOPF) problems have drawn the interest of a number of researchers. For instance, Shchetinin et al. (2019) present three new methods for tightening bounds, focusing on being efficient for large-scale grids in the ACOPF problem. Chen et al. (2016) also demonstrate the effectiveness of bound tightening techniques in the ACOPF problem. Sundar et al. (2019) present an optimality-based bound tightening (OBBT) to improve the bounds in different parts of the network, using a convex quadratic relaxation of the ACOPF problem.

Therefore, the study of how to improve bound tightening techniques has been a line of research of considerable interest in recent years. Furthermore, we have already shown the relevance of second-order cone programming (SOCP) and semidefinite programming (SDP) in Section 4.1. To best of our knowledge, the impact of enhancing an OBBT by adding SOCP and SDP constraints has never been studied in the literature. In this chapter, we analyse the impact of using the conic constraints introduced in Chapter 4 to improve the OBBT explained in Section 2.5.5. Similarly to what happens in Chapter 4, the use of conic constraints tights the feasible region and therefore the bounds deduced by the OBBT are generally better. The main drawback is that this conic approach usually requires more time to compute the bounds because both SOCP and SDP problems are generally more difficult to solve than linear optimization problems.

The remainder of this chapter is organized as follows. In Section 5.2, we explain how we can enhance OBBT with SOCP and SDP constraints. In Section 5.3 we present some computational experiments to assess the impact on the performance when adding these conic constraints. Then, in Section 5.4 we show how this new approach can be further exploited withing a machine learning framework. Finally, we conclude in Section 5.5.

5.2 Conic Enhancements for OBBT

In recent literature, two main approaches of bound tightening have been discussed: optimality-based bound tightening (OBBT) and feasibility-based bound tightening (FBBT), as we explain in Section 2.2.1. In this chapter, we focus on how to improve OBBT. First of all, we explain what the standard OBBT consists of in the context of the RLT.

5.2.1 RLT-Based OBBT

In the context of the RLT, OBBT consists of minimizing and maximizing each variable of the problem over the feasible region of the linearized problem. This provides optimal upper and lower bounds for each variable before starting the branching process. Given a polynomial programming problem ($PP(\Omega)$) and its linear relaxation ($LP(\Omega)$), OBBT consists of solving the following two optimization problems for each variable x_j, with $j \in N$.

$$
\begin{array}{rll}
\text{minimize} & x_j & \\
\text{subject to} & [\phi_r(\boldsymbol{x})]_L \geq \beta_r, & r = 1, \ldots, R_1 \\
& [\phi_r(\boldsymbol{x})]_L = \beta_r, & r = R_1 + 1, \ldots, R \\
& [F_\delta(J_1, J_2)]_L \geq 0, & \forall J_1 \cup J_2 \subset N^\delta, \\
& & |J_1 \cup J_2| = \delta \\
& \boldsymbol{x} \in \Omega \subset \mathbb{R}^n. &
\end{array}
\tag{$OBBT^{\text{MIN}}_{x_j}$}
$$

$$
\begin{array}{rll}
\text{maximize} & x_j & \\
\text{subject to} & [\phi_r(\boldsymbol{x})]_L \geq \beta_r, & r = 1, \ldots, R_1 \\
& [\phi_r(\boldsymbol{x})]_L = \beta_r, & r = R_1 + 1, \ldots, R \\
& [F_\delta(J_1, J_2)]_L \geq 0, & \forall J_1 \cup J_2 \subset N^\delta, \\
& & |J_1 \cup J_2| = \delta \\
& \boldsymbol{x} \in \Omega \subset \mathbb{R}^n. &
\end{array}
\tag{$OBBT^{\text{MAX}}_{x_j}$}
$$

First, we build linear relaxation ($LP(\Omega)$). Then, we solve problems ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$) for each variable x_j of ($PP(\Omega)$). By doing this we obtain new lower and upper bounds for each variable x_j. This helps to reduce the search space in the branch-and-bound tree and might reduce the number of iterations of the RLT-based algorithm.

Now, if we tight more the feasible region of ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$), we might obtain better lower and upper bounds for the variables of ($PP(\Omega)$). In this chapter we study tightenings based on SOCP and SDP.

5.2.2 Enhancing OBBT with SOCP and SDP Constraints

The purpose of this section is to explain how to add SOCP and SDP constraints to both ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$). To do this, we follow the same approach from Chapter 4, building constraints based on J-sets in order to obtain sparser constraints and to preserve the size of the relaxations on which the OBBT relies.

On the one hand, we add SDP constraints to ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$) following the same two approaches as in Section 4.2.2. The first one consists of using vector $\boldsymbol{y}^1 = (x_j)_{j \in J}$ to define the $\boldsymbol{M}_L$ matrix and the constraint $\boldsymbol{M}_L \succcurlyeq 0$. The second one consists of using the vector $\boldsymbol{y}^2 = (1, (x_j)_{j \in J})$ instead of $\boldsymbol{y}^1$. Then, we add to ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$) the same SDP constraints we add to the RLT relaxations in Section 4.2.2.

On the other hand, we add SOCP constraints to ($OBBT^{\text{MIN}}_{x_j}$) and ($OBBT^{\text{MAX}}_{x_j}$) following the same approach from Section 4.2.3.

Moreover, in this context it makes more sense to use all the SDP/SOCP constraints in

each approach, regardless of whether they are binding or not. This is because, for each variable, problems ($OBBT_{x_j}^{\text{MIN}}$) and ($OBBT_{x_j}^{\text{MAX}}$) have different objective functions, as opposed to what happens when we add these constraints to the RLT relaxation at each node. Thus, what constraints are binding for one OBBT subproblem is probably not informative for the following ones. Therefore, we do not include in our numerical analysis the approach with binding constraints from Section 4.2.4.

5.3 Computational Results

5.3.1 Testing Environment

The testing environment is the same described in Section 4.3, using the same sets of instances: DS (Dalkiran and Sherali, 2016), MINLPLib (Bussieck et al., 2003), and QPLIB (Furini et al., 2018). Again, our analysis builds upon the global solver for polynomial optimization problems `RAPOSa` (González-Rodríguez et al., 2022) introduced in Chapter 2.[1] The only difference is that we run the computational experiments with a time limit of ten minutes, instead of one hour. We set this time limit because the main objective is to evaluate the impact of adding conic constraints to the OBBT at the root node (with a time limit of one hour, this effect might fade out). We run `RAPOSa` with five different versions as follows:

- `RLT`: run OBBT over the standard RLT relaxation (with J-sets).
- `SOCP`^{G}: SOCP constraints added to the RLT relaxation and solved with `Gurobi`.
- `SOCP`^{M}: SOCP constraints added to the RLT relaxation and solved with `Mosek`.
- `SDP`^{1}: SDP constraints added to the RLT relaxation following Approach 1 from Section 4.2.2 and solved with `Mosek`.
- `SDP`^{2}: SDP constraints added to the RLT relaxation following Approach 2 from Section 4.2.2 and solved with `Mosek`.

5.3.2 Numerical Results and Analysis

The numerical analysis of this section is devoted to show the potential of adding SOCP or SDP constraints to the RLT relaxations solved by the OBBT. In order to evaluate the performance, we use the same KPIs as in Chapter 3 and Chapter 4: pace^{LB} and npace^{LB}.

[1] As in Chapter 3, we use the baseline version from Chapter 2 including the enhancements from Section 2.5.1 and Section 2.5.3.

Figure 5.1 shows, for the different data sets, the percentage of instances in which each approach delivers the best performance, according to pace^{LB}. We can see that, quite consistently across the three data sets, SOCP and SDP constraints perform better than RLT-based OBBT in around 45% of the instances. This suggests the potential of these new approaches in order to improve the performance of the OBBT.

Figure 5.1 also shows a great richness in terms of which of these new approaches is better. We can see that in the three data sets, each of the new strategies delivers the best performance in a similar percentage of problems. This suggests that a statistical learning technique might be useful for learning which approach is better depending on the features of the problem. This is discussed in Section 5.4.

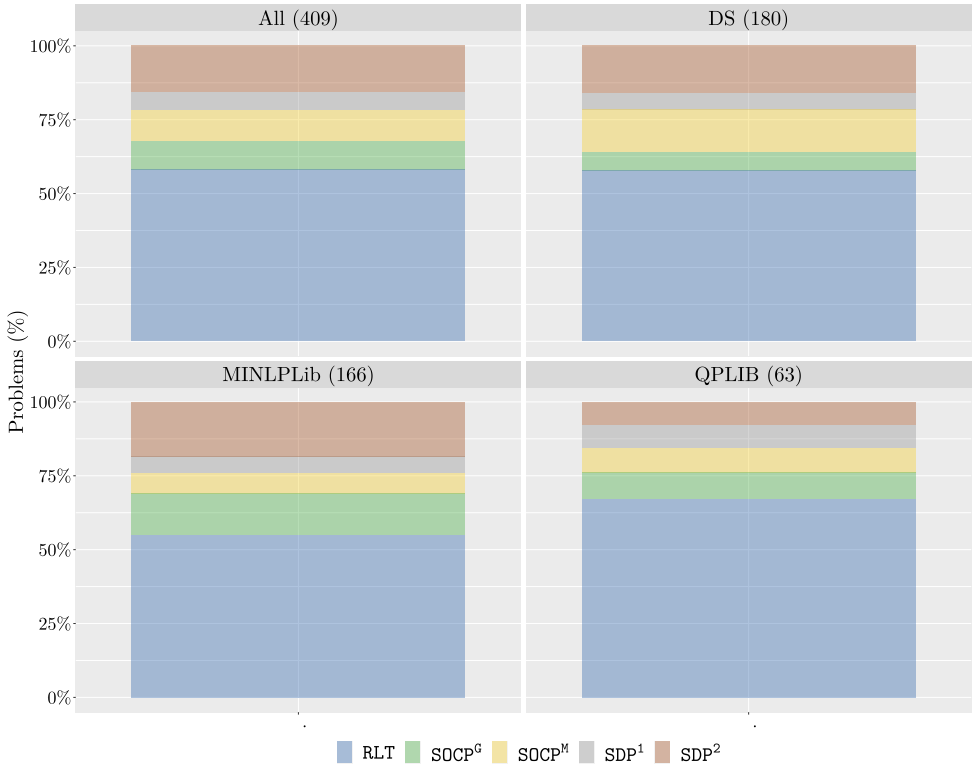


Figure 5.1: Percentage of times that each version delivers the best performance, according to npace^{LB}.

Furthermore, in Table 5.1 we see the room for improvement if we choose the best approach for each instance. The geometric mean of pace^{LB} over all instances can be improved by a 4.5%. Specifically, in MINLPLib, the room for improvement is 9.55% while in QPLIB there is almost no room for improvement. This suggests that characteristics of the instances can play an important role in determining which approach is preferable.

	All	DS	MINLPLib	QPLIB
Linear OBBT	4.84	0.100	346.44	50.68
Optimal version	4.62	0.096	313.37	50.64
Optimal improvement	4.5%	4.00%	9.55%	0.08%

Table 5.1: Performance of optimal version according to pace^{LB}.

5.4 Machine Learning for OBBT

Given the variability in performance of the different OBBT approaches, it seems natural to use statistical learning techniques to improve the performance of the RLT-based algorithm. The main objective of this section is to use a machine learning procedure to choose the most promising version of the OBBT, in terms of performance, using different features of the instances as input. To do this, we follow the same learning framework from Chapter 3 and Chapter 4. Again, the good results obtained in this context show the robustness of the learning framework introduced in Chapter 3.

In Table 5.2 we see that, considering all data sets together, the machine learning-based version improves a 1.45% out of 4.5% with respect to the RLT-based OBBT. This is not as remarkable as the improvement obtained in Chapter 3 and Chapter 4, but also the room for improvement is smaller. In fact, machine learning improves the performance by 32% of the maximum it could improve. Therefore, the machine learning framework shows again promising results. Furthermore, in MINLPLib, where we saw that there was more room for improvement, machine learning improves the performance in a 3.19% out of a 9.55%.

	All	DS	MINLPLib	QPLIB
Linear OBBT	4.84	0.100	346.44	50.68
ML-based version	4.77	0.099	335.39	50.69
Optimal version	4.62	0.096	313.37	50.64
Improvement after learning	1.45%	1.00%	3.19%	−0.02%
Optimal improvement	4.5%	4.00%	9.55%	0.08%

Table 5.2: Performance of ML-based version according to pace^{LB}.

Figure 5.2 represents, side by side, how often each of the ten RLT versions is selected by the machine learning version and by the optimal one. We can see that the machine learning approach maintains more or less the proportion of $\texttt{SOCP}^G$, $\texttt{SOCP}^M$, $\texttt{SDP}^1$, and $\texttt{SDP}^2$, with respect to the optimal ones. However, we see that `RLT` is chosen more often than it is optimal. The machine learning approach tends to choose `RLT` more often, because it is

more stable, in the sense that the other approaches, when they are not the best, return much worse results than RLT.

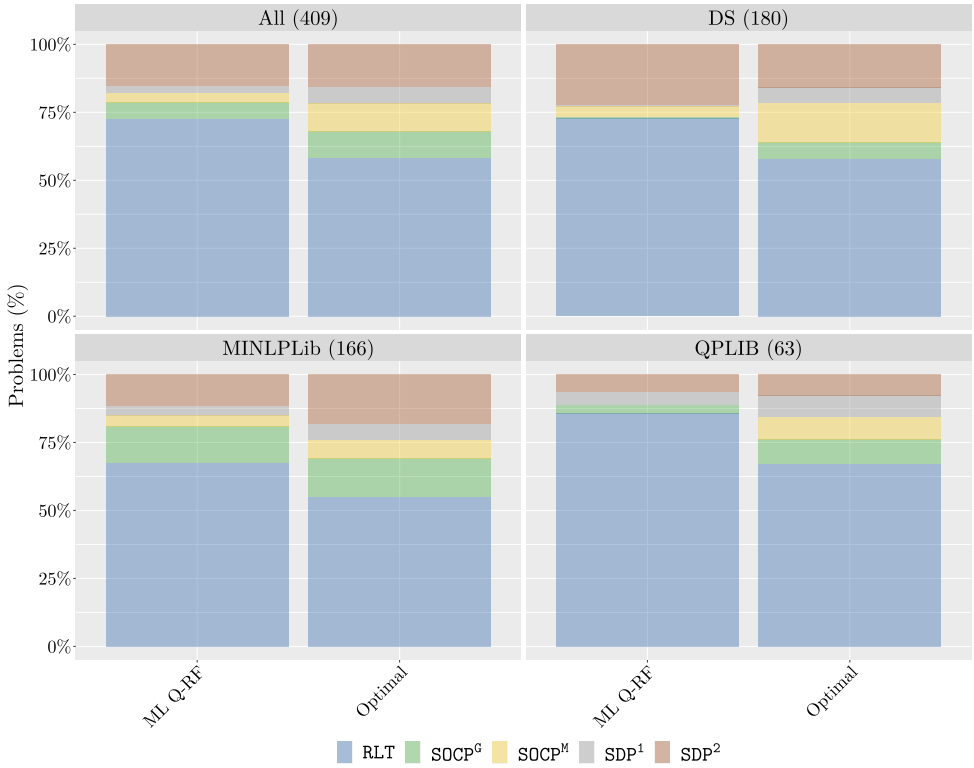


Figure 5.2: Percentage of times that each version is selected by ML Q-RF and the optimal version, according to npace^{LB}.

Figure 5.3 shows the performance profiles (Dolan and Moré, 2002) associated to the different approaches presented in this chapter and the machine learning. The main conclusion we can draw is that ML Q-RF delivers the best performance considering all instances together. Despite the fact that RLT delivers clearly better performance than SOCP^G, SOCP^M, SDP^1, and SDP^2, the main insight is that all these new approaches help to improve the performance of RAPOSa when we include them in a machine learning framework. We can see that this behaviour is very similar in both DS and MINLPLib data sets, while in QPLIB performance of ML Q-RF and RLT is very similar.

5.5 Conclusions and Future Work

In this chapter we have presented an enhanced OBBT using SDP and SOCP constraints. We have seen that these new approaches deliver the best performance in almost a half of the instances, showing the great potential of these new approaches. Moreover, an

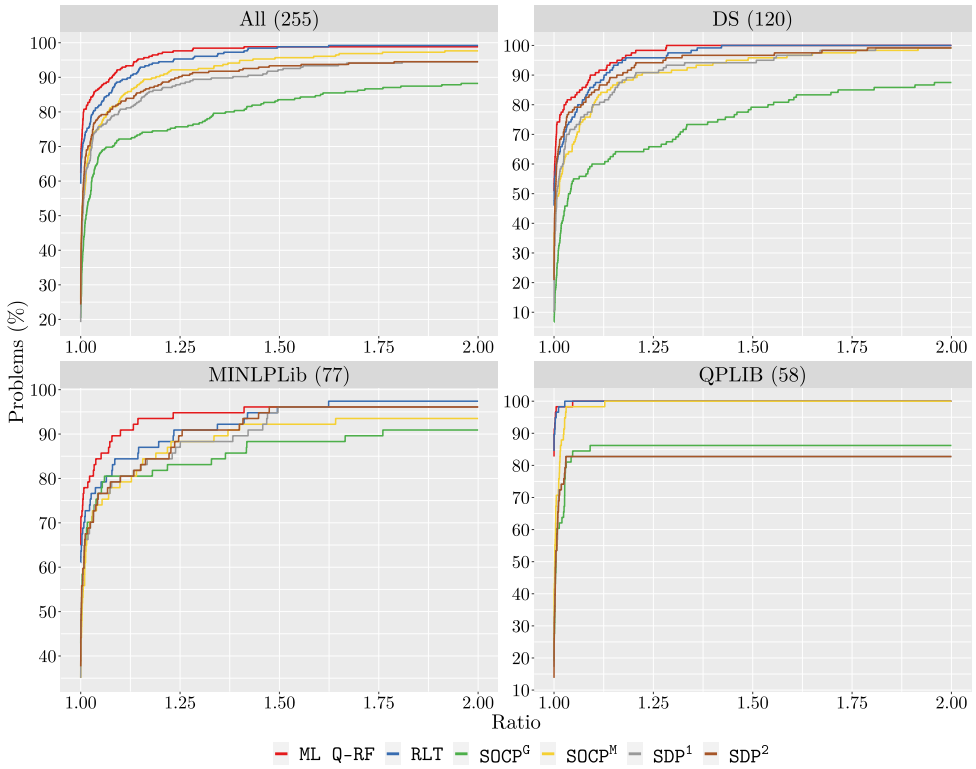


Figure 5.3: Performance profiles of pace^{LB} for each version.

out-of-the-box machine learning framework, with no specific tunning, results in a better performance than the RLT-based OBBT.

The next steps in this promising research direction would be the following ones:

- To adapt learning techniques to this context in order to obtain better results. This would involve, in the first place, the definition of new features that impact the performance of the OBBT. For example, measures such as the range or the density of each variable could give us an idea of how to prioritise the variables when applying the OBBT. It would be interesting to see what happens when we apply the OBBT only to the variables that we consider to have the most impact on solving the problem.

- To study different hybrid versions of the OBBT, being able to combine different approaches in the same instance. In line with the above, it might be interesting to apply an SDP/SOCP-based OBBT to the most important variables of the problem and an RLT-based OBBT to the less important ones. It would also be interesting to apply different OBBT approaches to one variable of the problem and, depending on the results obtained, decide which approach to use for the rest of the variables.

- To make a comprehensive analysis of the proposals already studied in the literature from Section 5.1 and integrate them into our learning framework.
- To review and propose strategies for deciding the value of the variable at which branching is done.
- To adapt bound tightening techniques to some classes of problems with relevance in practical applications, as the ACOPF problem.

Bibliography

Belotti, P. (2013): "Bound reduction using pairs of linear inequalities," *Journal of Global Optimization*, 56, 787–819.

Belotti, P., S. Cafieri, J. Lee, and L. Liberti (2012): "On feasibility-based bounds tightening," Optimization Online.

Belotti, P., J. Lee, L. Liberti, F. Margot, and A. Wächter (2009): "Branching and bounds tightening techniques for non-convex MINLP," *Optimization Methods and Software*, 24, 597–634.

Bussieck, M. R., A. S. Drud, and A. Meeraus (2003): "MINLPLib-A collection of test models for mixed-integer nonlinear programming," *INFORMS Journal on Computing*, 15, 114–119.

Chen, C., A. Atamtürk, and S. S. Oren (2016): "Bound tightening for the alternating current optimal power flow problem," *IEEE Transactions on Power Systems*, 31, 3729–3736.

Dalkiran, E. and H. D. Sherali (2016): "RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems," *Mathematical Programming Computation*, 8, 337–375.

Dolan, E. D. and J. J. Moré (2002): "Benchmarking optimization software with performance profiles," *Mathematical Programming*, 91, 201–213.

Furini, F., E. Traversi, P. Belotti, A. Frangioni, A. Gleixner, N. Gould, L. Liberti, A. Lodi, R. Misener, H. Mittelmann, N. V. Sahinidis, S. Vigerske,

and A. Wiegele (2018): "QPLIB: a library of quadratic programming instances," *Mathematical Programming Computation*, 11, 237–265.

Gleixner, A. M., T. Berthold, B. Müller, and S. Weltge (2017): "Three enhancements for optimization-based bound tightening," *Journal of Global Optimization*, 67, 731–757.

González-Rodríguez, B., J. Ossorio-Castillo, J. González-Díaz, Á. M. González-Rueda, D. R. Penas, and D. Rodríguez-Martínez (2022): "Computational advances in polynomial optimization: RAPOSa, a freely available global solver," *Journal of Global Optimization*, In press.

Puranik, Y. and N. V. Sahinidis (2017): "Domain reduction techniques for global NLP and MINLP optimization," *Constraints*, 22, 338–376.

Ryoo, H. S. and N. V. Sahinidis (1996): "A branch-and-reduce approach to global optimizations," *Journal of Global Optimization*, 8, 107–138.

Shchetinin, D., T. T. De Rubira, and G. Hug (2019): "Efficient bound tightening techniques for convex relaxations of AC optimal power flow," *IEEE Transactions on Power Systems*, 34, 3848–3857.

Sundar, K., H. Nagarajan, S. Misra, M. Lu, C. Coffrin, and R. Bent (2019): "Optimization-based bound tightening using a strengthened QC-relaxation of the optimal power flow problem," arXiv.

Chapter 6

Global Optimization for Bilevel Portfolio Design

6.1 Introduction

This chapter is based on González-Díaz et al. (2021) and shows an application of polynomial optimization. Mathematical optimization problems are pervasive in the fields of economics and management science, and the importance of developing realistic models to delve into the understanding of complex economic settings has long been recognized. In particular, bilevel optimization models have attracted a lot of attention since the pioneering work by von Stackelberg (1934). However, realistic models often result in difficult optimization problems and, thus, a compromise is required between realism and solvability of the model. Quite often the analyst is confronted with a nonlinear and nonconvex optimization problem, in which finding globally optimal solutions may be an extremely challenging task.

One of the contributions of this work is to illustrate how proper modelling skills may allow to efficiently solve complex optimization problems, enabling the development of qualitative and quantitative economic analysis in problems that, otherwise, would be hard to tackle. In order to do so, we formally study two novel bilevel portfolio design problems and show how one can rely on well-established approaches to reformulate them as single-level problems that can be fully solved with state-of-the-art optimization software.

Historically, the main criterion to design an optimal portfolio was to find the configuration of assets that generated the highest expected return. However, this perspective changed in 1952, when Harry Markowitz introduced a new variable along with the expected return: the risk of each portfolio (Markowitz, 1952). Thereafter, analysts began to incorporate a risk-return trade-off in their models.

The model proposed by Markowitz only focuses on finding an optimal portfolio from the investor's point of view. The literature is plenty of papers with similar approaches, as for instance Benati (2003, 2015), Castro et al. (2011), Kolm et al. (2014), Mansini et al. (2014), Cesarone (2020), Cesarone et al. (2020), Perrin and Roncalli (2020) and Puerto et al.

(2020), to mention a few. However, real markets are generally more complex, since there is another decision-maker who can set fees on the transactions of the securities to profit from anticipating the rational behaviour of investors. Transaction costs, those incurred by the investors when buying and selling assets on financial markets, have been widely studied in papers such as Kellerer et al. (2000), Baule (2010), Baumann and Trautmann (2013), Woodside-Oriakhi et al. (2013), Mansini et al. (2014), and Valle et al. (2014), among others. In this work we study these situations in which the investor, when deciding his optimal portfolio, has to consider the transaction fee to pay to an agent: the broker-dealer, hereafter, the broker. The broker makes decisions regarding fees associated to the assets, trying to maximize the resulting profit.[1]

One of the main contributions of this work is methodological: introducing a single-period hierarchical portfolio problem, with continuous choices, that accounts for decisions on transaction fees. The broker fixes these fees while the investor chooses his portfolio. The timing of these two decisions is crucial and bilevel optimization is necessary to understand the impact of different hierarchical structures. We discuss different models representing this situation, which arise depending on the order in which choices are made. The main model, and arguably the most realistic one, is the B-L model (Broker as Leader) in which, first, transaction fees are determined by the broker and then, after observing them, the investor chooses his portfolio. Two additional models are considered and can be seen as instrumental benchmarks to gain understanding on the B-L model: i) the I-L model (Investor as Leader) in which the broker chooses the transaction fees after observing the portfolio chosen by the investor and ii) the SW model, a *simultaneous*-choice model in which the goal is to maximize social welfare.

Importantly, regarding broker's fees, we allow for quite general market structures. The usual functioning of these markets, where the broker selects a fee, common to all assets, to be charged on top of the fees fixed by the stock market, is just a particular case. The extra generality enables the study of natural departures from this baseline setting and the analysis of the impact of the novel strategic aspects they may introduce. For instance, in our analysis we let the broker associate different fees to different assets, giving him extra freedom to tailor the fees to the attractiveness of the different assets. These models and the associated solution techniques may be seen as a first building block in the evaluation of this new competitive situation. An important challenge for future research is to enhance the models so that they can help to understand the effects on long-term dynamics of the

[1] This pricing aspect has been modelled as a bilevel problem in the literature in many different applications. See, for instance, Shioda et al. (2011), Labbé and Violin (2016), Grimm et al. (2019), Maravillo et al. (2020), Qiu et al. (2020) and the references therein.

ensuing competition between different brokers.

In Markowitz's seminal paper, the risk measure under consideration was the variance, which, despite being a sound measure of dispersion, is nowadays known to have important drawbacks as a risk measure. Since then, many different risk measures have been introduced and analysed, such as Gini's Mean Difference (Bey and Howe, 1984), Mean Absolute Deviation (Konno and Yamazaki, 1991), Value at Risk (Stambaugh, 1996), and Conditional Value at Risk (Rockafellar and Uryasev, 2000), to name a few. At the same time, a theoretical body around risk measures was developed and the notion of *coherency*, introduced in Artzner et al. (1997, 1999), was identified as a natural requirement. A coherent risk measure must satisfy the properties of monotonicity, sub-additivity, homogeneity, and translational invariance.[2]

Coherency is one of the main reasons to have developed our analysis for the Conditional Value at Risk (CVaR), also known as Expected Shortfall, which is the weighted average of the extreme losses in the tail of the distribution of the returns. The other main reason is that CVaR falls into the set of risk measures whose optimization can be formulated as a linear optimization problem, as shown in Rockafellar and Uryasev (2000). Reviews of other LP solvable risk measures can be found in Mansini et al. (2003, 2014).

A first approach to the type of bilevel models dealt with in this work can be seen in Leal et al. (2020). Nevertheless, there are a number of differences with respect to our approach, the main one being that transaction fees are assumed to be chosen from a discrete set in Leal et al. (2020), and solution approaches revolve around mixed-integer linear programming reformulations and Bender's decomposition techniques (Benders, 1962). Our analysis allows for continuous transaction fees and, hence, the resulting optimization problems are of a completely different nature: nonlinear and nonconvex polynomial optimization problems.[3] From the computational point of view, finding the global optimum of these problems is known to be an $\mathcal{NP}$-hard problem.[4] However, moderate size problems can be handled by state-of-the-art optimization techniques and solvers. Our computational analysis builds on two such solvers: `RAPOSa` (González-Rodríguez et al., 2022), the solver introduced in Chapter 2, and `BARON` (Tawarmalani and Sahinidis, 2005; Sahinidis, 2021). In addition, we have also addressed the issue of scalability of our solution techniques as compared with other solution methods for larger size instances. To this end, we have studied solution techniques built in different local solvers: `Ipopt` (Wächter and Biegler, 2006), `Knitro` (Byrd et al., 2006) and `MINOS` (Murtagh and Saunders, 1978). The comparative analysis of the quality

[2] Refer to Rockafellar (2014) for a recent review on the topic.

[3] In this context, by a polynomial optimization problem we refer to optimization problems in which both the objective function and the constraints are given by polynomials. Refer, for instance, to Lasserre (2015).

[4] A class of problems for which no polynomial-time algorithm to find the global solution is known.

of solutions found and running times is reported in Appendix 6.A.1. Another important difference with respect to Leal et al. (2020) is that we also model, and numerically study, situations in which there are multiple followers in the bilevel problems, which leads to richer economic interactions and potential for additional economic insights. One of the main findings in the numerical analysis is that the outcomes of the B-L model Pareto dominate those of the I-L model and, further, B-L outcomes are Pareto efficient.[5]

The organization of this chapter is as follows. First, in Section 6.2 we describe the baseline individual optimization problems for the investor and the broker. Second, in Section 6.3 we present the joint optimization problems: the two hierarchical Stackelberg models and the *simultaneous*-choice model; further, the solution techniques for these models are also developed. Then, in Section 6.4 we present a case study based on data from the Dow Jones Index and provide some economic insights.

6.2 Baseline Models for Broker-Investor Interactions

The portfolio optimization problem considered in this chapter is based on a single-period model of investment and incorporates a pricing aspect on the transaction costs. Borrowing from Leal et al. (2020), we assume the existence of two types of decision-makers: investors and brokers, with a hierarchical decision structure.

The investor faces the classic problem of allocating his capital among various financial securities, each of which will generate some uncertain returns whose random distribution is assumed to be known by the investor. Building upon the mean-variance approach initiated in Markowitz (1952), we assume that the goal of the investor is to minimize his risk subject to achieving a given expected return. On the other hand, the broker must determine the transaction costs associated with the different securities, with the goal of maximizing his profit. Thus, the decision variables of the investor are the proportions of his capital to invest in each security and those of the broker are the fees on the different securities. These decisions determine the amount paid by the investor to the broker and the risk and expected return of the investor. Note that the net return for the investor is the result of subtracting, for each security, the broker's fee from the security's return.

Formally, the main element of all our models is the set of securities S. The rate of return of each security $j \in S$ is uncertain and we model it through a random variable R_j. Following the standard approach in the field of decision making under uncertainty, we

[5] An outcome is Pareto efficient if no outcome makes one agent better off without hurting the other one. An outcome that is not Pareto efficient is Pareto dominated. Refer to Pareto (1971) for an English translation of Pareto's pioneering work in the late eighties.

assume that these random variables take values on a finite set T of scenarios.[6] Given a security $j \in S$ and a scenario $t \in T$, r_{jt} denotes the realization of the rate of return R_j in scenario t. Each scenario $t \in T$ has associated probability π_t and $\sum_{t \in T} \pi_t = 1$.

The broker has to choose the transaction costs associated with the securities in S, which we call prices and denote by $\boldsymbol{p}$. Thus, for each $j \in S$, $p_j \in [0, 1]$ represents the proportion of the amount invested in security j that must be paid to the broker. It is natural to consider that there are some limits on the prices that can be chosen by the broker, so we assume that there is a set of feasible prices P, which, moreover, is taken to be a bounded polyhedron. In actual cases, the associated constraints may be imposed by market regulations, by the board of the broker's company given some market analysis, or by a combination of both.

The investor has to choose a portfolio $\boldsymbol{x}$ so that, for each $j \in S$, $x_j \in [0, 1]$ represents the weight of security j in the portfolio. We assume that all capital is invested, so $\sum_{j \in S} x_j = 1$; if needed, a safe security with zero rate of return and zero price can be added to set S to represent the proportion of money that is not invested. Each portfolio $\boldsymbol{x}$ defines a random variable $R_{\boldsymbol{x}} = \sum_{j \in S} R_j x_j$ that represents the rate of return of the portfolio.

In some of the models we discuss we allow for multiple investors, differing only in their degrees of risk aversion. Importantly, throughout our analysis, although each of these investors may be thought of as a single agent, he can also represent a discrete or continuum set of agents, all of them with the same preferences. At optimality, since all the agents with the same degree of risk aversion have the same preferences, we can assume that they choose the same portfolio and, thus, in our mathematical formulation we "aggregate" them in a single investor.

Broker's Problem

The goal of the broker is to maximize his profit so, given a portfolio $\boldsymbol{x}$, this can be achieved by solving the linear optimization problem

$$\underset{\boldsymbol{p}}{\text{maximize}} \quad \sum_{j \in S} p_j x_j \tag{B$^{\text{P}}$}$$

$$\text{subject to} \quad \boldsymbol{p} \in P. \tag{B$^{\text{P}}$.a}$$

The main difference between (B$^{\text{P}}$) and (PricP) in Leal et al. (2020) relies on the continuous character of prices. This leads to a totally different family of optimization problems in terms of properties and solution techniques: the model in Leal et al. (2020) is combinatorial

[6] See, for instance, Chapter 1 in Birge and Louveaux (2011) and Chapter 4 in King and Wallace (2012).

whereas (B^P) is continuous.

Investor's Problem

We model the investor's optimization problem as one in which he wants to reduce the risk associated with his portfolio while attaining a certain expected return. As we have already discussed in Section 6.1, the risk measure under consideration is the Conditional Value at Risk (Rockafellar and Uryasev, 2000, 2002), which is the opposite of the expected return on the portfolio when considering only the worst $\alpha\%$ of cases. The smaller α is the more concerned is the investor with the lower tail of the distribution, *i.e.*, the more risk averse. Given a level α and a discrete random variable Y defined on the set of scenarios T, we denote the corresponding Conditional Value at Risk by $\text{CVaR}_\alpha(Y)$; the larger is its value, the riskier is the random variable Y.

Given a portfolio $\boldsymbol{x}$ and a price profile $\boldsymbol{p}$, let Y denote the random variable that, for each $t \in T$, gives the net rate of return for the investor: $y_t = \sum_{j\in S}(r_{jt}x_j - p_j x_j)$. Suppose that we have an investor with risk level α and minimum required expected profit given by $E_{\min}$. Then, given $\boldsymbol{p}$, he wants to solve the following optimization problem:

$$\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{y}}{\text{minimize}} \quad & \text{CVaR}_\alpha(Y) && && (\text{I}_0^{\text{P}})\\
\text{subject to} \quad & y_t = \sum_{j\in S}(r_{jt}x_j - p_j x_j), && t \in T && (\text{I}_0^{\text{P}}.\text{a})\\
& \sum_{t\in T} \pi_t y_t \geq E_{\min} && && (\text{I}_0^{\text{P}}.\text{b})\\
& \sum_{j\in S} x_j = 1 && && (\text{I}_0^{\text{P}}.\text{c})\\
& x_j \geq 0, && j \in S. && (\text{I}_0^{\text{P}}.\text{d})
\end{aligned}$$

An important property of CVaR is that its computation is equivalent to the solution of a linear programming problem. Following Rockafellar and Uryasev (2000, 2002), we have that, for $\alpha \in (0,1)$, $\text{CVaR}_\alpha(Y)$ can be computed as

$$\begin{aligned}
\underset{\eta,\boldsymbol{d}}{\text{minimize}} \quad & -\eta + \frac{1}{\alpha}\sum_{t\in T} d_t\pi_t && && (\text{CVaR}^{\text{P}})\\
\text{subject to} \quad & d_t \geq \eta - y_t, && t \in T && (\text{CVaR}^{\text{P}}.\text{a})\\
& d_t \geq 0, && t \in T. && (\text{CVaR}^{\text{P}}.\text{b})
\end{aligned}$$

The combination of the elements in problems (I_0^{P}) and (CVaR^{P}) leads to the full

formulation of the investor's problem:

$$\begin{aligned}
\underset{\boldsymbol{x},\boldsymbol{y},\eta,\boldsymbol{d}}{\text{minimize}} \quad & -\eta + \frac{1}{\alpha}\sum_{t\in T} d_t\pi_t && && (\mathrm{I}^{\mathrm{P}})\\
\text{subject to} \quad & y_t = \sum_{j\in S}(r_{jt}x_j - p_j x_j), && t\in T && (\mathrm{I}_0^{\mathrm{P}}.\mathrm{a})\\
& \sum_{t\in T}\pi_t y_t \geq E_{\min} && && (\mathrm{I}_0^{\mathrm{P}}.\mathrm{b})\\
& \sum_{j\in S} x_j = 1 && && (\mathrm{I}_0^{\mathrm{P}}.\mathrm{c})\\
& x_j \geq 0, && j\in S && (\mathrm{I}_0^{\mathrm{P}}.\mathrm{d})\\
& d_t \geq \eta - y_t, && t\in T && (\mathrm{CVaR}^{\mathrm{P}}.\mathrm{a})\\
& d_t \geq 0, && t\in T. && (\mathrm{CVaR}^{\mathrm{P}}.\mathrm{b})
\end{aligned}$$

Observe that ($\mathrm{I}_0^{\mathrm{P}}$.a) gives the expected return in each scenario, accounting for the transaction costs and ($\mathrm{I}_0^{\mathrm{P}}$.b) ensures the minimum expected return. Constraints ($\mathrm{I}_0^{\mathrm{P}}$.c) and ($\mathrm{I}_0^{\mathrm{P}}$.d) define the portfolio. Finally, the objective function and constraints ($\mathrm{CVaR}^{\mathrm{P}}$.a) and ($\mathrm{CVaR}^{\mathrm{P}}$.b) come from the optimization problem to compute $\mathrm{CVaR}_\alpha(Y)$. Different choices of parameters α and $E_{\min}$ allow one to model different investor risk profiles. Note that again we have a linear optimization problem.

Dual Problems

Now that we have formally defined both the broker and the investor problems, the next step is to put them together in a joint optimization problem. We do so in the next section by defining two bilevel problems, depending on who is the leader, the broker or the investor. These problems are then reformulated as single level problems by relying on strong duality in linear programming.[7] In order to do so, we need the formulations of the duals of problems (B^{P}) and (I^{P}), which we present below.

To formulate the broker's dual problem we can assume, without loss of generality, that the polyhedron P can be defined as $\boldsymbol{A}\boldsymbol{p} \leq \boldsymbol{b}$, with these constraints being indexed over the set P^{cons}. Then, the dual variables are $v_s \geq 0$ with $s \in P^{\text{cons}}$ and the dual constraints are $\boldsymbol{v}^\intercal\boldsymbol{A} - \boldsymbol{x}^\intercal = \boldsymbol{0}$, leading to the following dual problem.

[7] Refer, for instance, to Chapter 6 in Bazaraa et al. (2009).

$$\underset{\boldsymbol{v}}{\text{minimize}} \quad \sum_{s \in P^{\text{cons}}} b_s v_s \tag{B$^{\text{D}}$}$$

$$\text{subject to} \quad \boldsymbol{v}^{\intercal} \boldsymbol{A} - \boldsymbol{x}^{\intercal} = \boldsymbol{0} \tag{B$^{\text{D}}$.a}$$

$$v_s \geq 0, \qquad s \in P^{\text{cons}}. \tag{B$^{\text{D}}$.b}$$

The formulation of the investor's dual problem is more involved. The dual variables are

- $\boldsymbol{\delta}$, such that $\delta_t \in \mathbb{R}$ for each $t \in T$, associated with constraints (I_0^{P}.a).
- $\mu \leq 0$, associated with constraint (I_0^{P}.b).
- $\beta \in \mathbb{R}$, associated with constraint (I_0^{P}.c).
- $\boldsymbol{\gamma}$, such that $\gamma_t \leq 0$ for each $t \in T$, associated with constraints (CVaR^{P}.a).

Then, we get the dual problem

$$\underset{\boldsymbol{\delta},\mu,\beta,\boldsymbol{\gamma}}{\text{maximize}} \quad -\beta - E_{\min}\mu \tag{I$^{\text{D}}$}$$

$$\text{subject to} \quad \beta + \sum_{t \in T} (r_{jt} - p_j)\delta_t \geq 0, \quad j \in S \tag{I$^{\text{D}}$.a}$$

$$\sum_{t \in T} \gamma_t = -1 \tag{I$^{\text{D}}$.b}$$

$$\gamma_t \geq -\frac{\pi_t}{\alpha}, \qquad t \in T \tag{I$^{\text{D}}$.c}$$

$$\gamma_t - \delta_t + \pi_t \mu = 0, \qquad t \in T \tag{I$^{\text{D}}$.d}$$

$$\gamma_t \leq 0, \qquad t \in T \tag{I$^{\text{D}}$.e}$$

$$\mu \leq 0. \tag{I$^{\text{D}}$.f}$$

6.3 Joint Optimization Models for Broker-Investor Interactions

In this section we present different models for the interaction between brokers and investors. First, we consider bilevel optimization models in which either i) we have a leading broker with multiple investors acting after prices have been set or ii) we have a leading investor with multiple brokers setting prices once the investment made on each of them has been chosen. Although enriching the above models to include multiple leaders might lead to additional insights, the resulting optimization problems are substantially more complex to

solve and go beyond the scope of this work. On top of the aforementioned bilevel models, we also discuss a social welfare maximization problem, in which we look at the set of Pareto efficient combinations of prices and portfolios. As we already argued in Section 6.1, the model with a leading broker is the main object of interest in our analysis and the other two models are instrumental for comparison. In particular, the Pareto frontier represents a good benchmark to assess the quality of the outcomes obtained in the bilevel models.

6.3.1 B-L Model: One Broker-Leader, Several Investors-Followers

This model considers the situation where the broker makes his decision first and then a set M of investors, with possibly different degrees of risk aversion, choose their portfolios.[8] More precisely, once the leader's decision $\boldsymbol{p}$ is revealed, each investor $i \in M$ chooses his portfolio $\boldsymbol{x^i}$. Investors may have different risk levels, α^i, and minimum expected returns, $E^i_{\min}$. The associated bilevel optimization problem is

$$\begin{aligned}
\underset{\boldsymbol{p},(\boldsymbol{x^i},\boldsymbol{y^i},\eta^i,\boldsymbol{d^i})_{i\in M}}{\text{maximize}} \quad & \sum_{i\in M}\sum_{j\in S} p_j x^i_j && && \text{(B-L-BILEVEL)}\\
\text{subject to} \quad & \boldsymbol{p} \in P && && (\mathrm{B^P}.\mathrm{a})\\
& \forall i \in M, \quad \boldsymbol{x^i} \in \underset{\boldsymbol{x^i},\boldsymbol{y^i},\eta^i,\boldsymbol{d^i}}{\arg\min} \quad & -\eta^i + \frac{1}{\alpha^i}\sum_{t\in T}\pi_t d^i_t && && (\mathrm{I^P})\\
& \qquad\qquad \text{subject to} \quad & y^i_t = \sum_{j\in S}(r_{jt}x^i_j - p_j x^i_j), && t \in T && (\mathrm{I^P_0}.\mathrm{a})\\
& & \sum_{t\in T}\pi_t y^i_t \geq E^i_{\min} && && (\mathrm{I^P_0}.\mathrm{b})\\
& & \sum_{j\in S} x^i_j = 1 && && (\mathrm{I^P_0}.\mathrm{c})\\
& & x^i_j \geq 0, && j \in S && (\mathrm{I^P_0}.\mathrm{d})\\
& & d^i_t \geq \eta^i - y^i_t, && t \in T && (\mathrm{CVaR^P}.\mathrm{a})\\
& & d^i_t \geq 0, && t \in T. && (\mathrm{CVaR^P}.\mathrm{b})
\end{aligned}$$

The broker's leading problem has nested followers' subproblems in which investors choose their portfolios. Once the prices are set by the broker, the followers' subproblems are continuous linear programs. Hence, applying strong duality, we can obtain a single-level reformulation in which each subproblem is replaced by the feasibility constraints of the primal, ($\mathrm{I^P}$), the feasibility constraints of the dual, ($\mathrm{I^D}$), and the strong duality constraints,

[8] Recall that each investor type may itself be thought of as a set of investors sharing the same preferences.

(B-L-SD), namely, the equality between the objective functions of (I^P) and (I^D):

$$\underset{\boldsymbol{p},(\boldsymbol{x^i},\boldsymbol{y^i},\eta^i,\boldsymbol{d^i},\boldsymbol{\delta^i},\mu^i,\beta^i,\boldsymbol{\gamma^i})_{i\in M}}{\text{maximize}} \quad \sum_{i\in M}\sum_{j\in S} p_j x_j^i \tag{B-L}$$

$$\text{subject to} \quad \boldsymbol{p} \in P \tag{B^P.a}$$

$$y_t^i = \sum_{j\in S}(r_{jt}x_j^i - p_j x_j^i), \qquad t\in T, \quad i\in M \tag{I_0^P.a}$$

$$\sum_{t\in T}\pi_t y_t^i \geq E_{\min}^i, \qquad i\in M \tag{I_0^P.b}$$

$$\sum_{j\in S} x_j^i = 1, \qquad i\in M \tag{I_0^P.c}$$

$$x_j^i \geq 0, \qquad j\in S, \quad i\in M \tag{I_0^P.d}$$

$$d_t^i \geq \eta^i - y_t^i, \qquad t\in T, \quad i\in M \tag{$CVaR^P$.a}$$

$$d_t^i \geq 0, \qquad t\in T, \quad i\in M \tag{$CVaR^P$.b}$$

$$\beta^i + \sum_{t\in T}(r_{jt}-p_j)\delta_t^i \geq 0, \qquad j\in S, \quad i\in M \tag{I^D.a}$$

$$\sum_{t\in T}\gamma_t^i = -1, \qquad i\in M \tag{I^D.b}$$

$$\gamma_t^i \geq -\frac{\pi_t}{\alpha^i}, \qquad t\in T, \quad i\in M \tag{I^D.c}$$

$$\gamma_t^i - \delta_t^i + \pi_t\mu^i = 0, \qquad t\in T, \quad i\in M \tag{I^D.d}$$

$$\gamma_t^i \leq 0, \qquad t\in T, \quad i\in M \tag{I^D.e}$$

$$\mu^i \leq 0, \qquad i\in M \tag{I^D.f}$$

$$\eta^i - \frac{1}{\alpha^i}\sum_{t\in T}\pi_t d_t^i = \beta^i + E_{\min}^i\mu^i, \quad i\in M. \tag{B-L-SD}$$

Problem (B-L) is a polynomial optimization problem, since there are quadratic terms in the formulation. Moreover, some of these terms appear in equality constraints such as (I_0^P.a), which implies that (B-L) is a nonconvex optimization problem. Thus, local optimality does not imply global optimality, and the solution of this kind of problems requires the use of specialized global optimization solvers, as we discuss in Section 6.4.1.

6.3.2 I-L Model: One Investor-Leader, Several Brokers-Followers

This model deals with the less realistic situation in which the investor makes his decision first and a set K of brokers react optimally to the leader's decision. More precisely, the investor first decides how much to invest with each broker $k\in K$, $\boldsymbol{x^k}$, so that $\sum_{k\in K}\sum_{j\in S} x_j^k = 1$ and then each broker sets prices. In this case, the bilevel model that represents this

hierarchical situation is formulated as follows:

$$\begin{aligned}
\underset{\eta,\boldsymbol{d},(\boldsymbol{x^k},\boldsymbol{y^k},\boldsymbol{p^k})_{k\in K}}{\text{minimize}} \quad & -\eta + \frac{1}{\alpha}\sum_{t\in T}\pi_t d_t && && && \text{(I-L-BILEVEL)}\\
\text{subject to} \quad & y_t^k = \sum_{j\in S}(r_{jt}x_j^k - p_j x_j^k), && t\in T, && k\in K && (\text{I}_0^{\text{P}}\text{.a})\\
& \sum_{k\in K}\sum_{t\in T}\pi_t y_t^k \geq E_{\min} && && && (\text{I}_0^{\text{P}}\text{.b})\\
& \sum_{j\in S}\sum_{k\in K} x_j^k = 1 && && && (\text{I}_0^{\text{P}}\text{.c})\\
& x_j^k \geq 0, && j\in S, && k\in K && (\text{I}_0^{\text{P}}\text{.d})\\
& d_t \geq \eta - \sum_{k\in K} y_t^k, && t\in T && && (\text{CVaR}^{\text{P}}\text{.a})\\
& d_t \geq 0, && t\in T && && (\text{CVaR}^{\text{P}}\text{.b})\\
& \forall k\in K, \quad \boldsymbol{p^k} \in \underset{\boldsymbol{p^k}}{\arg\max} && \sum_{j\in S} p_j^k x_j^k && && (\text{B}^{\text{P}})\\
& && \text{subject to} \quad \boldsymbol{p^k}\in P^k. && && (\text{B}^{\text{P}}\text{.a})
\end{aligned}$$

This situation is similar to the one discussed for the B-L model. We have a bilevel problem such that, once the decisions of the investor are fixed, the resulting subproblems for the brokers are linear. Thus, we can again rely on strong duality to combine the feasibility constraints of problems (B^{P}) and (B^{D}) with the strong duality constraints, (I-L-SD), to transform the bilevel problem into an equivalent single level problem:

$$\begin{aligned}
\underset{\eta,\boldsymbol{d},\boldsymbol{y},\boldsymbol{x}}{\text{minimize}} \quad & -\eta + \frac{1}{\alpha}\sum_{t\in T}\pi_t d_t && && && \text{(I-L)}\\
\text{subject to} \quad & y_t^k = \sum_{j\in S}(r_{jt}x_j^k - p_j x_j^k), && t\in T, && k\in K && (\text{I}_0^{\text{P}}\text{.a})\\
& \sum_{k\in K}\sum_{t\in T}\pi_t y_t^k \geq E_{\min} && && && (\text{I}_0^{\text{P}}\text{.b})\\
& \sum_{j\in S}\sum_{k\in K} x_j^k = 1 && && && (\text{I}_0^{\text{P}}\text{.c})\\
& x_j^k \geq 0, && j\in S, && k\in K && (\text{I}_0^{\text{P}}\text{.d})\\
& d_t \geq \eta - \sum_{k\in K} y_t^k, && t\in T && && (\text{CVaR}^{\text{P}}\text{.a})\\
& d_t \geq 0, && t\in T && && (\text{CVaR}^{\text{P}}\text{.b})\\
& \boldsymbol{p^k}\in P^k, && k\in K && && (\text{B}^{\text{P}}\text{.a})\\
& (\boldsymbol{v^k})^\top \boldsymbol{A^k} - (\boldsymbol{x^k})^\top = \boldsymbol{0}, && k\in K && && (\text{B}^{\text{D}}\text{.a})
\end{aligned}$$

$$v_s^k \geq 0, \qquad s \in P^{\mathrm{k,cons}}, \quad k \in K \tag{B$^{\mathrm{D}}$.b}$$

$$\sum_{j \in S} p_j^k x_j^k = \sum_{s \in P^{\mathrm{k,cons}}} b_s^k v_s^k, \quad k \in K. \tag{I-L-SD}$$

This optimization problem is again a nonconvex polynomial optimization problem, whose solution requires the use of global optimization solvers.

6.3.3 SW Model: Social Welfare Maximization

In this section we study the situation in which the broker and the investor "cooperate" and try to jointly optimize their respective objective functions. We refer to this situation as the Social Welfare Model, SW. Despite being a less realistic model, it represents a good benchmark to get a better understanding of the efficiency of the outcomes obtained by the B-L model and facilitate its comparison with the outcomes of the I-L model. As a by-product, the model also serves to quantify the potential benefits of cooperation in this setting. Since one of the main goals of this model is to facilitate a comparison of the outcomes of the bilevel models, we assume that there is only one broker and one investor.

The classic approach to study this kind of multi-objective problems, introduced more than a century ago by Italian economist Vilfredo Pareto (refer to Pareto (1971) for an English translation), consists in characterizing the two-dimensional Pareto frontier. Each coordinate in this set represents one of the objective functions to optimize and the Pareto efficient points are those feasible pairs (z_1, z_2) such that, for each other feasible pair in which one of the agents is better off, the other agent is worst off. A standard approach to compute the Pareto frontier consists in optimizing with respect to one of the objective functions while requiring that the other attains a minimum level. For each value of the minimum level, we obtain a Pareto efficient allocation, whereas the full Pareto frontier is generated by varying the minimum level within the range of possible values for the corresponding objective function. Therefore, if we let B_0 be a minimum level of the broker's profit, we are interested in the following optimization problem:

$$\underset{\eta, \boldsymbol{d}, \boldsymbol{y}, \boldsymbol{x}, \boldsymbol{p}}{\text{maximize}} \ \eta - \frac{1}{\alpha} \sum_{t \in T} \pi_t d_t \tag{SW$_{B_0}$}$$

$$\text{subject to} \ \sum_{j \in B} p_j x_j \geq B_0 \tag{SW$_{\mathrm{PE}}$}$$

$$y_t = \sum_{j \in S} r_{jt} x_j - \sum_{j \in S} p_j x_j, \quad t \in T \tag{I$_0^{\mathrm{P}}$.a}$$

$$\sum_{t \in T} \pi_t y_t \geq E_{\min} \tag{I$_0^{\mathrm{P}}$.b}$$

$$d_t \geq \eta - y_t, \quad t \in T \qquad (\text{CVaR}^{\text{P}}.\text{a})$$

$$d_t \geq 0, \qquad t \in T \qquad (\text{CVaR}^{\text{P}}.\text{b})$$

$$\sum_{j \in S} x_j = 1 \qquad (\text{I}_0^{\text{P}}.\text{c})$$

$$x_j \geq 0, \qquad j \in S \qquad (\text{I}_0^{\text{P}}.\text{d})$$

$$\boldsymbol{p} \in P. \qquad (\text{B}^{\text{P}}.\text{a})$$

In the above model, the objective function together with constraints (CVaR^{P}.a) and (CVaR^{P}.b), correctly define the CVaR of the returns y_t. Constraint (B^{P}.a) ensures the feasibility of the prices and constraints (I_0^{P}.b), (I_0^{P}.c), and (I_0^{P}.d) ensure that the chosen portfolio is feasible and achieves an expected return greater than $E_{\min}$. Then, constraints (I_0^{P}.a) contain the interaction between prices and returns. Finally, constraint (SW_{PE}) ensures the minimum level of the broker's profit and by varying B_0 we can construct the Pareto frontier. Note that there is a Pareto frontier for each value of $E_{\min}$.

Yet again, we are confronted with a nonconvex polynomial optimization problem, whose solution requires the use of global optimization solvers.

6.4 Numerical Results: Case Study for the Dow Jones Index

This section is devoted to present the numerical results associated to a case study that builds upon real data taken from the Dow Jones Index and try to hint at potential economic insights that might be worth studying further. In Section 6.4.1 below we start by discussing how the parameters required by the optimization models presented in Section 6.2 and Section 6.3 are obtained from the data on the Dow Jones Index. Further, we also discuss how the resulting instances of these models have been solved by using state-of-the-art global optimization solvers.

Once the data and models for the case study are in place, we proceed to discuss the results. First, we separately analyse the B-L model in Section 6.4.2. Next, we present the results for the I-L model and a comparison of the results for the two models in Section 6.4.3. Finally, in Section 6.4.4 we discuss the results for the SW model, along with some concluding comments on the overall analysis.

It is important to note that we have performed similar analyses to the one we present below, but taking alternative financial indices (such as the Spanish IBEX) and different estimates on future realizations of the returns of their securities. We have found that the qualitative results and economic insights that we present below for the Dow Jones' companies also appeared quite consistently in all other analyses.

6.4.1 Instances and Solution Technique

There are two main ingredients common to all the models we have discussed: the set of securities in which to invest and the set of constraints P, that limits the fees the broker can impose on those securities. Regarding the latter, for the case study in this section we assume that the set P is given by the following constraints:

$$\begin{aligned} \sum_{j \in S} p_i &\leq 0.3 \\ p_j &\leq 0.1, \quad j \in S \\ p_j &\geq 0, \quad j \in S. \end{aligned}$$

The above constraints are just a simple example of the polyhedral set P, which facilitates the ensuing analysis. There is a maximum fee that can be imposed on each security and there is also a limit on the sum of the fees on the different securities. One natural consequence of these constraints is that the broker prefers to face investors which do not diversify much. If the investment is made on three or less securities, then the broker can get a profit of 0.1 by putting a fee of 0.1 in all those securities. On the contrary, if the investor puts his money in more than three securities, then the broker cannot go for a maximum fee of 0.1 in all of them and has to choose on which ones to concentrate. We do not claim that this choice for the set P is particularly realistic, but our approach could be readily applied to any other polyhedral set P.[9] As we have already argued, in actual cases these constraints may be imposed by a regulator, by some board of the company based on considerations regarding their competitors, or by both of them.

Regarding the set of securities, we consider those associated with the 30 companies of the Dow Jones Index: AAPL, AXP, BA, CAT, CSCO, CVX, DIS, DWDP, GS, HD, IBM, INTC, JNJ, JPM, KO, MCD, MMM, MRK, MSFT, NKE, PFE, PG, TRV, UNH, UTX, V, VZ, WBA, WMT, and XOM.[10]

For the above securities we have tracked the weekly market return during a six-month period, from August 15th, 2018, to March 17th, 2019. This results in a total of 30 vectors of joint realizations for the securities under study. Then, we have taken these past realizations and assumed them to be equally likely future scenarios. We do not claim that these estimates for future realizations represent optimal or accurate predictions in any sense. We

[9] In particular, the usual market functioning in which the fee has to be common to all assets is achieved, for instance, by adding the constraints $p_i = p_1$ for each $i \neq 1$. Adding these constraints to the set P in our case study would result in a straightforward decision problem for the broker, in which he would evenly split his "total budget", 0.3, among the different assets. The resulting outcomes would be worse for the broker than those obtained in our analysis, where he can associate different fees to the different assets.

[10] Information on the associated companies can be checked, for instance, at https://finance.yahoo.com/.

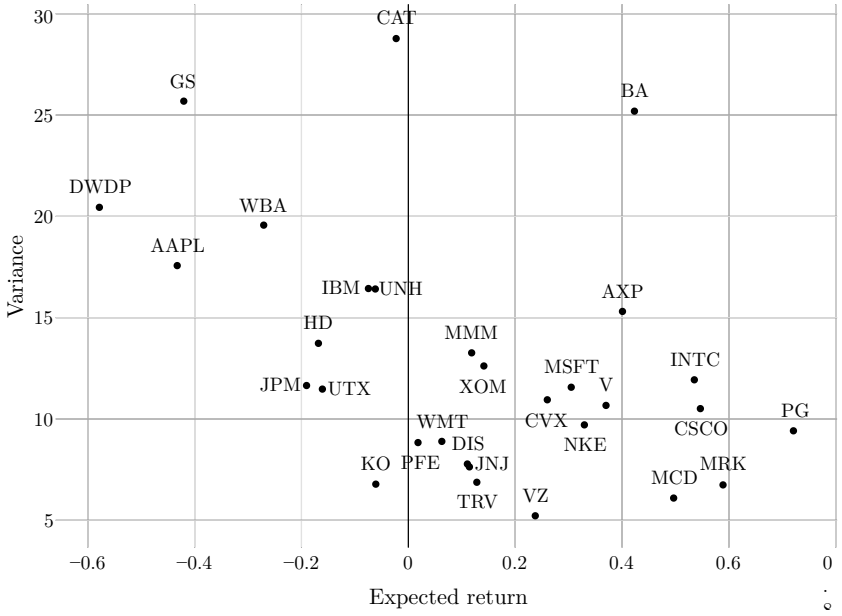


Figure 6.1: Expected return and variance for the Dow Jones securities.

just consider that they represent realistic scenarios (they have appeared in the past) which, for the sake of our analysis, ease the exposition and deliver similar qualitative results to those that would be obtained after more rigorous scenario estimations based, for instance, on time series techniques. In Figure 6.1 we represent the expected value and variance of the R_j random variables associated with each of the 30 Dow Jones securities in the 30 scenarios we have considered.

The mathematical programming problems associated to the "one level" formulations of the B-L, I-L, and SW models have been formulated in AMPL modelling language (Fourer et al., 1990). Then, each of the resulting instances has been solved on a server of ITMATI[11] using two global optimization solvers: `BARON` (Tawarmalani and Sahinidis, 2005; Sahinidis, 2021), a general solver for mixed-integer nonlinear programming problems, and `RAPOSa` (González-Rodríguez et al., 2022), the solver introduced in Chapter 2, specifically designed for polynomial programming problems, a class to which all the models discussed in this chapter belong to. Further, we have analysed the issue of the scalability of our methodology. In this regard, we have compared the above solution approach with different methodologies built in local solvers: `Ipopt` (Wächter and Biegler, 2006), `Knitro` (Byrd et al., 2006) and `MINOS` (Murtagh and Saunders, 1978). The results are reported in Appendix 6.A.1. As expected, global optimization solvers such as `BARON` and `RAPOSa` deliver the best solutions for small instances but, as problem sizes increase, they find it harder to close the optimality

[11] Technological Institute for Industrial Mathematics (http://www.itmati.com/en).

gap and global optimality is no longer guaranteed. In spite of that, `BARON` finds in most of the cases the best solution. On the other hand, local optimizers are much faster and always deliver solutions without even approaching the time limit (one hour). With the exception of `MINOS`, which performs rather poorly, the quality of the solutions obtained by these local solvers is quite acceptable, with `Knitro` being competitive with `BARON` for the largest instances.

There are a couple of important parameters associated with all the models we have presented, so it is important to clarify their values in the analysis.

Risk profile of the investor. We consider four different risk profiles for the investors, 0.05, 0.25, 0.50, and 0.99, with the first one being the most risk averse and the last one being essentially an expected utility maximizer.

Minimum expected return. In all the optimization models there is a constraint that sets the minimum expected return, $E_{\min}$, that the investor is willing to accept in order to make his investment. What we do in our analysis is to solve a set of 32 problems associated with different values of $E_{\min}$ ranging from -0.1 to 0.72 (the highest expected return of any security).

6.4.2 B-L Model

We start discussing the results assuming that there is only an investor profile which, as we have argued before, can be thought of as a continuum of agents with the same degree of risk aversion. Next, we present results for the model in which the population of followers is composed of investors with different degrees of risk aversion.

B-L Model. One Follower

For each risk profile $\alpha \in \{0.05, 0.25, 0.50, 0.99\}$, we solve the associated B-L model. In Figure 6.2 we present, for each value of α, a line representing the objective function CVaR_α for each value of $E_{\min}$. This figure seems to suggest a counterintuitive result. Namely, for each level of $E_{\min}$, the more risk averse the investor is, the more risk he must assume in the optimal solution. However, note that each line represents the CVaR_α associated with the optimal solution for that level α, so the degrees of risk implied by the different lines are not really comparable. For instance, in the line associated with type $\alpha = 0.05$ we can see that, in order to get an expected return of 0.2, the investor has to be willing to risk an average loss of almost 3 in the tail with the worst 5% realizations (given his security choices).

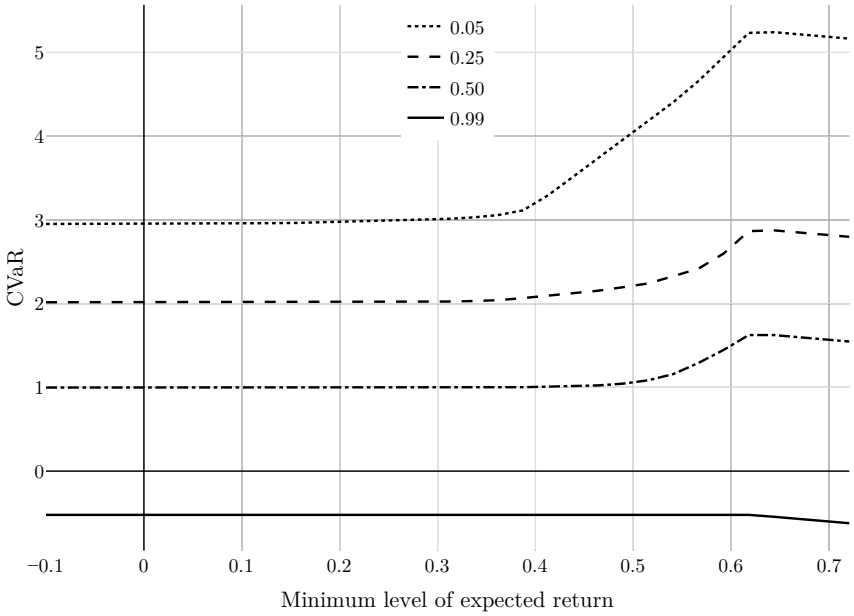
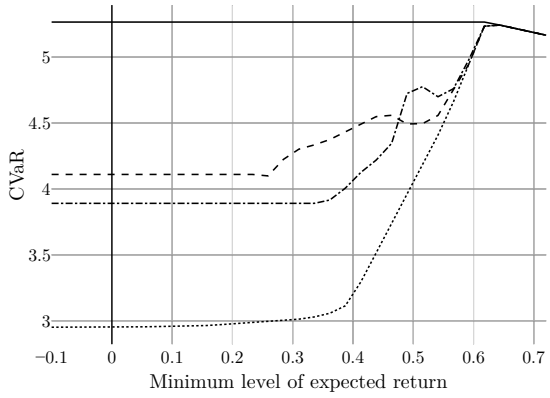


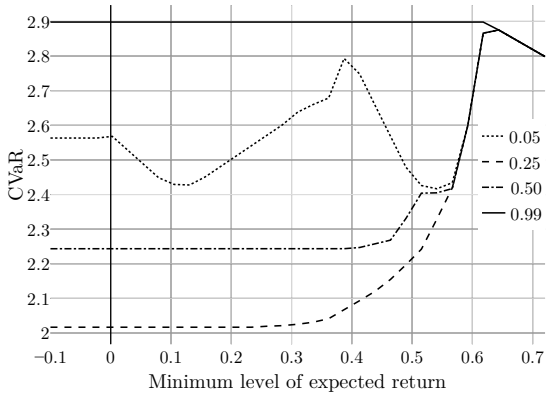
Figure 6.2: CVaR_α for each risk profile and each value of $E_{\min}$.

In order to obtain comparable results across risk profiles, in Figure 6.3 we present four subfigures, one for each value of α. For instance, Figure 6.3a contains the $\text{CVaR}_{0.05}$ associated to the optimal solutions of the different risk profiles (again, for each level of $E_{\min}$). We now find, as expected, that the line corresponding to each risk profile α is the one minimizing the corresponding CVaR_α. Interestingly, setting aside the 0.99 type, which is usually the one assuming the highest risk according to all other risk profiles (he just goes for the security with the highest expected return regardless of the value of $E_{\min}$), there is no clear pattern characterizing how the other risk profiles are ranked. For instance, in Figure 6.3a, the solutions for type 0.25 have a higher $\text{CVaR}_{0.05}$ than the solutions for type 0.50 for values of $E_{\min}$ below 0.45, but the situation reverses after that point.

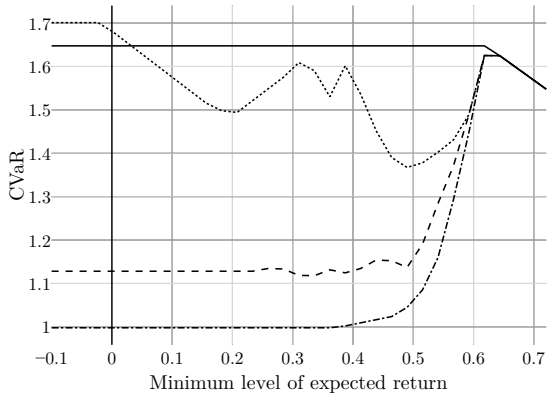
Going back to Figure 6.2, we can also see that, for each given risk profile, the CVaR at optimality remains constant for small values of $E_{\min}$, the reason being that the associated constraint is not demanding (indeed, not even binding at optimality). Once the threshold on the minimum level of expected return starts to matter, we see that the risk associated with the optimal solutions starts to increase; the risk-return trade-off starts to kick in. Then, once the value of $E_{\min}$ gets larger than 0.62 we see that the risk starts decreasing for each risk type, which may be counterintuitive at first. This is because, for sufficiently large values of $E_{\min}$, to get a large enough expected return, the investor has to put all his money in the security PG, the one with the highest return, 0.72 (see Figure 6.1). The broker anticipates this investment and puts a fee of 0.1 on security PG, resulting in an



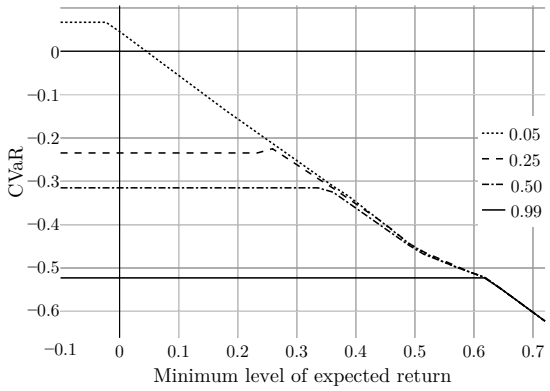
(a) $\text{CVaR}_{0.05}$.



(b) $\text{CVaR}_{0.25}$.

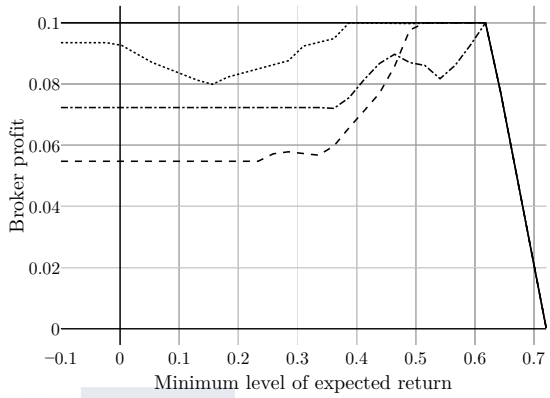


(c) $\text{CVaR}_{0.50}$.

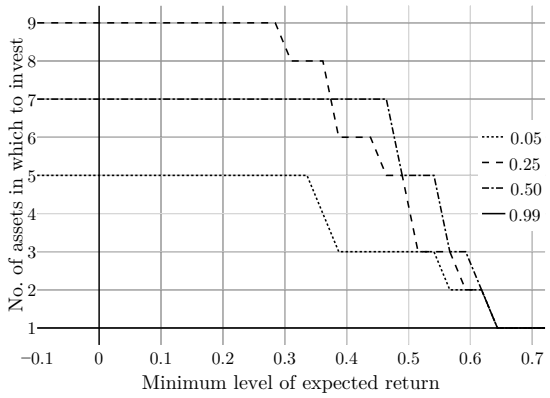


(d) $\text{CVaR}_{0.99}$.

Figure 6.3: Comparison of CVaR levels at the optimal solutions of the different risk profiles.



(a) Broker profit at optimality.



(b) Number of securities in the optimal portfolio.

Figure 6.4: Broker profit and number of securities at optimality for different risk profiles.

expected return of 0.62 for the investor. Beyond that level, in order to let the agent have feasible solutions in his "subproblem", the broker must reduce the fees and, indeed, we see in Figure 6.4a how the broker's profit decreases from that point onwards. We can also see in Figure 6.4a that for moderate values of $E_{\min}$, those at which the associated constraint is already demanding, the profit of the broker tends to be increasing. The reason for this is that the investor cannot diversify so much in order to get the target expected return and becomes more predictable for the broker. This can be seen in Figure 6.4b, where we see that the number of securities in which the agent invests decreases as $E_{\min}$ increases (until eventually the investment is concentrated on security PG).

To conclude the discussion regarding the B-L model with one follower, note that the broker seems to prefer to face investors with "extreme" degrees of risk aversion (0.05 and 0.99 in our analysis), since these types of investors tend to have less "appealing" securities in which to invest, making them more predictable (see Figure 6.4).

B-L Model. Multiple Followers

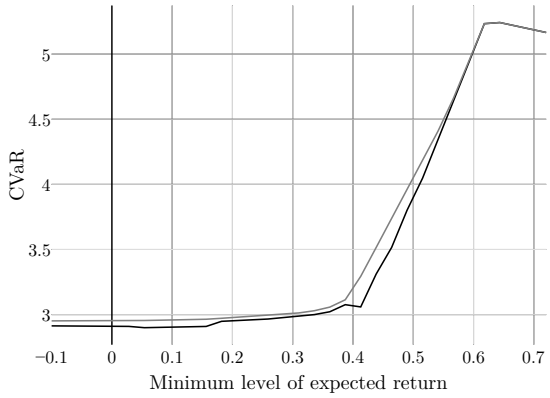
In this section we solve again the B-L model, but now with different investor profiles choosing simultaneously their portfolios at the second level, once the broker has fixed the fees on the different securities. We consider three different profiles, with $\alpha \in \{0.05, 0.50, 0.99\}$. The main reason for not including $\alpha = 0.25$ in the analysis is that the optimization problem with three investors on the second level is already quite involved, with the optimizers experiencing already some minor numerical difficulties. Most instances required more than one hour to be solved to optimality and adding a fourth profile just made time requirements even more demanding and numerical issues more accused.[12]

Since the qualitative results are analogous to those for the case with a single follower, we have moved the associated figures to Appendix 6.A.2 (figures 6.14, 6.15, and 6.16). The interesting part of the analysis comes when we compare the results obtained for the two cases: single follower and multiple followers.

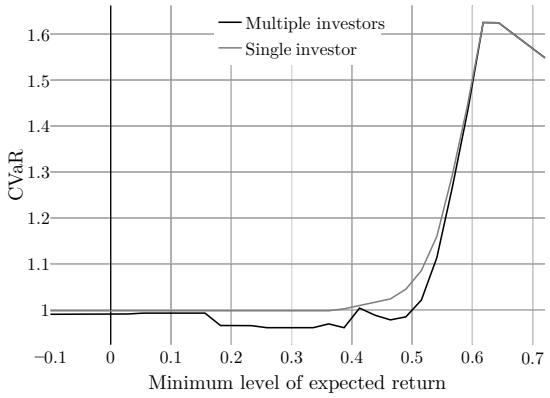
Intuitively, the situation with investors with different risk profiles acting simultaneously in the second level should be beneficial for them and reduce the profit of the broker, since investors with different profiles may go for different portfolios and the broker cannot put a maximal fee of 0.1 in all the involved securities. In the model with a single investor, the broker could tailor his fees specifically to each risk profile, and now has to divide his "total budget" of 0.3 among them. Accordingly, in Figure 6.5 we can see that, when the

[12] Indeed, the reader will notice that, in the figures in this section, the lines for the model with multiple followers are not as smooth as the ones for the model with a single follower. This is because the precision obtained in the final solutions is smaller in the former.

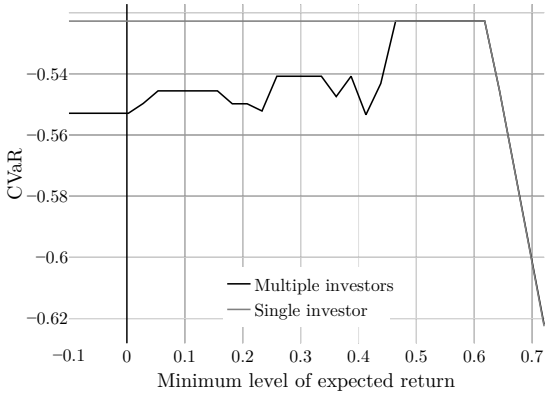
investors act independently, they must assume higher risks (higher CVaR) to ensure a given expected return. Further, this difference disappears when $E_{\min}$ is sufficiently large so that all the investors must essentially go for security PG and then the CVaR values for both models coincide.



(a) $\text{CVaR}_{0.05}$ risk profiles.



(b) $\text{CVaR}_{0.50}$ risk profiles.

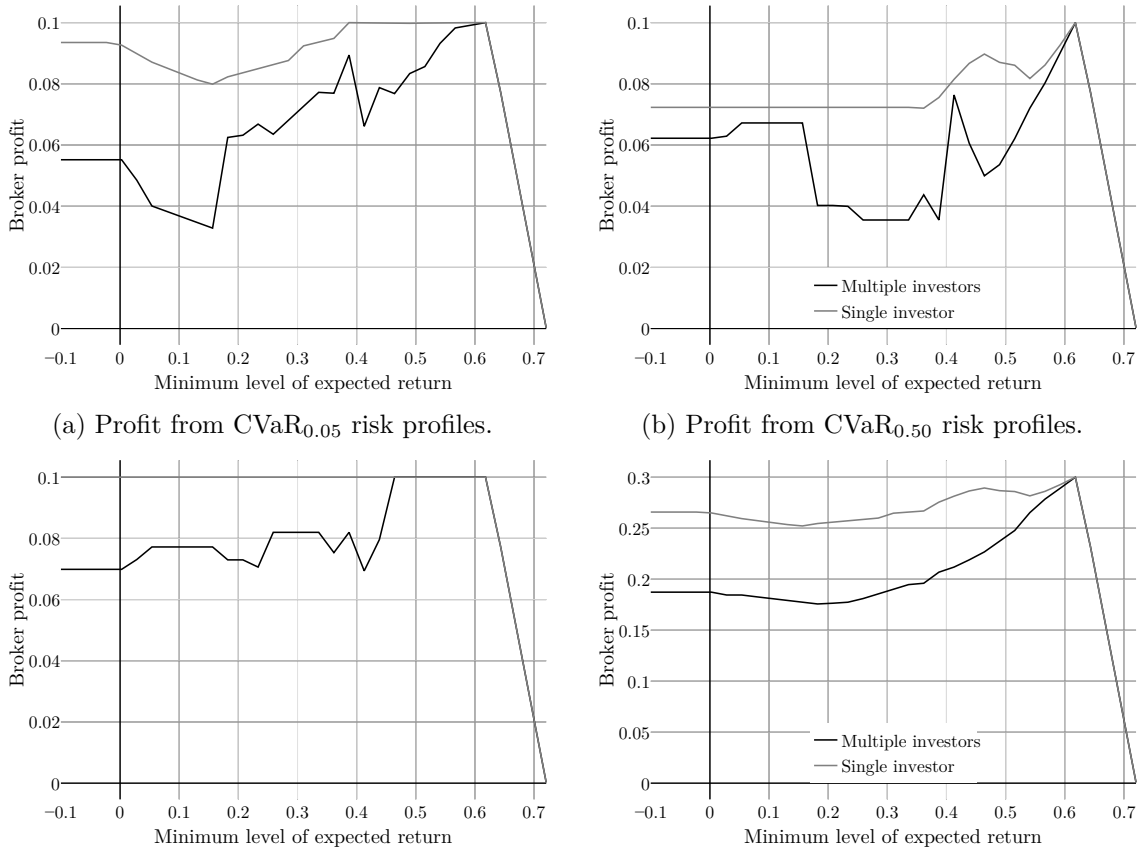


(c) $\text{CVaR}_{0.99}$ risk profiles.

Figure 6.5: CVaR_α for the models with one follower and multiple followers.

Figure 6.6 shows the profit for the broker which, as expected, is smaller in the multiple follower model. The effect is very clean for the investor with risk profile $\alpha = 0.99$. Since this investor just wants to go for security PG regardless of the value of $E_{\min}$, in the case of a single follower the broker can extract for him his maximum possible benefit, 0.1. However, in the case with multiple followers, the broker cannot extract so much profit from this investor, because of the trade-off with the fees on the securities of the other investors. Additionally, we can see that, although the profit of the broker tends to increase with the value of $E_{\min}$, the profit is not a monotone function of it. The reason for this is that, for relatively small values of $E_{\min}$, it is not so clear how the diversification on the different

securities will affect the benefit of the broker (it is not just a matter of the total number of securities in which the investor invests, but also about how evenly the money is split among them).[13]



(a) Profit from $\text{CVaR}_{0.05}$ risk profiles.

(b) Profit from $\text{CVaR}_{0.50}$ risk profiles.

(c) Profit from $\text{CVaR}_{0.99}$ risk profiles.

(d) Total profit in each model.

Figure 6.6: Profit of the broker for the models with one follower and multiple followers.

The above discussion of the B-L model hints at the following insight: given a market with multiple brokers and multiple investors, even if each of them can choose asset specific fees, as long as these fees cannot be different for different investors, then investors are better-off by concentrating all on the same broker, limiting his power to tailor the fees to specific risk profiles. Validating such an insight would require the analysis of a model with multiple leading brokers, which goes beyond the scope of this work.

[13] For the sake of completeness, in Figure 6.15b in Appendix 6.A.2 we represent the number of securities in which each risk profile invests as a function of $E_{\min}$.

6.4.3 I-L Model and Comparison of B-L and I-L Models

We now move to the setting in which the investor is the leader, with the broker choosing the fees with full knowledge of the investment made on each security. Arguably, this is a less realistic model which, intuitively, should favour the broker. For the sake of exposition, we present the figures that allow to compare the B-L and I-L models. Before proceeding, it is worth discussing a couple of "technical" differences between the results obtained for the different models:

- For the I-L model we do not consider the case of multiple followers. The reason is that, as long as all the brokers face the same constraints, there is no additional richness and the investor cannot do better than what he would do in the model with a single follower. On the other hand, although allowing for different feasible sets for the different brokers might lead to different results and additional insights, such an analysis is beyond the scope of this work.

- The value of $E_{\min}$ is taken in the interval $[-0.1, 0.62]$ (recall that in the B-L model the interval was $[-0.1, 0.72]$). This difference is driven by the order in which the investor and the broker take their actions. In the I-L model, once the investor has chosen his portfolio, the broker will choose fees in order to maximize his profit. In particular, in order to get an expected value of at least 0.62, the investor has to put all the money in security PG (which has an expected return of 0.72), after which the broker would associate a 0.1 fee to security PG, leading to an expected return of precisely 0.62. The situation in the B-L model was different since the broker chooses first and can "credibly" let the investor gain more than 0.62 by associating a fee smaller than 0.1 to security PG.

We start the analysis with Figure 6.7, which contains a comparison of the CVaR values obtained for the I-L (grey) and B-L (black) models and shows that, for the investor, both models result in essentially the same objective function. Yet, if we were to zoom in into the different lines, we would observe that the CVaR for the B-L is often slightly higher than the one for the I-L model (and never lower). Qualitatively this is quite natural, since the broker chooses his fees being fully informed of the investor's portfolio, the investor's objective function gets worse with respect to the B-L model.

Interestingly, although the difference in the CVaR values is very small, the difference is more noticeable in the associated portfolios. In particular, it seems that in the I-L model the investor tends to diversify more in order to reduce the fees paid to the broker (even if it requires to choose a portfolio with slightly worse returns). This difference is

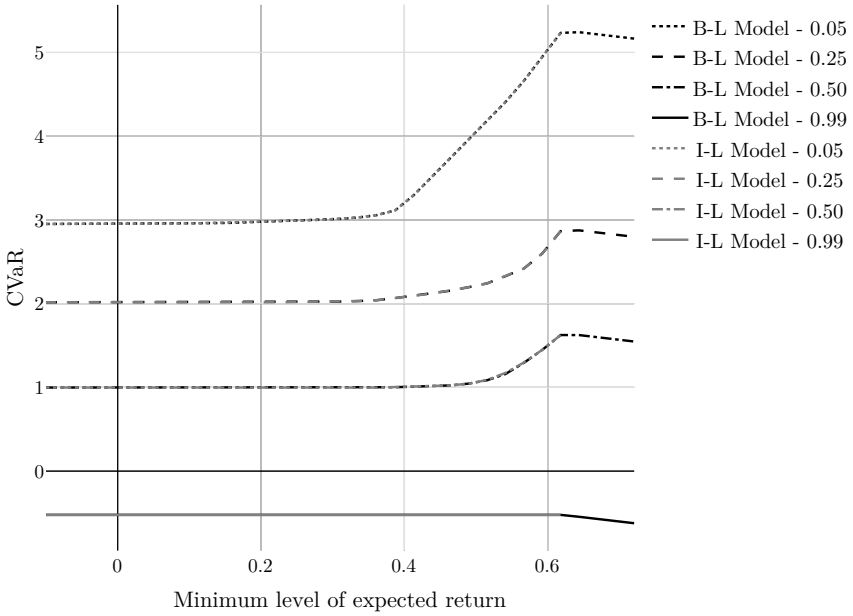
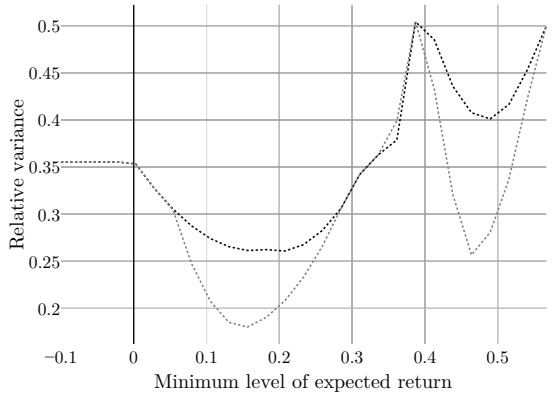


Figure 6.7: CVaR_α for each risk profile and each value of $E_{\min}$ in the B-L and I-L models.

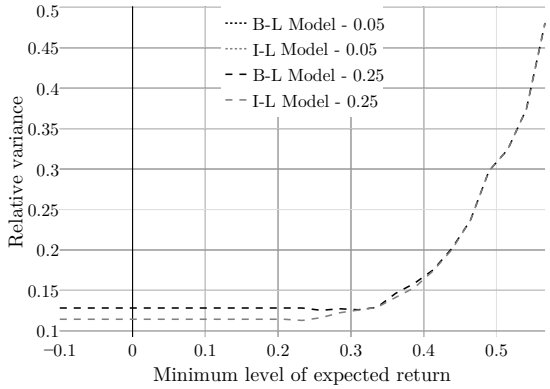
illustrated in Figure 6.8, where we represent the relative variance of the optimal portfolios. More precisely, the variance of portfolio $\boldsymbol{x}$ is computed as $\sum_{j\in S}(x_j - 1/30)^2$. In this case, the largest possible variance is achieved when all the investment is made on the same asset, resulting on a value of 0.0322.[14] On the other hand, the variance would be 0 if all assets get investment 1/30, which corresponds with full diversification. In Figure 6.8 we represent the relative variances of the portfolios, computed as the variances divided by 0.0322, the maximum variance. Since the smaller relative variance, the larger the diversification, the figure shows that, indeed, the I-L tends to exhibit more diversification. Finally, this additional diversification in the I-L model should also result in a decrease of the profit of the broker, which we show in Figure 6.9.

The above discussion and the associated figures show that the outcomes of the B-L model Pareto dominate those of the I-L model. Thus, not only the B-L model seems more realistic, but also seems to lead to outcomes that are more efficient from the social point of view.

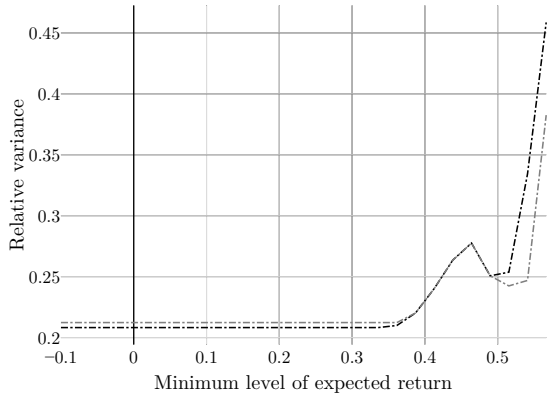
[14] Given a feasible portfolio $x \in [0,1]^{30}$, the condition $\sum_{j\in S} x_j = 1$ implies that the average investment is 1/30. Thus, when the portfolio concentrates on a single asset, we get a term of the form $(1 - 1/30)^2$ which leads to the highest possible variance.



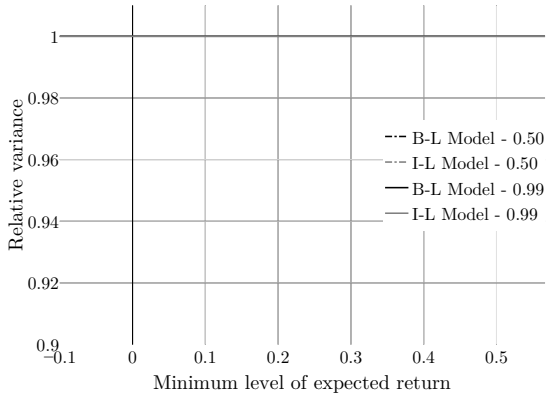
(a) $\text{CVaR}_{0.05}$ risk profiles.



(b) $\text{CVaR}_{0.25}$ risk profiles.



(c) $\text{CVaR}_{0.50}$ risk profiles.



(d) $\text{CVaR}_{0.99}$ risk profiles.

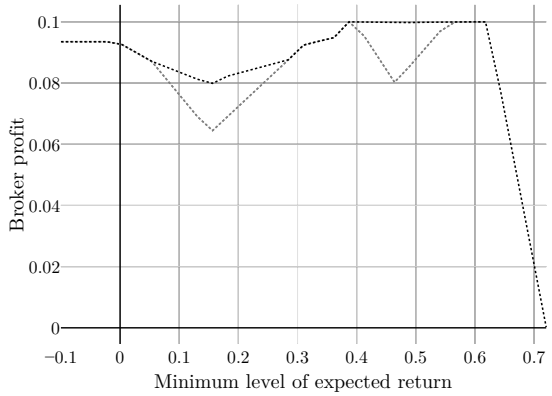
Figure 6.8: Relative variances of the optimal portfolios in B-L and I-L models.

6.4.4 SW Model

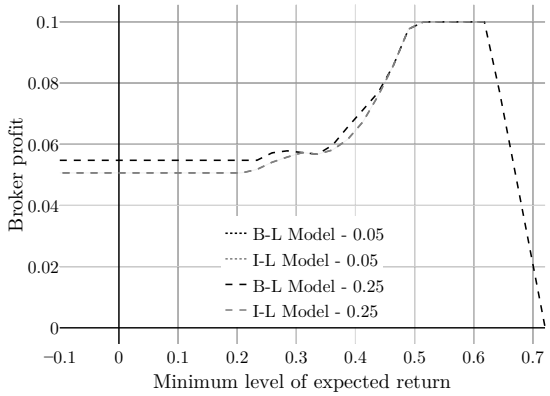
The final discussion in the previous section makes it natural to study the SW model in order to understand not only how the outcomes of the B-L and I-L models relate to each other in terms of Pareto domination, but also how they perform in terms of Pareto efficiency.

Recall that if we fix a value for $E_{\min}$ then, varying the value of B_0 in model (SW_{B_0}), we obtain the Pareto frontier associated with minimum return $E_{\min}$. In Figure 6.10 we present the Pareto frontiers for four different values of $E_{\min}$, evenly distributed among those values in the grid used in the previous sections (0.156, 0.310, 0.464, and 0.617). In this figure we also represent the outcomes obtained in the B-L and I-L models for the same values of $E_{\min}$, so we can get a better understanding of the efficiency of these models.

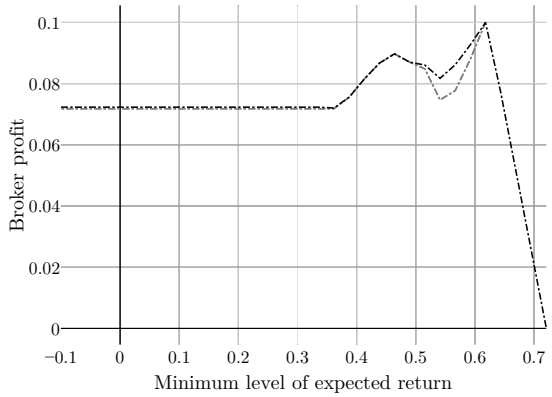
As expected, the results in Figure 6.10 confirm the findings in the previous section. The outcomes of B-L and I-L models are generally close to each other but, for some instances, the B-L outcome Pareto dominates the I-L outcome. The clearest cases in Figure 6.10



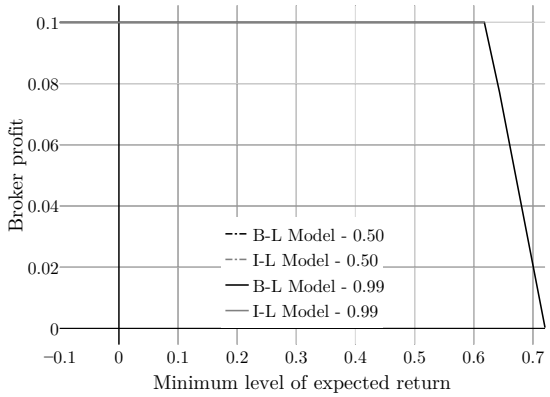
(a) Profit from $\text{CVaR}_{0.05}$ risk profiles.



(b) Profit from $\text{CVaR}_{0.25}$ risk profiles.



(c) Profit from $\text{CVaR}_{0.50}$ risk profiles.

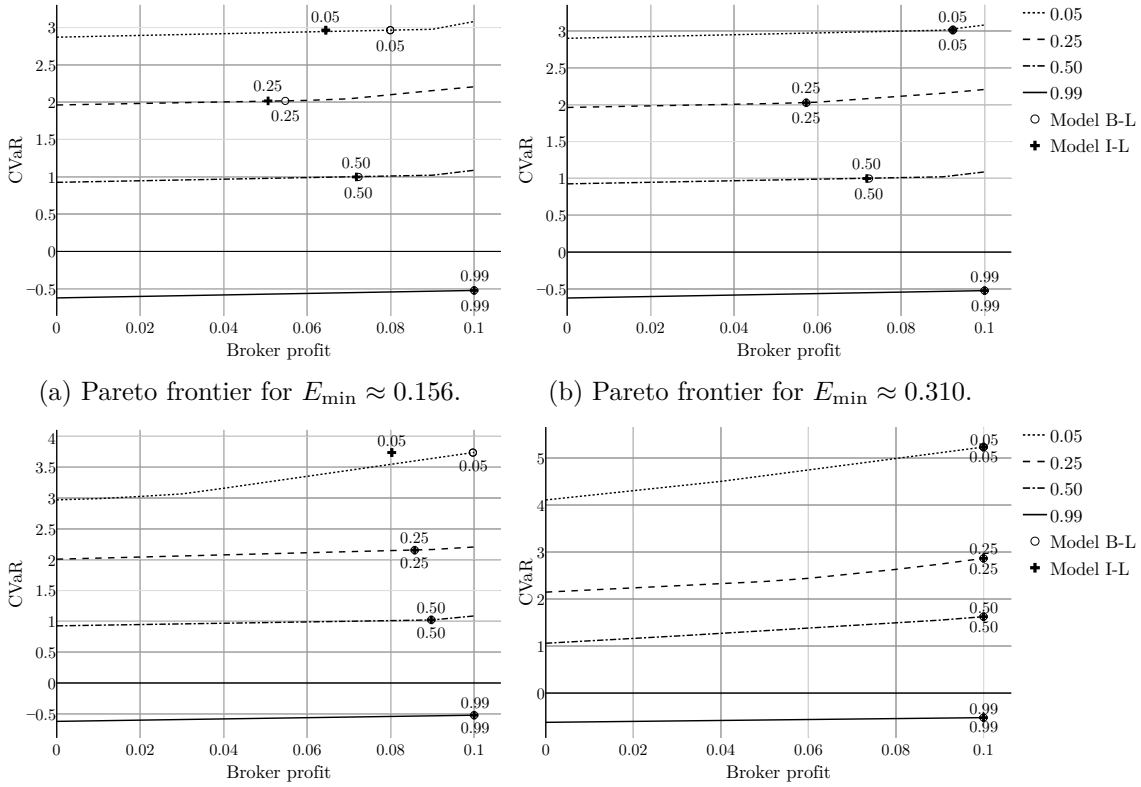


(d) Profit from $\text{CVaR}_{0.99}$ risk profiles.

Figure 6.9: Profit of the broker in B-L and I-L models.

are obtained for $E_{\min} = 0.156$ and $E_{\min} = 0.464$ with $\alpha = 0.05$. Last, but not least, note also that the B-L outcomes always lie in the Pareto frontier, whereas the I-L outcomes are sometimes inefficient. Thus, our numerical results show some form of domination of the B-L model with respect to the I-L model. An interesting line of research, that goes beyond the scope of this work, would be to explore to what extent these numerical results can be backed up with theoretical results, not only for the models discussed in this work but, possibly, with respect to more general versions of them.

As a conclusion of the discussion in the last two sections on the comparison of the B-L model with the I-L and SW models, we want to highlight once again the fact that it is precisely the most realistic model, B-L, the one leading to efficient outcomes. This is in a sense reassuring for the way in which these markets currently operate. In particular, the results suggest that the I-L model might lead to inefficient markets in which investors might choose to "give up" some rent in order to be less predictable.



(a) Pareto frontier for $E_{\min} \approx 0.156$.

(b) Pareto frontier for $E_{\min} \approx 0.310$.

(c) Pareto frontier for $E_{\min} \approx 0.464$.

(d) Pareto frontier for $E_{\min} \approx 0.617$.

Figure 6.10: Pareto frontiers and solutions for B-L and I-L models.

6.5 Conclusions

This chapter deals with a single-period portfolio selection problem with transaction cost and two levels of decision-making. On the one hand, there is a broker that controls the fee to be charged to the different securities in order to maximize his benefit and, on the other hand, there is an investor that chooses his portfolio trying to minimize the risk while ensuring a certain expected return. This structure gives rise to an implicit competition in order to anticipate the rational decision of the other party so as to optimize the decision-makers' own criteria. We have presented different models depending on the order in which choices are made. This produces three situations: i) the main one, in which the broker is leader and the investor follows with his decision (B-L model), ii) the investor acts first and the broker is the follower (I-L model), and iii) the social welfare model (SW) in which both parties collaborate to maximize the aggregated objective functions. We have developed mathematical programming formulations to model the three of them. As opposed to the models in Leal et al. (2020), we have assumed continuous sets of possible transaction costs,

which leads to nonlinear and nonconvex optimization problems. The first two are bilevel programs that can be reformulated into single level polynomial optimization problems and the third model also belongs to this class. We solved them using two global optimization solvers: `BARON` and `RAPOSa`. To illustrate the economic insights that can be gained with these models, we have developed a case study based on data from the Dow Jones index with weekly market returns during a six months period from August 15th, 2018, to March 17th, 2019. These results report a comparison among the models from different angles. One of the most interesting findings of our analysis is that there seems to be some form of domination of the B-L model over the I-L model with respect to the broker-dealer profit and the CVaR risk obtained by the two decision-makers, with the B-L model delivering Pareto efficient outcomes. This work opens up some lines for future research such as the extensions to multiple leader-follower models, the consideration of multiple period situations, and the study of whether or not some of the economic insights from the case study can lead to theoretical results, not only for the models discussed in this work, but also to some of their extensions mentioned above.

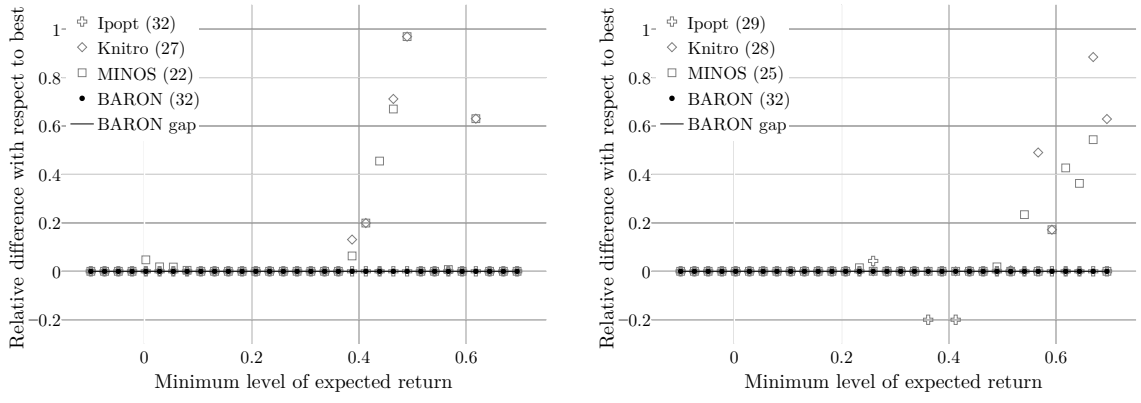
6.A Appendix

6.A.1 Comparison of Different Solution Methods

This appendix is devoted to compare different solution methods applied to the problems in this chapter. Additionally, we have also addressed the issue of scalability of our solution technique as compared with other solution methods for larger size instances. To this end, we have studied solution techniques built in different local solvers: `Ipopt` (Wächter and Biegler, 2006), `Knitro` (Byrd et al., 2006) and `MINOS` (Murtagh and Saunders, 1978). The comparative analysis of the quality of solutions found and running times is shown in figures 6.11, 6.12 and 6.13.

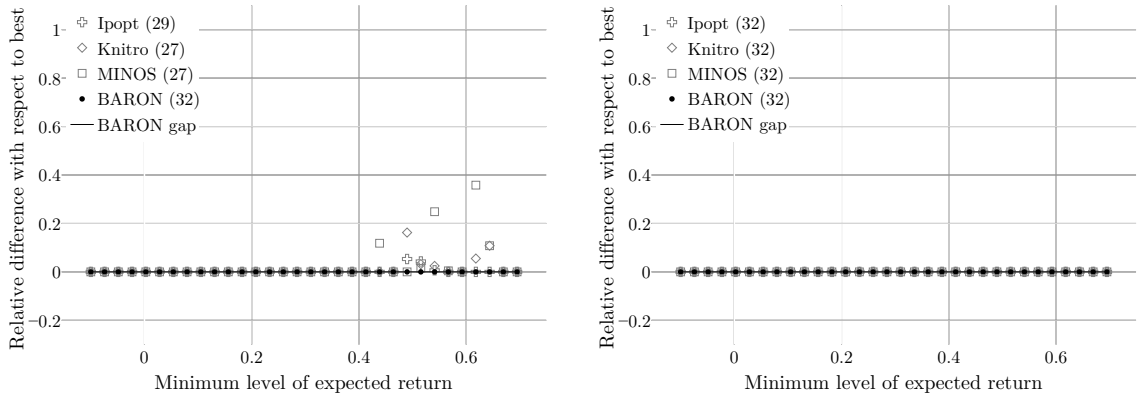
Figure 6.11 compares four solution methods built in different solvers (three of them local `Ipopt`, `Knitro` and `MINOS` and one global `BARON`) over our benchmark problem B-L with one investor and different risk profiles ($\alpha \in \{0.05, 0.25, 0.50, 0.99\}$). After the name of each solver, we show the number of problems, out of 32, in which the corresponding solver found the best solution. On the x-axis we represent the different instances (as the minimum level of expected return varies). On the y-axis we represent the relative difference in the quality of the solution of each solver compared to the best solver.[15] The performance

[15] For instance, a value 0.5 for a solver indicates that it obtained an objective function worse, by 50%, than the best solution found. To provide additional information on the quality of the solutions, we also represent the final optimality gap obtained by `BARON`.



(a) One investor with $\alpha = 0.05$.

(b) One investor with $\alpha = 0.25$.



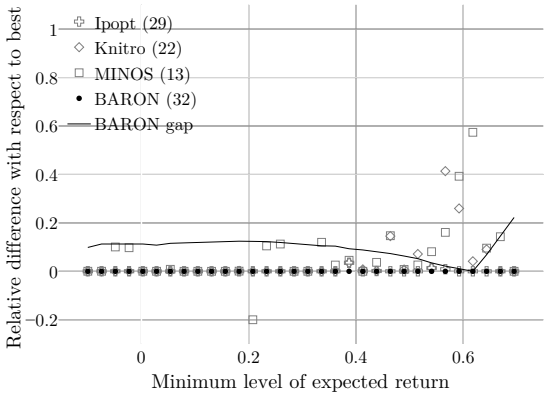
(c) One investor with $\alpha = 0.50$.

(d) One investor with $\alpha = 0.99$.

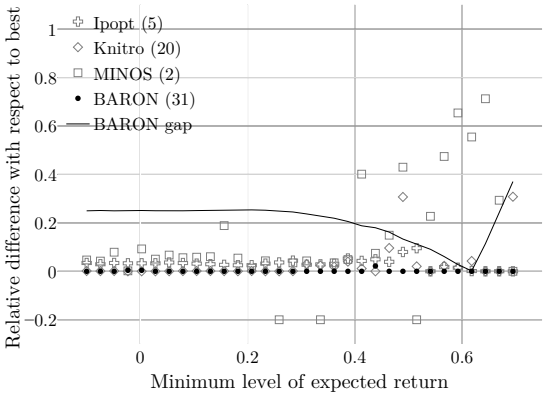
Figure 6.11: Comparison between solvers in BoT model with one investor and different risk profiles.

of the different methods is quite similar and most of them find optimal or near optimal solutions within the time limit. Despite the good performance of all solvers, `BARON` is the only one finding the best solution for each problem. Indeed, given the reported gaps, we know that `BARON` found the global optimum for these problems. Note that `MINOS` is the solver with the worst performance.

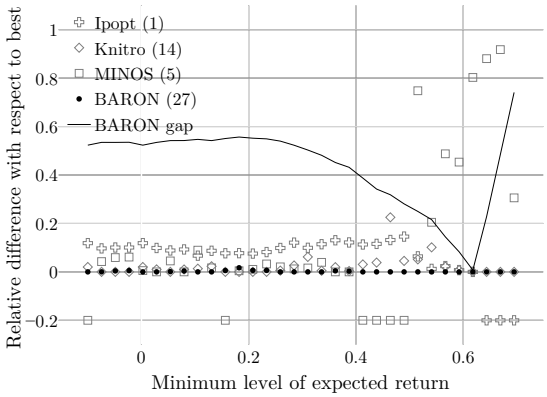
Figure 6.12 analyses the issue of scalability of solutions methods for larger problems. We report results for problems with 3 to 20 investor profiles. Again, after the name of each solver we show the number of problems, out of 32, in which the corresponding solver found the best solution. Increasing the number of investors augments the complexity of these problems. As it can be seen in the graphs, our original solution method (based on the global solver `BARON`) is quite robust and in almost all cases it provides the best results in terms of gap with respect to the best solution found. Yet, we can see how, as problem sizes increase, `BARON` finds it more difficult to close the optimality gap and, indeed, for the



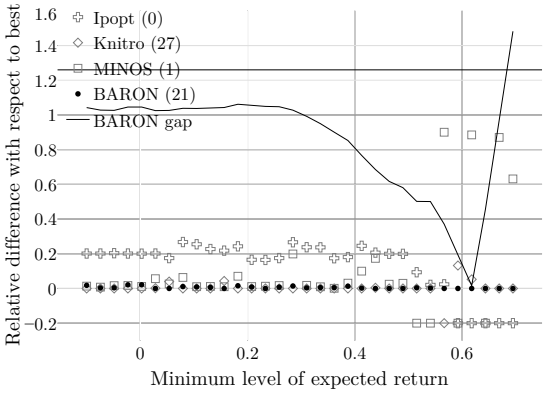
(a) Three investors.



(b) Five investors.



(c) Ten investors.



(d) Twenty investors.

Figure 6.12: Comparison between solvers in BoT model when the number of investors increases.

larger instances `Knitro`'s results are competitive with those obtained by `BARON`.

Finally, for the more difficult problem with 20 investor profiles, we report the running time required for the different methods until they stop with a possibly local optimal solution. (Observe that local solvers report solution without guaranteed global optimality). The three local solvers always report a solution quickly (in all cases less than 200 seconds) whereas `BARON` always took one hour trying to certify optimality. Comparing the local solvers, the most efficient for these problems seems to be again `Knitro`, which is the fastest with only one exception.

6.A.2 Additional Graphs

In this section we present the Figures associated with the B-L model with multiple followers. These are parallel to those discussed in Section 6.4.2 for the case with one follower.

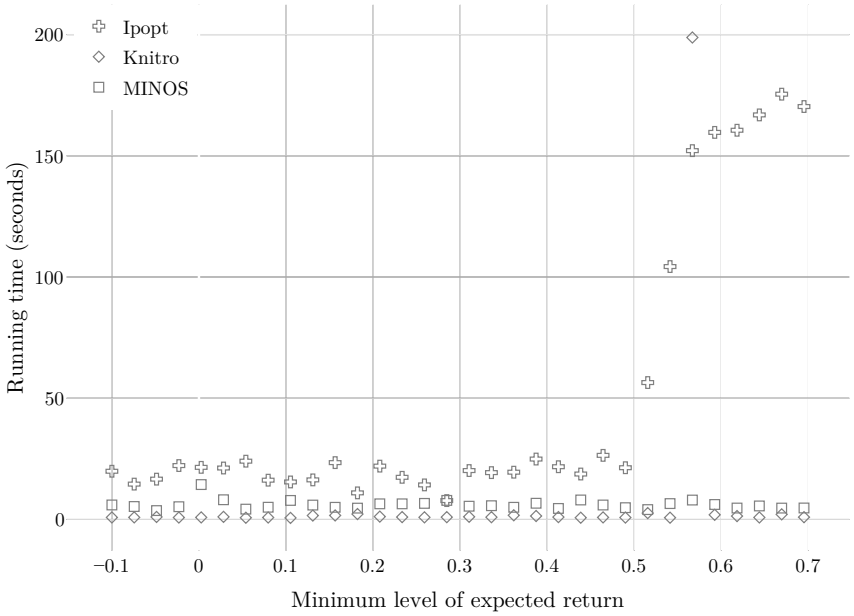
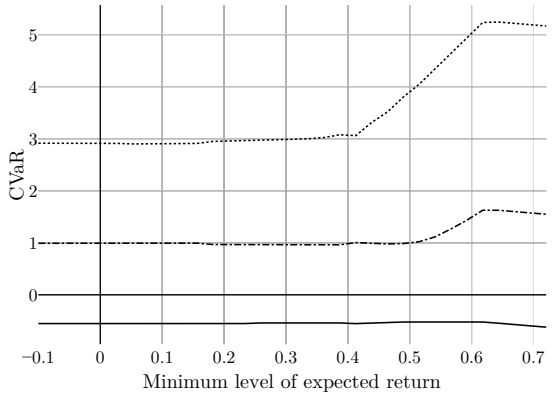


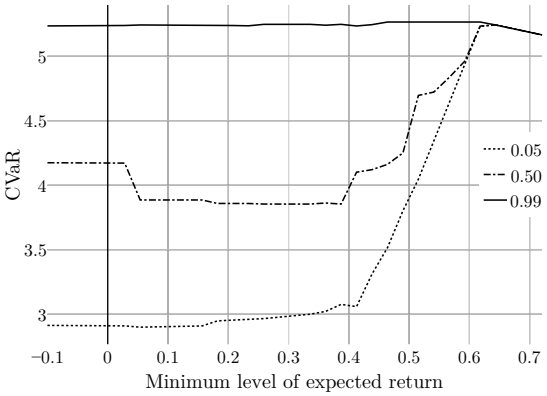
Figure 6.13: Running time in BoT model with 20 investors.

In Figure 6.14 we show the objective function of each risk profile as a function of the value of $E_{\min}$ and also the CVaR_α at the different optimal solutions for each $\alpha \in \{0.05, 0.50, 0.99\}$. In Figure 6.15 we show the broker-dealer profit and also the number of securities chosen at optimality for the different risk profiles.

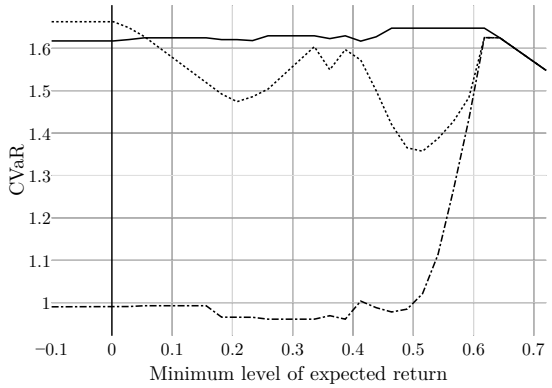
In Figure 6.16 we represent the number of securities in which each risk profile invests as a function of $E_{\min}$.



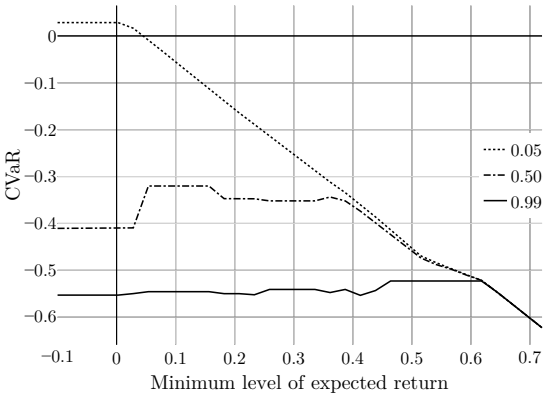
(a) CVaR_α for each profile.



(b) $\text{CVaR}_{0.05}$ for different optimal solutions.

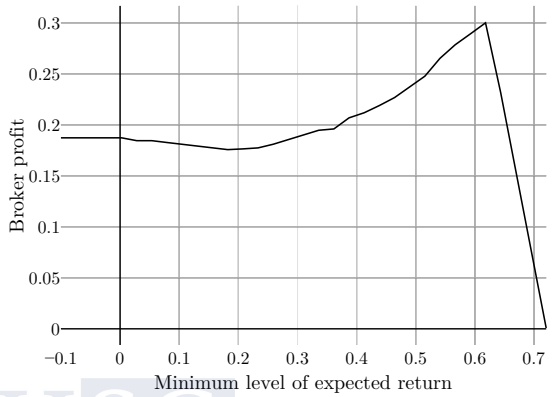


(c) $\text{CVaR}_{0.50}$ for different optimal solutions.

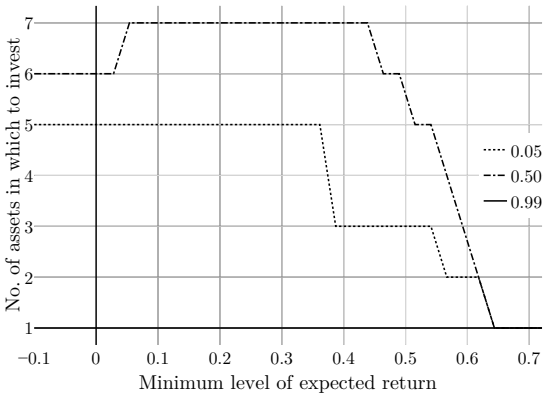


(d) $\text{CVaR}_{0.99}$ for different optimal solutions.

Figure 6.14: Comparison of CVaR_α levels at the optimal solutions of the different risk profiles.

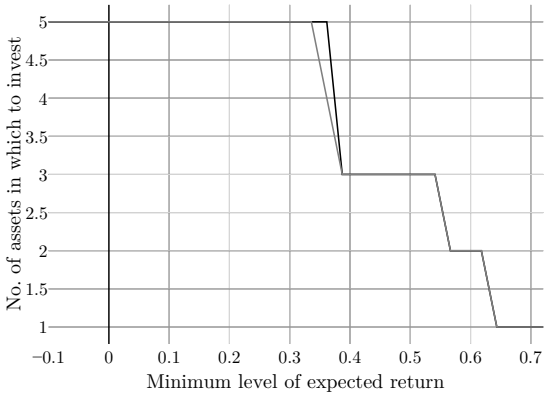


(a) Broker profit at optimality.

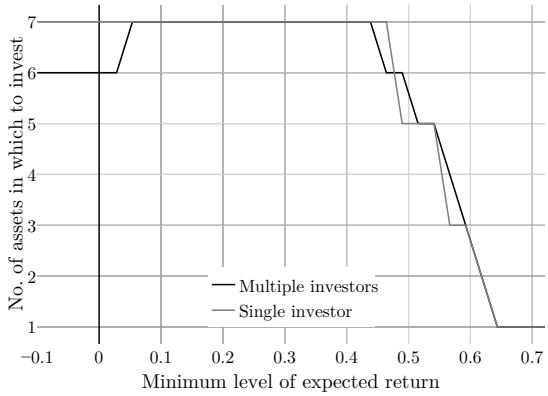


(b) Number of securities in the optimal portfolio.

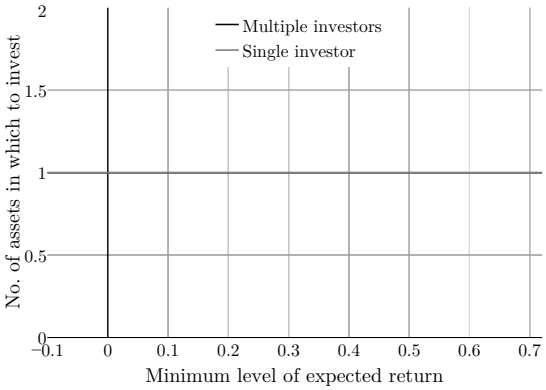
Figure 6.15: Broker profit and number of securities at optimality for different risk profiles.



(a) Securities for $\text{CVaR}_{0.05}$ risk profiles.



(b) Securities for $\text{CVaR}_{0.50}$ risk profiles.



(c) Securities for $\text{CVaR}_{0.99}$ risk profiles.

Figure 6.16: Comparison of the number of securities as a function of $E_{\min}$.

BIBLIOGRAPHY

ARTZNER, P., F. DELBAEN, J. EBER, AND D. HEATH (1997): "Thinking coherently," *Risk*, 10, 68–71.

——— (1999): "Coherent measures of risk," *Mathematical Finance*, 9, 203–227.

BAULE, R. (2010): "Optimal portfolio selection for the small investor considering risk and transaction costs," *OR Spectrum*, 32, 61–76.

BAUMANN, P. AND N. TRAUTMANN (2013): "Portfolio-optimization models for small investors," *Mathematical Methods of Operations Research*, 77, 345–356.

Bazaraa, M. S., J. J. Jarvis, and H. D. Sherali (2009): *Linear programming and network flows*, John Wiley & Sons.

Benati, S. (2003): "The optimal portfolio problem with coherent risk measure constraints," *European Journal of Operational Research*, 150, 572–584.

——— (2015): "Using medians in portfolio optimization," *Journal of the Operational Research Society*, 66, 720–731.

Benders, J. F. (1962): "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, 4, 238–252.

Bey, R. P. and K. M. Howe (1984): "Gini's mean difference and portfolio selection: an empirical evaluation," *Journal of Financial and Quantitative Analysis*, 19, 329–338.

Birge, J. R. and F. Louveaux (2011): *Introduction to stochastic programming*, Springer New York.

Byrd, R. H., J. Nocedal, and R. A. Waltz (2006): *Knitro: an integrated package for nonlinear optimization.*

Castro, F., J. Gago, I. Hartillo, J. Puerto, and J. M. Ucha (2011): "An algebraic approach to integer portfolio problems," *European Journal of Operational Research*, 210, 647–659.

Cesarone, F. (2020): *Computational finance. MatLab oriented modelling*, Routledge.

Cesarone, F., A. Scozzari, and F. Tardella (2020): "An optimization–diversification approach to portfolio selection," *Journal of Global Optimization*, 76, 245–265.

Fourer, R., D. M. Gay, and B. W. Kernighan (1990): "A modelling language for mathematical programming," *Management Science*, 36, 519–554.

González-Rodríguez, B., J. Ossorio-Castillo, J. González-Díaz, Á. M. González-Rueda, D. R. Penas, and D. Rodríguez-Martínez (2022): "Computational advances in polynomial optimization: RAPOSa, a freely available global solver," *Journal of Global Optimization*, In press.

González-Díaz, J., B. González-Rodríguez, M. Leal, and J. Puerto (2021): "Global optimization for bilevel portfolio design: economic insights from the Dow Jones index," *Omega*, 102, 102353.

GRIMM, V., T. KLEINERT, F. LIERS, M. SCHMIDT, AND G. ZÖTTL (2019): "Optimal price zones of electricity markets: a mixed-integer multilevel model and global solution approaches," *Optimization Methods and Software*, 34, 406–436.

KELLERER, H., R. MANSINI, AND M. G. SPERANZA (2000): "Selecting portfolios with fixed cost and minimum transaction lots," *Annals of Operations Research*, 99, 287–304.

KING, A. J. AND S. W. WALLACE (2012): *Modelling with stochastic programming*, Springer New York.

KOLM, P. N., R. TÜTÜNCÜ, AND F. J. FABOZZI (2014): "60 years of portfolio optimization: practical challenges and current trends," *European Journal of Operational Research*, 234, 356–371.

KONNO, H. AND H. YAMAZAKI (1991): "Mean-absolute deviation portfolio optimization model and its applications to Tokyo stock market," *Management Science*, 37, 501–623.

LABBÉ, M. AND A. VIOLIN (2016): "Bilevel programming and price setting problems," *Annals of Operations Research*, 240, 141–169.

LASSERRE, J. B. (2015): *An introduction to polynomial and semi-algebraic optimization*, Cambridge University Press.

LEAL, M., D. PONCE, AND J. PUERTO (2020): "Portfolio problems with two levels decision-makers: optimal portfolio selection with pricing decisions on transaction costs," *European Journal of Operational Research*, 284, 712–727.

MANSINI, R., W. OGRYCZAK, AND M. G. SPERANZA (2003): "On LP solvable models for portfolio selection," *Informatica*, 14, 37–62.

——— (2014): "Twenty years of linear programming based on portfolio optimization," *European Journal of Operational Research*, 234, 518–535.

MARAVILLO, H., J. CAMACHO-VALLEJO, J. PUERTO, AND M. LABBÉ (2020): "A market regulation bilevel problem: a case study of the Mexican petrochemical industry," *Omega*, 97, 102–105.

MARKOWITZ, H. (1952): "Portfolio selection," *The Journal of Finance*, 7, 77–91.

MURTAGH, B. A. AND M. A. SAUNDERS (1978): "Large-scale linearly constrained optimization," *Mathematical Programming*, 14, 41–72.

Pareto, V. (1971): *Manual of political economy*, A.M. Kelley.

Perrin, S. and T. Roncalli (2020): *Machine learning optimization algorithms & portfolio allocation*, John Wiley & Sons.

Puerto, J., M. Rodríguez-Madrena, and A. Scozzari (2020): "Clustering and portfolio selection problems: a unified framework," *Computers & Operations Research*, 117, 104891.

Qiu, R., J. Xu, R. Ke, Z. Zeng, and Y. Wang (2020): "Carbon pricing initiatives-based bilevel pollution routing problem," *European Journal of Operational Research*, 286, 203–217.

Rockafellar, R. T. (2014): "Coherent approaches to risk in optimization under uncertainty," *INFORMS Tutorials in Operations Research*, 38–61.

Rockafellar, R. T. and S. Uryasev (2000): "Optimization of Conditional Value at Risk," *Journal of Risk*, 2, 21–42.

——— (2002): "Conditional Value at Risk for general loss distributions," *Journal of Banking & Finance*, 26, 1443–1471.

Sahinidis, N. V. (2021): *BARON 21.1.13: global optimization of mixed-integer nonlinear programs*.

Shioda, R., L. Tunçel, and T. G. J. Myklebust (2011): "Maximum utility product pricing models and algorithms based on reservation price," *Computational Optimization and Applications*, 48, 157–198.

Stambaugh, F. (1996): "Risk and Value at Risk," *European Management Journal*, 14, 612–621.

Tawarmalani, M. and N. V. Sahinidis (2005): "A polyhedral branch-and-cut approach to global optimization," *Mathematical Programming*, 103, 225–249.

Valle, C. A., N. Meade, and J. E. Beasley (2014): "Absolute return portfolios," *Omega*, 45, 20–41.

von Stackelberg, H. (1934): *Market structure and equilibrium*, Springer Berlin.

Wächter, A. and L. T. Biegler (2006): "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, 106, 25–57.

WOODSIDE-ORIAKHI, M., C. LUCAS, AND J. E. BEASLEY (2013): "Portfolio rebalancing with an investment horizon and transaction costs," *Omega*, 41, 406–420.

Resumo en Galego

Esta tese doutoral céntrase no estudo dos problemas de programación polinómica e na súa resolución. Ao longo deste documento preséntase un algoritmo para a resolución dos problemas de programación polinómica e estúdanse diversas melloras do mesmo. Por unha parte, melloras clásicas como poden ser o axuste de cotas ou técnicas de programación semidefinida. Por outra parte, tamén se estudan melloras no marco da aprendizaxe estatística. Finalmente, rematamos cun capítulo no cal se presenta unha situación que se pode modelar coma un problema de programación polinómica.

A continuación, explicaremos de forma máis pormenorizada os contidos de cada un dos capítulos que conforman esta tese doutoral.

Capítulo 1: Fundamentals of the Reformulation-Linearization Technique

O principal obxectivo deste capítulo é introducir ao lector nos problemas de programación polinómica e no algoritmo basado na *Reformulation-Linearization Technique* (RLT). En primeiro lugar, comezamos coa definición formal dun problema de programación polinómica. A continuación explicamos de xeito detallado o algoritmo basado na RLT, introducindo para iso o concepto de *bound-factor constraints* así como o de variables RLT. Posteriormente, enunciamos e demostramos os principais resultados que garanten a converxencia do algoritmo basado na RLT a un óptimo global do problema de programación polinómica.

Finalmente, apoiándonos nos resultados anteriores, demostramos a converxencia do algoritmo basado na RLT cando facemos certas variacións do mesmo. En primeiro lugar, demostramos que a inclusión de cortes válidos nas relaxacións lineais non compromete a converxencia do algoritmo. En segundo lugar, definimos unha familia de regras de ramificación, e demostramos que podemos alterar a regra de ramificación orixinal mantendo a converxencia do algoritmo. Por último, enunciamos e demostramos un teorema que permite estender o algoritmo basado na RLT a problemas de programación polinómica con variables enteiras.

Capítulo 2: Enhancing the RLT-Based Algorithm: RAPOSa, a New Solver

Neste segundo capítulo, introducimos unha nova ferramenta de optimización, chamada `RAPOSa` (*Reformulation Algorithm for Polynomial Optimization Santiago*), que se basea na

RLT introducida no Capítulo 1. Ademais, facemos unha análise computacional detallada de diferentes melloras do algoritmo basado na RLT orixinal, apoiándonos para iso en `RAPOSa`.

Primeiramente, tratamos a problemática de que o algoritmo basado na RLT necesita engadir ás relaxacións lineais un certo número de *bound-factor constraints*, o cal crece de forma exponencial cando aumentan o grao e o número de variables do problema. Isto fai que as relaxacións lineais sexan demasiado grandes para problemas con poucas variables e grao baixo. Para solucionar isto, defínense os J-sets, que son un conxunto de *bound-factor constraints* que garanten que a converxencia do algoritmo mantense e, polo tanto, non é necesario engadir todas as *bound-factor constraints*. Os resultados que obtemos son altamente satisfactorios, xa que por exemplo pasamos de resolver 105 problemas a resolver 192 en 10 minutos de execución. Ademais, tanto os tempos de resolución como os gaps de optimalidade vense notablemente reducidos coa inclusión dos J-sets.

A continuación, vemos que ocorre cando utilizamos unha ferramenta de optimización non lineal local para obter cotas superiores do problema de optimización polinómica. Desta forma, facemos unha chamada cada certo tempo a unha ferramenta que devolve óptimos locais do problema de optimización polinómica. Isto permite obter cotas superiores para o óptimo do problema polinómico. Ademais, cada vez que chamamos á ferramenta de optimización non lineal local, pasámoslle como punto inicial a mellor solución obtida por `RAPOSa` ata ese momento. Os resultados obtidos mostran unha clara melloría, pasando de resolver 105 problemas a 134 problemas. Do mesmo xeito, vemos que, sen esta mellora, en 199 problemas `RAPOSa` non era capaz de devolver un gap de optimalidade despois de 10 minutos, mentres que con esta mellora o número de problemas nos que ocorre isto redúcese a 4.

Posteriormente, analizamos o impacto no rendemento de `RAPOSa` do uso de diferentes optimizadores para resolver as relaxacións lineais. Deste modo, vemos que o rendemento de `RAPOSa` mellora claramente cando o optimizador lineal utilizado é `Gurobi`. Por outra parte, observamos que tanto `Glop` como `Clp` devolven resultados similares entre eles.

Por outra parte, tamén se presenta unha versión paralela de `RAPOSa`, a cal inclúe as melloras mencionadas anteriormente. Dita versión mostra unha clara mellora no rendemento, aproveitándose dos núcleos do ordenador para resolver de forma simultánea as diferentes relaxacións lineais xeradas polo algoritmo basado na RLT.

A seguinte mellora é puramente computacional e consiste en actualizar en cada iteración soamente as *bound-factor constraints* que cambian, e non xerar todas dende cero. Os resultados mostran unha mellora no rendemento de `RAPOSa`.

A continuación presentamos unha estratexia para resolver máis rapidamente as relaxacións lineais. Esta mellora consiste en pasarlle como solución inicial a unha relaxación lineal,

a solución óptima obtida na relaxación lineal que a precede na árbore de ramificación. Tendo en conta que unha relaxación é moi similar á relaxación que a precede (soamente cambia a cota da variable de ramificación e as *bound-factor constraints* asociadas), semella razoable utilizar esta técnica. Os resultados obtidos mostran unha lixeira mellora, especialmente en problemas que son fáciles de resolver.

Na seguinte sección introducimos unha nova mellora que consiste en engadir restricións resultantes de multiplicar as restricións orixinais do problema por *bound-factor constraints*. Isto permite reducir a rexión factible das relaxacións lineais, sen deixar fóra o óptimo do problema. Neste capítulo preséntanse dúas formas de construír estas restricións. Os resultados obtidos mostran unha pequena mellora no rendemento de `RAPOSa` cando se segue unha das dúas estratexias.

Posteriormente, estudamos o impacto no rendemento de `RAPOSa` de diferentes regras de ramificación. Para isto ponderamos as violacións RLT con diferentes pesos baseados nos coeficientes do problema, nos prezos sombra tras resolver a correspondente relaxación lineal, no rango das variables ou na densidade. Ademais, pasamos de calcular a variable de ramificación considerando o máximo das violacións a calculala utilizando sumas de violacións. Dos resultados que se mostran podemos concluír que claramente hai unha gran mellora no rendemento ao empregar sumas en vez de máximos. Ademais, as regras de ramificación que utilizan tanto os prezos sombra como o rango das variables mostran un comportamento superior ao resto.

A penúltima mellora consiste na aplicación de técnicas de redución de cotas. Por unha parte, implementamos o que se coñece como FBBT (*Feasibility-based bound tightening*). Consiste en deducir cotas das variables explorando as restricións do problema, utilizando técnicas de aritmética de intervalos. Por outra parte, implementamos o OBBT (*Optimality-based bound tightening*), que consiste en resolver, para cada variable, dous problemas de optimización que nos devolven as mellores cotas posibles. Por unha parte minimízase a variable sobre a rexión factible para obter unha cota inferior e por outra parte maximízase a variable para obter así unha cota superior. Debido a isto, o OBBT devolve xeralmente mellores cotas que o FBBT, pero a cambio dun maior tempo computacional. Neste traballo expóñense diferentes combinacións do OBBT e o FBBT. O que se observa nos resultados obtidos é que estas técnicas producen unha enorme mellora no rendemento de `RAPOSa`.

Finalmente, a última mellora que se estuda neste capítulo é o uso de cortes lineais baseados en programación semidefinida con fin de reducir a rexión factible das relaxacións lineais. Para isto, construímos matrices $\boldsymbol{M}$ baseadas en produtos de variables, que pola súa propia definición teñen que ser semidefinidas positivas no problema orixinal. Posteriormente, ao resolver unha relaxación lineal, avaliamos dita matriz na solución e comprobamos se

a matriz ten algún autovector $\boldsymbol{v}$ con autovalor asociado negativo. Nese caso engadimos a restrición $\boldsymbol{v}^\top \boldsymbol{M}\boldsymbol{v} \geq 0$ á relaxación lineal. Neste capítulo presentamos diversas estratexias para implementar o anterior. Observamos novamente que a adición destes cortes lineais impacta positivamente no rendemento de `RAPOSa`.

Por último, finalizamos o capítulo comparando `RAPOSa` con outras ferramentas de optimización de referencia, tanto comerciais coma gratuítas, como son `BARON`, `Couenne` e `SCIP`. Observamos que, para problemas xerados de forma aleatoria, tanto a versión de `RAPOSa` sen melloras como a que conta con todas as melloras deste capítulo, presentan un mellor rendemento que o resto de ferramentas. Por outra banda, en problemas baseados en situacións reais, a versión sen melloras de `RAPOSa` non é competitiva co resto de ferramentas, pero a versión con melloras mostra un rendemento similar a `BARON`, `Couenne` e `SCIP`.

Capítulo 3: Learning for Spatial Branching

Neste capítulo profundamos na importancia da regra de ramificación no algoritmo basado na RLT. En primeiro lugar, presentamos novas regras de ramificación baseados en grafos. Dado un problema de programación polinómica, definimos dous grafos asociados. O primeiro deles, que denotamos por VIG (*variables intersection graph*), ten tantos nodos como variables e dous nodos están conectados mediante unha aresta se e so se as correspondentes variables aparecen xuntas nun mesmo monomio. Por outra parte, definimos o grafo CMIG (*constraints-monomials intersection graph*) da seguinte forma: consideramos un vértice asociado a cada monomio, a cada restrición e á función obxectivo. Posteriormente, creamos una aresta entre un monomio e unha restrición (ou función obxectivo) se o correspondente monomio se atopa na restrición (ou función obxectivo).

Definidos os dous grafos anteriores, consideramos diferentes medidas de centralidade asociadas a eles e de aí deducimos medidas relativas á importancia das variables no problema de programación polinómica. As medidas de centralidade consideradas son a densidade, a anchura da árbore, a modularidade, a transitividade e a centralidade de autovectores. Tras unha detallada análise das regras de ramificación introducidos no Capítulo 2 e das novas regras asociados aos grafos, seleccionamos as que demostraron ter un mellor rendemento en `RAPOSa`. Ademais observamos que non hai unha regra que sexa claramente mellor que o resto na maioría dos problemas.

Debido ao anterior, o capítulo continúa describindo técnicas de aprendizaxe estatística para determinar cal é a mellor regra de ramificación en función das características do problema. Por unha parte, definimos unha serie de características asociadas a un problema de programación polinómica como son o número de variables, o grao, a densidade, o

rango das variables, medidas de centralidade dos grafos VIG e CMIG, etc. Por outra banda, definimos unha nova KPI (*key performance indicator*) que nos permitirá avaliar o rendemento de cada unha das regra de ramificación en cada un dos problemas a resolver. O seguinte paso é presentar diferentes metodoloxías de aprendizaxe, seleccionando finalmente tres delas: modelos QGAM (*quantile generalized additive models*), modelos de *boosting* para regresión cuantil e modelos de regresión cuantil con bosques aleatorios.

Finalmente, mostramos os resultados que obtivemos tras aplicar as metodoloxías anteriores de aprendizaxe estatística. Ditos resultados mostran que, seguindo o aprendido por estas metodoloxías, o rendemento de `RAPOSa` é claramente superior ao rendemento obtido ao fixar calquera das regras de ramificación para todos os problemas.

Capítulo 4: Enhancing RLT with Conic Constraints

O obxectivo deste capítulo é presentar unha serie de restricións non lineais, que podemos engadir ás relaxacións xeradas polo algoritmo basado na RLT, co fin de reducir a rexión factible e acelerar a converxencia do algoritmo. Para definir estas novas restricións, partiremos do concepto dos cortes SDP, introducidos no Capítulo 2.

A primeira familia de restricións que tratamos son as restricións SDP. Ditas restricións consisten en impoñer a condición de que certas matrices sexan semidefinidas positivas. As matrices consideradas definímolas de forma análoga ao caso dos cortes SDP. A principal vantaxe destas restricións é que poden contribuír notablemente á redución da rexión factible das relaxacións. Sen embargo, pasamos de ter unha relaxación lineal a unha relaxación SDP, o cal aumenta xeralmente o tempo de resolución.

Como punto intermedio entre as relaxacións lineais e as relaxacións SDP, preséntanse as restricións SOC (*second-order cone*). Estas restricións non son tan fortes como as SDP, pero a cambio a resolución de problemas SOCP é xeralmente máis rápida que a resolución de problemas SDP.

Tras implementar o anterior en `RAPOSa`, o que observamos é que, na metade dos problemas que consideramos, as opcións que mostran mellor rendemento son as relacionadas con SOCP ou SDP (habendo variabilidade entre elas). Polo tanto, xorde de xeito natural a aplicación de técnicas de aprendizaxe estatística para determinar cal das estratexias é a mellor. Baseándonos na técnica de regresión cuantil con bosques aleatorios introducida no Capítulo 3, obtemos unha clara mellora no rendemento de `RAPOSa`. Por exemplo, a porcentaxe de mellora que teriamos con respecto á RLT base se escolleramos a mellor configuración en cada problema, sería do 50.5 % (é o máximo ao que podería aspirar a técnica de aprendizaxe). Os resultados obtidos, mostran unha mellora dun 32.9 % con

respecto á RLT base. Ademais, somos capaces de identificar dúas familias de problemas para as cales as versións SDP e SOCP teñen un rendemento moi superior á versión lineal. Do mesmo xeito, a técnica de aprendizaxe estatística é capaz de detectar isto e devolve moi bos resultados nestas dúas clases de problemas.

Capítulo 5: Enhanching Bound Tightening with Conic Constraints

Neste capítulo preséntase unha análise similar á do Capítulo 4, pero introducindo as restricións SDP e SOCP nos problemas de optimización que resolve o OBBT presentado no Capítulo 2. O principal obxectivo é mostrar o potencial que teñen as restricións SOCP e SDP de cara a mellorar as cotas das variables que devolve o OBBT.

Tras implementar o anterior en `RAPOSa`, o que mostran os resultados é que hai unha cantidade non depreciable de problemas (en torno ao 25 %) nos que o OBBT con restricións SDP ou SOCP presenta un rendemento superior ao OBBT estándar.

O seguinte paso é utilizar técnicas de aprendizaxe estatística análogas ás dos capítulos anteriores. Neste caso, os resultados obtidos non son tan bos como os do Capítulo 4, xa que o impacto no rendemento de `RAPOSa` é menor. Aínda así, obtense unha lixeira mellora utilizando técnicas de aprendizaxe sobre o OBBT mellorado con restricións SDP e SOCP.

Capítulo 6: Global Optimization for Bilevel Portfolio Design

O último capítulo desta tese doutoral céntrase no estudo dun problema de selección de carteiras entre brokers e inversores. Ao longo do capítulo explícase de forma detallada a situación de partida e proporciónase un modelo matemático da mesma. Dito modelo resulta ser un problema de optimización polinómica, polo que podemos resolvelo con `RAPOSa`.

A situación de partida é a que segue. Por unha banda, contamos con un inversor que ten unha carteira de posibles activos nos que investir. Por outra banda, un broker que fixa unhas comisións a cada un dos activos. Ademais traballamos con diferentes escenarios nos cales coñecemos o retorno estimado de cada un dos activos.

En primeiro lugar, definimos o problema ao que se enfronta o inversor. Para iso, discutimos brevemente sobre diferentes medidas do risco dunha inversión, quedándonos finalmente co CVaR (*Conditional Value at Risk*). Deste xeito, a situación á que se enfronta o inversor consiste en minimizar o risco que asume (é dicir, o seu CVaR), pero garantindo un mínimo retorno esperado.

A continuación, definimos o problema ao que se enfronta o broker. Dito problema consiste en maximizar os beneficios que obtén coas comisións que pon a cada activo, suxeitas ditas comisións a uns certos límites, tanto individuais como en conxunto.

Ao longo do capítulo propóñense tres modelos que representan a interacción entre o problema do inversor e o problema do broker. O primeiro deles consiste nun modelo xerárquico no cal primeiro o broker fixa as comisións correspondentes e, posteriormente, o inversor decide en que activos investir. Desta forma, o modelo matemático resultante é un modelo de programación matemática binivel. Posteriormente, utilizando conceptos de dualidade e o teorema de dualidade forte para programación lineal, transformamos o problema de optimización binivel nun problema polinómico dun só nivel.

Por outra banda, expoñemos tamén o modelo no que primeiro o inversor decide en que activos investir e, posteriormente, o broker fixa as comisións. Do mesmo xeito que no caso anterior, convertemos o modelo binivel nun problema de programación matemática polinómico dun só nivel.

Finalmente, o último modelo que presentamos é un modelo cooperativo no cal se pretende maximizar o beneficio do broker á vez que minimizar o risco que asume o inversor, garantindo este último un mínimo retorno esperado. Para resolver este modelo, o que faremos será xerar a correspondente fronteira de Pareto.

Na sección de resultados, consideramos os datos relativos ós activos do Dow Jones dende o 15 de Agosto de 2018 ata o 17 de Marzo de 2019. Realizamos diversas execucións, considerando o caso tanto con un inversor como con varios, así coma diferentes perfís de risco para os inversores e diferentes retornos esperados mínimos que se quere garantir o inversor. Tras analizar os resultados, o que observamos é que semella haber unha certa dominación do modelo no que o broker é o líder con respecto ao modelo no que o inversor é o líder, devolvendo o primeiro modelo valores Pareto eficientes.

Por último, neste capítulo tamén se inclúe un apéndice comparando a diferenza que hai ao utilizar ferramentas de optimización local fronte a ferramentas de optimización global, como é o caso de `RAPOSa`.

Further Information

Objectives

The main objectives of this thesis are as follows:

- To introduce the reader to polynomial programming problems, as well as to the RLT (Reformulation-Linearization Technique).
- To present new theoretical results in order to prove the convergence in the presence of some modifications of the baseline RLT-based algorithm.
- To implement an own version of the RLT-based algorithm, creating a solver called `RAPOSa` for this purpose.
- To make an exhaustive computational analysis of the already known enhancements of the RLT-based algorithm and other enhancements from other contexts adapted to this one.
- To present and to analyse the performance of new enhancements of the RLT-based algorithm, related to semidefinite programming, second-order cone programming and different branching rules.
- To use statistical learning techniques to predict which `RAPOSa` configuration will perform better depending on the features of a given problem.
- To present a portfolio design problem between investors and brokers, model it as a polynomial programming problem, and present some insightful results.

Methodology

The methodology followed in each of the chapters of this thesis is based on the following points:

- Firstly, an exhaustive bibliographical review. Subsequently, to think about how this can be adapted to the context of polynomial optimization.

- Secondly, to implement in `RAPOSa` everything learned in the previous point. This allows us to evaluate the performance of the new enhancements.
- Finally, to analyse the results obtained in detail and to draw conclusions.

Results

In this PhD thesis we achieved the following results:

- We have developed a new solver for polynomial optimization problems, called `RAPOSa`, and make it publicly available.
- We have presented some results to ensure the convergence of some extensions of the RLT-based algorithm.
- We have presented a detailed computational analysis of different enhancements of the RLT-based algorithm. Furthermore, we have implemented all of these enhancements in `RAPOSa`.
- We have provided new insights when using machine learning techniques in a branch-and-bound algorithm. We have presented new branching rules for the RLT-based algorithm and we have used statistical learning techniques to predict the best rule for a given instance, taking into account the features of that instance.
- We have given some insights of the use of semidefinite and second-order cone constraints in the RLT-based algorithm. Furthermore, we have integrated machine learning techniques in this framework.
- We have provided computational results in order to assess the performance of bound tightening techniques when we enhance them with semidefinite and second-order cone constraints.
- We have presented, model, and solve a portfolio design problem. We have modelled a portfolio design problem between brokers and investors as a bilevel optimization problem. Subsequently, we have used duality techniques to transform the problem into a single-level problem. Finally, we have presented a detailed analysis of the solutions obtained.

Conclusions

In Chapter 1 we have introduced the reader to polynomial programming problems, as well as to the RLT-based algorithm. Subsequently, we have presented and proved different theorems in order to ensure the convergence in the presence of some modifications of the baseline RLT-based algorithm.

In Chapter 2, we have introduced `RAPOSa`, a new global optimization solver specifically designed for polynomial programming problems with box-constrained variables. Furthermore, we have thoroughly analysed the impact of different enhancements of the RLT-based algorithm on the performance of the solver.

In Chapter 3 we have presented a framework for learning rules for spatial branching and shown in different numerical experiments that the resulting gains in performance can be significant. We have also introduced some new graph-related branching rules for the RLT-based algorithm.

In Chapter 4 we have explored different families of constraints, building upon either second-order cones or positive semidefiniteness. Moreover, we have identified specific subclasses of problems in which these constraints deliver consistently good performance. Finally, we have also shown that the potential of these conic constraints can be successfully exploited by embedding them into a learning framework.

In Chapter 5 we have discussed some enhancements related to bound tightening techniques. More precisely, we have shown that the impact of using SOCP/SDP constraints can be beneficial for bound tightening techniques. Finally, we have also shown that the potential of this using a learning framework.

In Chapter 6 we have presented a case study of polynomial optimization related to finance. First, we have presented a bilevel optimization problem between brokers and investors. Then, using duality, we have modelled this problem as a single-level polynomial optimization problem. Finally, we have presented a detailed analysis of the obtained results.

Articles and Journals

Global Optimization for Bilevel Portfolio Design: Economic Insights from the Dow Jones Index

Year: 2021.

Authors: Julio González-Díaz (University of Santiago de Compostela), Brais González-Rodríguez (University of Santiago de Compostela), Marina Leal (University

of Seville), and Justo Puerto (University of Seville).

Journal: Omega.

Publisher: Elsevier.

DOI: https://doi.org/10.1016/j.omega.2020.102353.

Journal Impact Factor: Q1 in Operations Research & Management (7/87, 92.53) in 2021.

Copyright and Use: Information about permissions can be found in the website: https://www.elsevier.com/about/policies/copyright/permissions. The following is explicitly mentioned: "Can I include/use my article in my thesis/dissertation?". "Yes. Authors can include their articles in full or in part in a thesis or dissertation for non-commercial purposes".

Contents of This Thesis Based on This Article: Chapter 6.

Contribution of the Author of This Thesis: The author of this PhD thesis has contributed to the theoretical development of the models presented in the article. Furthermore, he has also actively collaborated in the implementation of these models in `AMPL`, as well as in the analysis of the results.

Computational Advances in Polynomial Optimization: RAPOSa, a Freely Available Global Solver

Year: 2022

Authors: Brais González-Rodríguez (University of Santiago de Compostela), Joaquín Ossorio-Castillo (CITMAga), Julio González-Díaz (University of Santiago de Compostela), Ángel M. González-Rueda (University of A Coruña), David R. Penas (University of Santiago de Compostela), and Diego Rodríguez-Martínez (CITMAga).

Journal: Journal of Global Optimization.

Publisher: Springer.

DOI: https://doi.org/10.1007/s10898-022-01229-w.

Journal Impact Factor: Q2 in Applied Mathematics (80/267, 70.22) in 2021.

Copyright and Use: The answer from Springer regarding this point was "Springer Nature journal authors may reuse their article's Version of Record, in whole or in part, in their own thesis without any additional permission required, provided the

original publication is properly cited. This includes the right to make a copy of your thesis available in your academic institution's repository, or other repository required by your awarding institution".

Contents of This Thesis Based on This Article: Chapter 2.

Contribution of the Author of This Thesis: The author of this PhD thesis has contributed to the theoretical development of the enhancements of the RLT-based algorithm presented in the article. Furthermore, he has also actively collaborated in the implementation of these enhancements in `RAPOSa`, as well as in the analysis of the results.

Learning for Spatial Branching: An Algorithm Selection Approach

Year: 2022.

Authors: Bissan Ghaddar (Ivey Business School), Ignacio Gómez-Casares (University of Santiago de Compostela), Julio González-Díaz (University of Santiago de Compostela), Brais González-Rodríguez (University of Santiago de Compostela), Beatriz Pateiro-López (University of Santiago de Compostela), and Sofía Rodríguez-Ballesteros (University of Santiago de Compostela).

Journal: Submitted to INFORMS Journal on Computing. Available in arXiv.

Link: https://arxiv.org/abs/2204.10834.

Copyright and Use: Information about permissions can be found in the website: https://arxiv.org/help/license/reuse#thesis. The following is explicitly mentioned: "I want to include a paper of mine from arXiv in my thesis, do I need specific permission?". "If you are the copyright holder of the work, you do not need arXiv's permission to reuse the full text".

Contents of This Thesis Based on This Article: Chapter 3.

Contribution of the Author of This Thesis: The author of this PhD thesis has contributed to the development and implementation of the machine learning methods presented in the article. Moreover, he has implemented in `RAPOSa` the different branching rules presented in the article. Furthermore, he has also actively collaborated in the design of the executions as well as in the analysis of the results.

Polynomial Optimization: Enhancing RLT relaxations with Conic Constraints

Year: 2022.

Authors: Brais González-Rodríguez (University of Santiago de Compostela), Raúl Alvite-Pazó (University of Santiago de Compostela), Samuel Alvite-Pazó (University of Santiago de Compostela), Bissan Ghaddar (Ivey Business School), and Julio González-Díaz (University of Santiago de Compostela).

Journal: Submitted to Mathematical Programming Computation. Available in arXiv.

Link: https://arxiv.org/abs/2208.05608.

Contents of This Thesis Based on This Article: Chapter 4.

Contribution of the Author of This Thesis: The author of this PhD thesis has contributed to the development and implementation of the machine learning methods presented in the article. Moreover, he has implemented in `RAPOSa` the different conic approaches presented in the article. Furthermore, he has also actively collaborated in the design of the executions as well as in the analysis of the results.

Funding and Acknowledges

- Spanish Ministry of Education, Culture and Sport through contract FPU 17/02643.
- FEDER and Spanish Ministry of Science and Technology through the following projects: MTM2014-60191-JIN, MTM2017-87197-C3, and PID2021-124030NB-C32.
- The Supercomputing Centre of Galicia (CESGA) is acknowledged for providing the resources to run the computational experiments of this thesis.
- The author of this thesis acknowledges Gustavo Bergantiños Cid, Balbina Casas Méndez, Silvia Lorenzo Freire, Gonzalo Muñoz and Marcos Raydan for reviewing the manuscript. The author also thanks the anonymous referees who reviewed the papers on which this thesis is based.

Polynomial optimization has a wide range of practical applications in fields such as optimal control, energy and water networks, facility location, management science, and finance. It also generalizes relevant optimization problems thoroughly studied in the literature, such as mixed-binary linear optimization, quadratic optimization, and complementarity problems. As finding globally optimal solutions is an extremely challenging task, the development of efficient techniques for solving polynomial optimization problems is of particular relevance. In this thesis we provide a detailed study of different techniques to solve this kind of problems and we introduce some nobel approaches in this field, including the use of statistical learning techniques. Furthermore, we also present a practical application of polynomial optimization to finance and more specifically, portfolio design.