ESCUELA DE DOCTORADO
INTERNACIONAL DE LA USC

Ignacio
Gómez Casares

Tesis doctoral

Contributions to the design of polynomial optimization algorithms

Santiago de Compostela, 2024

Programa de doctorado en Estadística e Investigación Operativa

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

ESCOLA DE DOUTORAMENTO
INTERNACIONAL DA USC

TESIS DOCTORAL

Contributions to the design of polynomial optimization algorithms

Autor

Ignacio Gómez Casares

Directores: Julio González Díaz y Bissan Ghaddar
Tutor: Julio González Díaz

PROGRAMA DE DOCTORADO EN ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

Santiago de Compostela

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

A mis padres

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Hoy termino una etapa en mi vida en la que no puedo sino estar muy agradecido a cuantas personas me han acompañado en su transcurso. Primeramente me gustaría agradecer a mis padres todo su cariño y amor. Ellos me transmitieron su pasión por las matemáticas y no estaría aquí si no fuese por su apoyo incondicional. Gracias de todo corazón. Agradezco también a los profesores de la Facultad de Matemáticas su dedicación y empeño en enseñar y transmitir las matemáticas, en especial a Julio González Díaz, mi director de tesis, sin el que no podría haber llevado a cabo la tesis doctoral. Agradezco también a Bissan Ghaddar, mi otra directora de la tesis, el apoyo que me ha brindado estos años. A todos mis coautores, gracias por haberlo hecho tan fácil. Trabajar con vosotros ha sido una experiencia muy positiva, y he aprendido mucho de ella. También agradecer a Pietro Belotti por su acogida durante mi estancia en Milán.

No puedo sino agradecer también a todos mis compañeros de la carrera, del máster y del doctorado. He tenido el privilegio de compartir con vosotros estas etapas importantes en las que hemos crecido juntos como personas y simultáneamente hemos aumentado nuestro conocimiento científico. Especialmente agradecer a todos con los que he compartido la sala π, a Alejandra, Diego, Daniel, Héctor, Iria, Laura Freijeiro, Laura Davila, María Alonso, María Vidal, Mohamad, y en particular a Brais, con el que he colaborado estrechamente estos años. No me olvido tampoco de Ángel y Jose, compañeros fieles para el café de las 11:30.

Por último, agradecer también a todos los que en estos años (que no han sido fáciles) han estado ahí para sostenerme y acompañarme, con su oración y su compañía: a mi familia y amigos; a mis amigos más íntimos, que se han convertido en mi nueva familia compostelana; a todos con los que he compartido los miércoles en la capilla universitaria; a los carmelitas contemplativos, que con su oración y acogida han hecho más fácil estos últimos meses; a todos los sacerdotes que me han acompañado en esta etapa de formación no solo científica sino también como persona; y a Dios por todos los dones que me ha regalado y por los que he llegado a este momento en mi vida. A todos, ¡gracias!

Ignacio Gómez Casares
Santiago de Compostela, 23 de julio de 2024, Santa Brígida de Suecia

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Contents

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Introduction

In the effort of developing algorithms for solving nonlinear optimization problems, most of the research is aimed at solving general nonlinear problems. However, there is a subclass of problems, polynomial optimization problems, that it's relevant as it provides an additional structure to the problems while still covering classes of problems often studied, such as continuous convex and nonconvex problems with quadratic costs and constraints or binary linear problems. In this thesis, we study this field of polynomial optimization and focus on three relevant aspects: solving the problem, using learning techniques to improve the performance of a solver and applying polynomial optimization techniques to a real-world problem in power network optimization.

Part 1 of the dissertation is aimed at studying one technique for solving polynomial optimization problems, namely the Reformulation-Linearization Technique, hereafter referred to as RLT. This technique has been used by the researchers at the University of Santiago de Compostela and CITMAga to develop a general solver for polynomial optimization problems called RAPOSa. Although the base for the chapters in this part is the RLT technique, the conclusions extracted can be applied to different techniques based on the same fundamental building blocks as the RLT. Nonetheless, all the numerical analysis have been developed using RAPOSa and the RLT technique, and used to test the impact different enhancements have in an algorithm designed to solve polynomial optimization problems. In Chapter 1 a formal introduction to the polynomial optimization class of problems is made and the RLT technique along its implementation in RAPOSa is described. There is also a discussion of some already studied enhancements to the RLT technique that are used in subsequent chapters, as well as a summary of the performance metrics used in the thesis. Chapter 2 studies the impact of domain reduction techniques in a spatial branching algorithm, in particular focuses on bound-tightening and the selection of the branching point. Chapter 3 analyzes how a seemingly simple decision such as imposing some bounds to the auxiliary RLT variables can change the performance of the algorithm depending on the linear solver chosen for the linear problems in a spatial branching context. Finally, Chapter 4 extends the RLT technique to solve polynomial optimization problems with unbounded variables, and discusses the theoretical guarantees of optimality of the obtained solution.

Part 2 enters the field of statistical learning and its potential to improve the performance of optimization algorithms. Machine learning has always been related to optimization, since the use of optimization algorithms is at the core of the different learning techniques. However, there has been a recent increase in research looking for ways in which machine learning can help optimization algorithms. In particular, one approach is using ML for the selection between different optimization algorithms or configurations of an algorithm depending on the problem at hand,

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

since there may be performance differences between the available options as a consequence of the characteristics of the problem itself. In Chapter 5 a general learning framework aimed at this selection is presented, and its performance is analyzed in the case of selecting a branching rule for a spatial branching algorithm, selecting between different solvers, as well as selecting the best combination of the different enhancements presented in Chapter 2 and Chapter 3. This selection is made based only on the characteristics of the optimization problem, without including anything about how the algorithm is performing while solving the problem, i.e., its done on a "static" fashion and this allows its application to general settings such as the already mentioned selection between different optimization solvers. In Chapter 6 we focus our effort on checking if, for the particular case of spatial branching for polynomial optimization, including information extracted during the execution of the algorithm and "dynamically" changing the decision made by the learning can have an impact and improve upon the "static" approach in Chapter 5.

Finally, Part 3 is devoted to studying the Security-Constrained Unit Commitment problem, hereafter the SCUC problem, a relevant optimization problem in the field of power networks management. Although Part 3 is somehow disconnected from Part 1 and Part 2, the resulting optimization problem is polynomial and choosing this SCUC problem as part of the research objectives for this thesis is not casual, since the aim is to be able to adapt RAPOSa in the future to be able to handle the SCUC problem and have specific enhancements included for this problem. In the meantime, we present a framework to solve the SCUC problem and obtain solutions with optimality guarantees. In Chapter 7 we describe the basic building blocks needed to define the SCUC problem, since it is a combination of the Optimal Power Flow and the Unit Commitment problems. We also discuss some of the techniques used to solve these problems, as we use them in Chapter 8 to describe the proposed framework. Also in Chapter 8 we include a description of the computational setup used and the numerical results obtained.

Appendix A describes the computational environment to manage the executions developed by Brais González Rodríguez and the author of this dissertation to streamline the process of obtaining the different numerical results in this thesis. Appendix B includes the proofs of the results in Chapter 1.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Methodology

As already mentioned in the introduction, this thesis has three clearly differentiated parts. Part 1 deals with the RLT technique and its implementation in RAPOSa. Here, the main tool used for the different chapters is the solver RAPOSa, used for the different executions and obtain the numerical results presented. All the executions were made in the environment described in Appendix A. Also, the implementation of the different enhancements of the RLT technique was made after an exhaustive bibliographical review of the already existing techniques in the literature.

Part 2 discusses the use of machine learning techniques to improve the performance of an algorithm like the RLT technique implemented in RAPOSa. Here, we employ the numerical results obtained using RAPOSa to train a machine learning method using a particular learning technique. From there, using the environment described in Appendix A, we compare the performance of this "learned" configuration with the base performance of RAPOSa, and analyze its impact. This also requires an exhaustive bibliographical review.

Finally, in Part 3, we tackle one of the problems used in the optimization of power networks: the SCUC problem. To define the problem a literature review was made, to select the most appropriate formulation of the problem for the scheme we planned to use to solve the problem. Also, an extensive study of the API from Mosek was required to implement the algorithm described in Section 8.4, as well as an study of the package from [132] and its use in Julia and the modelling languages Pyomo [21, 65] in Python and JuMP [86] in Julia.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Obtectives

Part 1 of this thesis focuses on studying the methods to solve a polynomial optimization algorithm, focusing specifically on the RLT technique. In particular, the objectives of this part are the following:

- Introduce the formal definition of a polynomial optimization problem, as well as the notation that is used in the thesis.
- Describe the Reformulation-Linearization Technique, the building block of several chapters in the thesis, as a specific algorithm to solve polynomial optimization problems to global optimality.
- Present some enhancements of the RLT technique, some of which are discussed in Chapter 2 and Chapter 3.
- Introduce RAPOSa, a global optimization solver for polynomial optimization problems based on the RLT technique, and describe the specifics of its implementation.
- Describe the different performance metrics used to present the numerical results in the thesis.
- Study the impact of different enhancements that can be introduced in domain reduction techniques, in particular bound-tightening techniques and the selection of the branching point in a spatial branching algorithm.
- Study the effect of sending to the linear solver bounds for the auxiliary variables in the RLT technique depending on the linear solver chosen for the branch-and-bound.
- Develop the theory that allows for the extension of the RLT technique to solve optimization problems with unbounded variables, and proof the theoretical guarantees of optimality for the obtained solution.

Part 2 builds upon what was done in Part 1, since machine learning techniques are used to select between the different enhancements described depending on the particular characteristics of the problem that needs to be solved. The aims of the part are:

- Formally introduce the pace, a performance metric that generalizes time and gap and allows for a simultaneous comparison of solved and unsolved instances, crucial for the use of any learning technique.
- Introduce a learning framework to select between different solvers or configurations of the same solver depending on features computed on the optimization problem at hand.
- Study the performance of the learning framework for choosing between branching rules (configurations) and between different solvers.
- Apply the learning framework to select between the enhancements described in Part 1, to further improve their impact.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- Analyze if incorporating some "dynamic" aspect to the learning can lead to better performance of the framework in the context of a spatial branching algorithm.

Part 3 discusses three problems usually employed in the power networks optimization field. The study of these problems is not disconnected from Part 1 and Part 2, since they are polynomial optimization problems and the motivation for including this two chapters in the thesis is for RAPOSa to be able to tackle them in the future. For the time being, we study the properties and solution methods for these problems using an ad-hoc framework designed to solve them. The objectives of the part are:

- Summarize the basic concepts in electrical circuits used subsequently in the definition of the optimization problems.
- Introduce the Optimal Power Flow and the Unit Commitment optimization problems, building blocks of the main subject of study, the Security-Constrained Unit Commitment problem.
- Define the SCUC problem and the formulation employed in our case.
- Describe the framework developed to solve the SCUC problem and obtain solutions with theoretical guarantees of optimality.
- Present some numerical results and the computational setup used.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Part 1

Solving Polynomial Optimization Problems

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

List of Symbols

Sets

N	variables
M	constraints
M_j	constraints involving variable j
B	bounded variables, i.e., with a finite upper bound
U	unbounded variables, i.e., without a finite upper bound
$\mathcal{Q}$	queue of problems
$\mathcal{N}^k$	variables for the hyperrectangle expansion phase
U^k	unbounded variables at node k

Variables, vectors and matrices

X_J	RLT variable associated to multiset J
$\bar{\boldsymbol{x}}$	values of variables in solution
$\bar{\boldsymbol{\lambda}}$	values of dual variables in solution
$\boldsymbol{x}^{best}$	current best solution
$\bar{\boldsymbol{x}}^k$	values of variables in solution at node k
$\bar{\boldsymbol{X}}^k$	values of RLT variables in solution at node k
$\mathcal{S}^n$	space of symmetric matrices of order n
$\mathcal{S}^n_+$	positive semidefinite matrices
$\mathcal{Q}^n$	second-order cone
$\mathcal{Q}^n_r$	rotated second-order cone

Functions

ϕ_r	polynomial
$F_\delta(J_1, J_2)$	bound-factor constraint for multisets J_1 and J_2
$[\cdot]_L$	linearization function
$\nu[PP(\Omega)]$	objective function of $PP(\Omega)$

Values

b_r	right hand side of constraint r
δ_r	degree of a polynomial r
δ	degree of a problem
l_j	lower bound of variable j
u_j	upper bound of variable j
LB	lower bound of algorithm
UB	upper bound of algorithm
LB^k	potencial of node k
l^k_j	lower bound of variable j at node k

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

u_j^k	upper bound of variable j at node k
$\bar{z}^k$	optimal value of objective function at node k
θ_j^k	branching score for variable j at node k
Θ_j^{vt}	violation transfer score for variable j
$w(j, J)$	weight for variable j and multiset J in branching rule
β	branching point for variable
Λ	global parameter for the hyperrectangle expansion phase

Counters and references

σ^k	current reference node for node k
γ^k	depth of node k
τ	counter for the number of problems

Other

Ω	hyperrectangle containing the feasible region of $PP(\Omega)$
$\dot{\Omega}$	hyperrectangle containing the feasible region of $PP(\dot{\Omega})$
Ω^k	hyperrectangle containing the feasible region of node k
$\dot{\Omega}^k$	hyperrectangle containing the feasible region of node k
(N, μ)	multiset
$\lvert(N, \mu)\rvert$	cardinality of a multiset
N^λ	the collection of multisets $\{(N, \mu)$ s.t. $\lvert(N, \mu)\rvert = \lambda\}$
N_-^λ	the collection $\{(N, \mu)$ s.t. $1 \leq \lvert(N, \mu)\rvert < \lambda\}$
$\text{supp}((N, \mu))$	the support of a multiset, $\{j \in N$ s.t. $\mu(j) > 0\}$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Objectives

Part 1 of this thesis focuses on studying the methods to solve a polynomial optimization algorithm, focusing specifically on the RLT technique. In particular, the objectives of this part are the following:

- Introduce the formal definition of a polynomial optimization problem, as well as the notation that is used in the thesis.
- Describe the Reformulation-Linearization Technique, the building block of several chapters in the thesis, as a specific algorithm to solve polynomial optimization problems to global optimality.
- Present some enhancements of the RLT technique, some of which are discussed in Chapter 2 and Chapter 3.
- Introduce RAPOSa, a global optimization solver for polynomial optimization problems based on the RLT technique, and describe the specifics of its implementation.
- Describe the different performance metrics used to present the numerical results in the thesis.
- Study the impact of different enhancements that can be introduced in domain reduction techniques, in particular bound-tightening techniques and the selection of the branching point in a spatial branching algorithm.
- Study the effect of sending to the linear solver bounds for the auxiliary variables in the RLT technique depending on the linear solver chosen for the branch-and-bound.
- Develop the theory that allows for the extension of the RLT technique to solve optimization problems with unbounded variables, and proof the theoretical guarantees of optimality for the obtained solution.

The content is divided into four chapters. Chapter 1 deals with the preliminaries, i.e., the introduction of the RLT technique and its enhancements, the solver RAPOSa and the performance metrics used. Chapter 2 deals with the domain reduction improvements and their impact on a spatial branching algorithm. Chapter 3 studies the interaction with the linear solver and how it can affect the performance of a branch-and-bound based algorithm to solve polynomial optimization problems. And finally, Chapter 4 extends the RLT technique to problems with unbounded variables and proves its convergence.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 1

Preliminaries: Polynomial Optimization, RLT and RAPOSa

1.1. Introduction

Part 1 and Part 2 of this thesis are aimed at studying different aspects of polynomial optimization from the algorithm design point of view. Although our goal is not to restrict ourselves to a particular technique and talk about polynomial optimization in general, unavoidably some particular algorithm needs to appear from time to time, if not to have some numerical results. This algorithm that shows up is the RLT technique, introduced by Sherali and Tuncbilek in 1992 [120].

We say "RLT" referring ourselves to the algorithm based on RLT to solve polynomial optimization problems. The RLT technique itself was formally introduced in [117, 116] to solve linear problems with binary variables, and was later on extended to the case of continuous nonlinear problems [115]. The context at that time was, as mentioned in [120], that apart from literature aimed at tackling special cases of polynomial optimization problems, "polynomial programs have not received much attention". One of the first mentions of the use of branch-and-bound methods to solve general nonlinear problems is [79], but it was not until some years later when this technique started to be used in the context of polynomial optimization problems. In [68] a general branch-and-bound algorithm is presented to solve global optimization problems. Then [69] describes several promising ways to solve global optimization problems which include the polynomial optimization ones.

[120] introduces the RLT technique, an algorithm designed to solve polynomial optimization problems to global optimality. Other works such as [118, 119, 31, 30] refined that original proposal. The original technique is described in Section 1.2 and some of the refinements are also mentioned in this chapter. Both the RLT technique and some of the enhancements are implemented in RAPOSa [59], an RLT-based solver developed at the University of Santiago de Compostela and CITMAga that has become a competitive solver for polynomial optimization problems. RAPOSa is the solver we use throughout this thesis to obtain numerical results in several chapters. As this solver was developed in-house, we can modify the code, which represents an ideal environment to test the impact a change in an algorithm designed to solve nonlinear problems is impacted by these changes. RAPOSa is formally introduced in Section 1.8.

The aims of this chapter are: to present formally what is a polynomial optimization problem and show how the RLT technique works, in Section 1.2; discuss several important aspects of the technique that can or have already been improved by the introduction of modifications to the original algorithm, in Sections 1.3, 1.4, 1.5 and 1.6; summarise the convergence results proven in [120], in Section 1.7;

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

and present the solver RAPOSa, in Section 1.8. In Section 1.9 the computational environment used in Part 1 and Part 2 is also described.

1.2. The Reformulation-Linearization Technique

The *reformulation-linearization technique* or *RLT* was originally introduced in [120]. It was designed to find global optima in (continuous) polynomial optimization problems given by:

$$PP(\Omega) \qquad \begin{array}{lll} \text{minimize} & \phi_0(\boldsymbol{x}) & \\ \text{subject to} & \phi_r(\boldsymbol{x}) \geq b_r, & r = 1, 2, \ldots, m_1 \\ & \phi_r(\boldsymbol{x}) = b_r, & r = m_1 + 1, \ldots, m \\ & \boldsymbol{x} \in \Omega \subset \mathbb{R}^{|N|}, & \end{array}$$

where N denotes the set of variables, each $\phi_r(\boldsymbol{x})$ is a polynomial of degree $\delta_r \in \mathbb{N}$ and the region $\Omega = \{\boldsymbol{x} \in \mathbb{R}^{|N|} : 0 \leq l_j \leq x_j \leq u_j < \infty, \text{ for all } j \in N\} \subset \mathbb{R}^{|N|}$ is a hyperrectangle containing the feasible region. Then, $\delta = \max_{r \in \{0,\ldots,m\}} \delta_r$ is the degree of the problem.

Next, in order to formally define the linear relaxation of $PP(\Omega)$, it is convenient to formally introduce the notion of *multiset*. Given a finite set N and a map $\mu : N \to \mathbb{N} \cup \{0\}$, a multiset is a pair (N, μ), where, for each $j \in N$, $\mu(j)$ denotes j's multiplicity. The *cardinality* of a multiset is given by $|(N, \mu)| := \sum_{j \in N} \mu(j)$. We define the *sum* of two multisets defined over N as $(N, \mu_1) + (N, \mu_2) := (N, \mu)$ where $\mu(j) = \mu_1(j) + \mu_2(j)$ for all $j \in N$. We say that a multiset (N, μ_1) is *included* in (N, μ_2), denoted by $(N, \mu_1) \subseteq (N, \mu_2)$ if $\mu_1(j) \leq \mu_2(j)$ for all $j \in N$. Given a set N and $\lambda \in \mathbb{N}$, we denote by N^λ the collection of multisets $\{(N, \mu) \text{ s.t. } |(N, \mu)| = \lambda\}$, and by $N^\lambda_\leq$ the collection $\{(N, \mu) \text{ s.t. } 1 \leq |(N, \mu)| < \lambda\}$. The *support* of a multiset (N, μ), $\text{supp}((N, \mu))$, is the set $\{j \in N \text{ s.t. } \mu(j) > 0\}$.

The RLT is based on the construction of a linear relaxation of the polynomial problem, which is then embedded in a branch-and-bound scheme. This is accomplished by replacing each monomial of the problem with a corresponding RLT variable. For example, associated with a monomial of the form $x_1 x_2 x_4$ one would define the RLT variable X_{124}. More generally, RLT variables are defined as

$$X_J = \prod_{j \in J} x_j, \tag{1}$$

where J is a multiset containing the information about the multiplicity of each variable in the underlying monomial. Then, at each node of the branch-and-bound tree, the corresponding linear relaxation is solved. In order to get tighter relaxations and ensure convergence, additional constraints, called *bound-factor constraints*, are also added (and linearized). These constraints are given for each pair of multisets J_1 and J_2 such that $J_1 + J_2 \in N^\delta$, by

$$F_\delta(J_1, J_2) = \prod_{j \in J_1} (x_j - l_j) \prod_{j \in J_2} (u_j - x_j) \geq 0, \tag{2}$$

being δ the degree of the problem.

[31] identifies 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. We adopt this approach since it can dramatically reduce the size of the resulting relaxation while preserving most

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

of its tightness, as shown in [31, 59]. We discuss J-sets a bit further in Section 1.3 but for now we write everything without using them to simplify notation.

With all of the above and denoting by $[\cdot]_L$ the linearization of a polynomial obtained by substituting its monomials by their associated RLT variables, the linear relaxation of $PP(\Omega)$ has the following form:

$$
LP(\Omega) \quad \begin{aligned}
&\text{minimize} && [\phi_0(\boldsymbol{x})]_L && \\
&\text{subject to} && [\phi_r(\boldsymbol{x})]_L \geq b_r, && r = 1, 2, \ldots, m_1 \\
& && [\phi_r(\boldsymbol{x})]_L = b_r, && r = m_1 + 1, \ldots, m \\
& && [F_\delta(J_1, J_2)]_L \geq 0, && J_1 + J_2 \in N^\delta \\
& && \boldsymbol{x} \in \Omega \subset \mathbb{R}^{|N|}. &&
\end{aligned}
$$

Importantly, if the RLT defining identities in Eq. (1) are added to $LP(\Omega)$, then it becomes equivalent to $PP(\Omega)$. Note that, for the sake of exposition, we are slightly abusing notation, since the bound-factor constraints depend on the bounds in Ω. Thus, changes in Ω imply changes not only in the bounds of the problem, but also in the bound-factor constraints.

Branching is performed as follows. At each node, once the corresponding relaxation has been solved, a score is assigned to each variable depending on the violations of the RLT identities in Eq. (1) in which it is involved. Then, the variable with the highest score is chosen for branching, splitting the interval between its lower and upper bound at its optimal value at the current node. As we will see, both the criteria for selecting the variable and the selection of the branching point can be susceptible to change giving rise to different variations of the baseline algorithm.

In Figure 1.1 there is the pseudocode of the RLT algorithm `plainRLT`. In the rest of this chapter, we analyze specific parts of the RLT, focusing for example on some specific aspects of the branching, and we also show the fundamental ideas that ensure the converge of the algorithm. This analysis will be relevant for the following chapters of the thesis.

1.3. J-sets

Recall that $LP(\Omega)$ contains, not only the linearization of the polynomials in $PP(\Omega)$ but also the linearization of the bound-factor constraints. This is done to ensure convergence to the global optimum, as can be seen in the proof of convergence in [120]. The downsize of this approach is that when the number of variables increases, the number of posible monomials built from combination of variables increases exponentially. This is why, in [31], the authors try to reduce the number of these auxiliary constraints, so that convergence is still guaranteed but the number of added constraints is more manageable.

As far as this thesis is concerned, there are two noteworthy results in [31]. 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.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Initialization. Let $LB := -\infty$ and $UB := +\infty$ be the lower and upper bounds of the algorithm. Let $\Omega^1 := \Omega$ be the root node's hyperrectangle; $LB^1 := \infty$ the root node's potential; $Q := \{1\}$ the queue of problems to solve; $\tau := 1$ a counter for the problems; and $\boldsymbol{x}^{best} := \emptyset$ the current best solution.
For $j \in N$ and $k \in \mathbb{N}$, let l_j^k and u_j^k denote the lower and upper bound of variable x_j in $LP(\Omega^k)$.
Stage 1 ***(main).*** Choose $k \in Q$ such that $LB^k = \min_{s\in Q} LB^s$. Let $Q := Q\backslash\{k\}$. Solve problem $LP(\Omega^k)$.

- If $LP(\Omega^k)$ is infeasible, go to **Stage 2**.
- If $LP(\Omega^k)$ is feasible, an optimal solution, $(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k)$, and an optimal value of the objective function, $\bar{z}^k$, are obtained. Compute θ_j^k for $j \in N$ as in Eq. (3).
  - If $\bar{z}^k < UB$ and $\theta_j^k = 0$ for all $j \in N$:
    - ◇ *Update upper bound.* $UB := \bar{z}^k$. Let $\boldsymbol{x}^{best} := \bar{\boldsymbol{x}}^k$.
    - ◇ *Prune.* Remove all $s \in Q$ such that $LB^s \geq UB$.
    - ◇ Go to **Stage 2**.
  - If $\bar{z}^k < UB$ and $\theta_j^k > 0$ for some $j \in N$:
    - ◇ *Select branching variable.* Choose $p \in N$ such that $\theta_p^k = \max_{j\in N} \theta_j^k$. Branch at $\beta := \bar{x}_p^k$.
    - ◇ *Branch.* Let
    $$\Omega^{\tau+1} := \Omega^k \cap \{\boldsymbol{x} \in \mathbb{R}^{|N|} \text{ s.t. } l_p^k \leq x_p \leq \beta\},$$
    $$\Omega^{\tau+2} := \Omega^k \cap \{\boldsymbol{x} \in \mathbb{R}^{|N|} \text{ s.t. } \beta \leq x_p \leq u_p^k\}.$$
    - ◇ Update queue $Q := Q \cup \{\tau+1, \tau+2\}$ and $\tau := \tau + 2$. The potentials of the new nodes are $LB^{\tau+1} = LB^{\tau+2} := \bar{z}^k$.
    - ◇ Go to **Stage 2**.
  - If $\bar{z}^j \geq UB$. *Prune branch.* Go to **Stage 2**.

Stage 2 ***(control).*** *Update lower bound.* $LB := \min\{\min_{k\in Q} LB^k, UB\}$.

- If $LB = UB$, END:
  - If $\boldsymbol{x}^{best} = \emptyset$, $PP(\Omega)$ is infeasible.
  - If $\boldsymbol{x}^{best} \neq \emptyset$, $\boldsymbol{x}^{best}$ is a global optimum and UB is the optimal value.
- In other case, go to **Stage 1**.

FIGURE 1.1. Algorithm `plainRLT`.

Using the above results, a subset of monomials called *J-sets* can be identified, such that they are sufficient to guarantee the convergence of the algorithm. J-sets correspond with the maximal multisets associated with the monomials in $PP(\Omega)$. Their use allows to reduce the size of the relaxation (even though they're still a big number of extra constraints, that still grows exponentially fast as the number of variables and monomials in the problem increases) but has the drawback that the relaxation is less tight than the original one proposed, so more iterations may be required for the algorithm to converge. Nevertheless, [31, 59] showed that, in practice, the use of J-sets is preferable.

1.4. Bound tightening

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Domain reduction is a core element of global optimization algorithms for non-convex optimization problems. As can be seen in [108, 126, 7, 105], domain reduction encompasses a wide variety of techniques, which may concern different stages

of a (spatial) branch-and-bound algorithm. In particular, it may involve branching decisions such as the criteria to select the branching variable or the branching point (which we discuss in Section 1.5) or approaches to tighten the bounds of the variables and reduce the search space. We deal with both in this work, and we ilustrate them in the context of the RLT, but the concepts involved can and are applied to most global optimization techniques.

Regarding the tightening of the bounds of the variables, although the definition of $PP(\Omega)$ requires to have bounds for the variables, they might be unnecessarily large and lead to a hyperrectangle Ω including many infeasible points. *Bound tightening (BT)* refers to a series of techniques used to reduce the size of any such hyperrectangle by removing infeasible parts of it. There are two main approaches: *optimality-based bound tightening (OBBT)* and *feasibility-based bound tightening (FBBT)*.

OBBT techniques typically solve two optimization problems for each variable, just by substituting the objective function with the variable at hand and then minimizing and maximizing the resulting optimization problem. For example, given optimization problem $PP(\Omega)$, we would define, for variable j the associated minimization problem to obtain a lower bound:

$$\begin{aligned} \text{minimize} \quad & x_j \\ \text{subject to} \quad & \phi_r(\boldsymbol{x}) \geq b_r, \qquad r = 1, 2, \ldots, m_1 \\ & \phi_r(\boldsymbol{x}) = b_r, \qquad r = m_1 + 1, \ldots, m \\ & \phi_0(\boldsymbol{x}) \leq UB \\ & \boldsymbol{x} \in \Omega \subset \mathbb{R}^{|N|}, \end{aligned}$$

where UB is the best upper bound available for the problem. We would do the same for all variables, and also define and solve the maximization problems as well. This procedure gives the best possible bounds for each variable given the constraints and the bounds of the other variables. The downside is that these subproblems may be as hard to solve as the original problem, so the process is often computationally prohibitive. Because of this, OBBT is normally applied to relaxations of the original problem, easier to solve, or letting it run only for a limited amount on time. Even if relaxations are used for OBBT, it can still be computationally demanding, which motivates that OBBT is often used only at the root node or at a few selected nodes of the algorithm.

FBBT, on the other hand, uses constraint propagation techniques to tighten the bounds and, since no optimization problem is solved in the process, the computational cost is typically so small that FBBT is usually applied at every node of the branch-and-bound tree. There are different ways of implementing the FBBT. One of the possible implementations works as follows. Given the constraint $2x + 0.7y \leq 3$, and the bounds $-2 \leq x \leq 2$ and $0 \leq y \leq 4$, the first step computes new bounds for the constraint using the bounds of the variables and interval arithmetic, leading to $-4 \leq 2x + 0.7y \leq 6.8$. Then, in the second step, these bounds are propagated back to the variables, and intersected with the original ones. For variable x, we obtain bounds $-3.4 \leq x \leq 1.5$ that combined with the original ones lead to a reduction in the upper bound from 2 to 1.5.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

1.5. Branching

One of the most important steps in a branch-and-bound algorithm, and therefore in the RLT algorithm, is the selection of the branching variable and the branching point for that variable. Changing any of these branching decisions can have a big impact on the resulting tree and consequently on the speed of convergence for that particular instance. We now present the original branching decision rule proposed in [120], and also some alternatives proposed in [118, 119]. Further, we also discuss some generalizations from [59, 54] that we use later on.

In Sherali's original work, [120], the proposed approach was, in a node k with optimal solution $(\bar{\boldsymbol{X}}^k, \bar{\boldsymbol{x}}^k)$, to compute the violation of each variable as:

$$\theta_j^k = \max_{J \in N_-^\delta} \{|\bar{X}^k_{J\cup\{j\}} - \bar{x}_j^k \bar{X}_J^k|\}. \tag{3}$$

Then, one of the variables that maximizes this value is chosen for branching and the branching point is chosen to be the optimal value of that variable in the optimal solution $\bar{\boldsymbol{x}}^k$.

One natural modification of Eq. (3) is to change the maximum and calculate the sum of the violations instead, as proposed in [118, 119]. Moreover, as discussed in [59, 54], one can also put an aditional weight to the violations, obtaining the following formula:

$$\theta_j^k = \sum_{J \in N_-^\delta} w(j,J) \cdot |\bar{X}^k_{J\cup\{j\}} - \bar{x}_j^k \bar{X}_J^k|, \tag{4}$$

where $w(j,J)$ are the weights associated with the violations. There are different ways these weights can be defined. We do not pretend to do an exhaustive exploration of this but just describe here the ones we will use in this dissertation. Before that we need to introduce some definitions.

1.5.1. Graphs associated to polynomial problems.

The relevance of using graph representations of general optimization problems has long been recognized and, as discussed in [12] and [37], polynomial optimization problems are not an exception. We introduce here two graphs associated to a polynomial problem as discussed in [12, 37, 54].

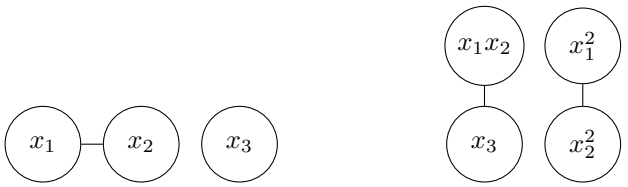
For a given polynomial optimization problem, one can associate different graphs that capture some characteristics of its structure. Here, to any $PP(\Omega)$ problem, we associate two different graphs, both based on the concept of intersection graphs as defined in [48]. The first one is the *variables intersection graph, VIG*, analogous to the one used in [12]. 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). We include here an example to illustrate the construction of these graphs. Consider the following optimization

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

problem:

$$\begin{aligned} &\text{minimize} && x_1x_2 + x_3 \\ &\text{subject to} && x_1^2 + x_2^2 \leq 10 \\ &&& x_1 \geq 0, x_2 \geq 0, x_3 \geq 4. \end{aligned} \tag{5}$$

The graphs associated to problem (5) are displayed on Figure 1.2.



(A) VIG graph (B) CMIG graph

FIGURE 1.2. Graphs associated to problem 5

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 [77]).

- **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. It measures how close a graph is to being a complete graph. A higher density means the nodes in the graph are more connected.
- **Treewidth**. Given a graph G, the *treewidth* is the smallest width of any tree-decomposition of G (see [107]). It gives a measure of how close a graph is to a tree. Trees have a treewidth of one and complete graphs with k nodes have a treewidth of $k-1$. To estimate the treewidth we use the min-fill heuristic [13].
- **Modularity**. Given a partition of a graph G, the *modularity* of the partition is a measure that tries to assess how "good" it is, in the sense that a high modularity means that the graph has dense connections between the nodes within each set in the partition but sparse connections between nodes in different sets. The modularity of G is the maximum modularity over all the possible partitions of the graph. Since the exact computation of this parameter is a hard computational problem, we use the greedy algorithm in [27] 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 (groups of 3 connected nodes) in G. It gives relevance to links between nodes which are both adjacent to another common node.
- **Eigenvector centrality or eigencentrality**. For each node in a graph, its *eigencentrality* score is the corresponding value in the first eigenvector of the graph adjacency matrix. The centrality of a node is then proportional to the sum of the centralities of its neighbors.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

1.5.2. Reliability branching. In the context of MILP problems, there is a branching approach that is considered to be the "best" in terms of the size of the resulting tree. This approach is called *strong branching* and consists in branching, at each node, on all problem variables and then select the one that leads to a larger improvement on the lower bound. The drawback with strong branching is the computational overhead of branching on all variables in each node, and there is a lot of research devoted to imitate its performance but without the associated computational overhead. Leaving aside that strong branching may not be the best approach to imitate, something we briefly discuss in Chapter 6, we introduce the concept of *reliability branching* that was designed to imitate strong branching, as described in [1].

Reliability branching builds upon another approach to imitate strong branching called *pseudocost branching*. The idea behind it is to store the "performance" of previous branchings on each of the variables, and use that past "performance" to decide the branching variable in the current node. The issue with this approach is that at the beginning of the tree there is no past information available since most variables have not been used as branching variables yet. To overcome this issue, a hybrid between strong branching and pseudocost branching is used, where strong branching is employed in the tree up to a given depth is reached, and from there pseudocost branching is used. The improvement reliability branching brings to this hybrid approach is to define a "reliability" indicator for each variable, so strong branching is used until the reliability of the pseudocost is "good enough", and from that point onward the pseudocost for that variable is used.

In this thesis, we build upon the approach described in [7] to adapt reliability branching to the context of MINLP problems. The weights in Eq. (4) are modified to incorporate the value of the pseudocosts, but only after they are deemed reliable:

- We start with a selected branching criteria that puts some weights $w(j, J)$ (one that does not have reliability in it; we discuss some possibilities in Section 1.5.4) and we branch as usual.
- After branching, we store the improvement in the lower bound for that variable, and compute an average over all previous improvements, obtaining the pseudocost for that variable.
- Once a variable has been chosen for branching a number of times chosen by the user, we deem its pseudocost as "reliable", and start incorporating it into the weight in Eq. (4). As we already have some weights given by the criteria, we just multiply them by the pseudocost. At the beginning we can consider all the pseudocosts as one, but once we have a "reliable" variable, the weight of the "unreliable" variables is computed as the average of the pseudocosts of the "reliable" ones.
- We keep updating the pseudocosts at every remaining node, even when we already consider them as "reliable".

1.5.3. Violation transfer. As discussed in Section 5.2 in [6], it consists of an algorithm "to estimate the impact of a variable on the current status of the problem, when all other variables are fixed". The precise computations rely on the specific reformulations of factorable functions on which state-of-the-art solvers such as BARON [110] and Couenne [6, 7] build upon, and which are of a different nature of the RLT reformulations. In [6], it is shown that the score of a variable j can be

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

computed as:

$$\Theta_j^{\text{vt}} = w(j) \sum_{r \in M} |\lambda_r a_{ri}|,$$

where M is the set of constraints of the problem, and the variable with the largest score is chosen for branching. The coefficient $w(j)$ is related to the infeasibility behind variable j (and has no direct counterpart in the RLT relaxations), λ_r is the dual value of the constraint r and the a_{ri} values come from the constraints' coefficients. In order to provide a branching rule in the spirit of violation transfer, we use again the violations of the RLT defining identities as our measure of infeasibility (instead of the $w(j)$ coefficients). These violations are weighted using the dual values as follows. Let M_j be the set of constraints involving variable j in the relaxation. Then, given $r \in M_j$, let a_{hJ} denote the coefficient of each monomial J in constraint r. Now, we compute the weighted coefficient of each variable j in each constraint $r \in M_j$ in two different ways, a^1_{rj} and a^2_{rj}, as follows:

$$a^1_{rj} = \sum_{J \in r : j \in J} \frac{|\{j' \in J : j' = j\}|}{|J|} \cdot a_{rJ} \quad \text{and} \quad a^2_{rj} = \sum_{J \in r : j \in J} \frac{|\{j' \in J : j' = j\}|}{|J|} \cdot |a_{rJ}|,$$

so each weight is proportional to the multiplicity of j in the corresponding monomial. Then, we calculate the score of variable j using again Eq. (4), where we define the weights in two different ways (again, they are not "monomial-specific", i.e., $w^{\text{vt}}(j, J) = w^{\text{vt}}(j)$ for all J):

$$w^{\text{vt}_1}(j) = \sum_{r \in M_j} |\lambda_r a^1_{rj}| \quad \text{or} \quad w^{\text{vt}_2}(j) = \sum_{r \in M_j} |\lambda_r a^2_{rj}|.$$

1.5.4. Branching rules. We can now build upon the concepts described in the preceding sections to present, in Table 1.1, the different weights that are used later in this thesis. They have all appeared in different works: the first four in [59], the graph-based ones in [54] and the reliability ones in [57]. Since all of the branching rules in Table 1.1 are particular cases of Eq. (4), the convergence of the resulting RLT algorithm is guaranteed (see, for instance, Theorem 1.7 in [58]).

1.6. Conic programming and associated enhancements

We now want to introduce the basic concepts behind *conic programming*, as we use them in Section 1.6.3, and also in Chapter 2 and Chapter 8. This section is based on [81].

1.6.1. Basic conic programming concepts. The basic concept behind this class of optimization problems is the concept of *cone*. A cone is just a subset of points in a vector space with the property that given a point x in the cone, λx is a point of the cone for all $\lambda > 0$. An interesting property cones have, is that, without being an arbitrary nonlinear region, they are sufficiently general so that a number of commonly used regions can be written as intersection of cones and hyperplanes (for example, a circle, an ellipse, a parabola, etc.). This allows to solve problems more general than those of linear programming, but still maintaining a structure that allows for efficient algorithms.

A cone C is called *proper* if it is convex, closed, no hyperplane contains it and if $x \in C$, then $-x \notin C$. Next, we present two classes of proper cones that are particularly important in the filed of optimization: the *positive semidefinite*

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 1.1. List of branching rules used.

Branching rule	**Definition**
Maximum (`max`)	Eq. (3).
Sum (`sum`)	Eq. (4), with $w(j, J) = 1$ for all j and J.
Dual (`dual`)	Eq. (4), with $w(j, J)$ defined as the sum of the absolute values of the shadow prices of the problem constraints containing $J \cup \{j\}$.
Range (`range`)	Eq. (4), with $w(j, J) = \frac{\min\{u_j^k - \bar{x}_j^k, \bar{x}_j^k - l_j^k\}}{u_j^1 - l_j^1}$.
Eigencentrality VIG (`eig-VI`)	Eq. (4), with $w(j, J)$ defined as the eigencentrality of x_j's node in VIG.
Eigencentrality CMIG (`eig-CMI`)	Eq. (4), 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).
Sum Rel. (`sum rel.`)	Section 1.5.2 using `sum` as the base criterion.
Dual Rel. (`dual rel.`)	Section 1.5.2 using `dual` as the base criterion.
Range Rel. (`range rel.`)	Section 1.5.2 using `range` as the base criterion.
Eigencentrality CMIG Rel. (`eig-CMI rel.`)	Section 1.5.2 using `eig-CMI` as the base criterion.
Violation transfer 1 ($\texttt{vt}_1$)	Section 1.5.3 using w^{vt_1} in Eq. (4).
Violation transfer 2 ($\texttt{vt}_2$)	Section 1.5.3 using w^{vt_2} in Eq. (4).

cone and the *second-order cone*. The first one is defined in the space of symmetric matrices of order n, denoted by $\mathcal{S}^n$, where the matrices belonging to this cone are called *positive semidefinite matrices*. The formal definition is the following:

$$\mathcal{S}_+^n := \left\{ W \in \mathcal{S}^n \text{ s.t. } \boldsymbol{v}^T W \boldsymbol{v} \geq 0, \text{ for all } \boldsymbol{v} \in \mathbb{R}^n \right\}. \tag{6}$$

The cone $\mathcal{S}_+^n$ is well known in linear algebra, and there are many alternative definitions equivalent to the one above. For example, $W \in \mathcal{S}_+^n$ if and only if all eigenvalues of W are nonnegative, or if and only if all principal submatrices of W have nonnegative determinants. The second one, the second-order cone, is defined in $\mathbb{R}^n$, and its formal definition is

$$\mathcal{Q}^n := \left\{ \boldsymbol{x} \in \mathbb{R}^n \text{ s.t. } x_1 \geq \sqrt{x_2^2 + x_3^2 + \cdots + x_n^2} \right\}. \tag{7}$$

These two cones, together with the *nonnegative cone*, which is just $\mathbb{R}_+$, represent the fundamental cones usually employed in conic optimization.

With conic optimization we refer to problems of this form:

$$\inf\{\boldsymbol{c}^T \boldsymbol{x} \,:\, A\boldsymbol{x} = \boldsymbol{b},\, \boldsymbol{x} \in C\} \tag{8}$$

where $c \in \mathbb{R}^n, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$ and $C \subset \mathbb{R}^n$ is a proper cone. This is a general definition, and in practice there are only efficient algorithms for some of these problems. A class of problems where efficient algorithms can be designed was proved by [97] to be those defined with *symmetric* cones[1]. In particular, some examples are: linear programming problems where the cone is a Cartesian product of nonnegative cones and therefore symmetric; *second-order cone programming (SOCP)* problems,

[1]The definition of symmetric cone can be seen in [38].

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

with the cone being a product of a finite number of second-order cones and therefore symmetric; and *semidefinite-programming (SDP)* problems, where the cone is defined as the product of a finite number of positive semidefinite cones and therefore also symmetric. One thing to note here is that SDP is a generalization of SOCP, and SOCP is a generalization of linear programming.

Regarding the solvers for conic problems, there has recently been an improvement in the number of solvers available as well as an improvement in their performance. There are solvers for linear programming that have been expanding their functionalities and adding support for second order cone constraints, such as Gurobi [64], Xpress [40] and CPLEX [70]. On the other hand, there is Mosek [93], a reliable solver for general semidefinite optimization problems.

One of the applications of conic optimization, which is particularly relevant for Chapter 8 in this thesis, is that, despite being convex problems, they can provide tight relaxations of nonconvex optimization problems. Also, it will be convenient for us to use SOCP constraints to equivalently impose quadratic constraints and a quadratic objective function, as described in Section 1.6.2. In Section 1.6.3 we also describe some of the enhancements that can be incorporated into the RLT algorithm using conic programming.

1.6.2. Some reformulations using conic constraints. Conic solvers can solve optimization problems with some nonlinearities, as long as these can be written in the form of a cone or an intersection of a cone with a hyperplane. Here we present two examples: writing a quadratic constraint as an SOCP one, and reformulating a problem to remove a quadratic function from the objective function using an SOCP constraint. This is something that some solvers do automatically, if they detect these expressions that can be rewritten as conic constraints, but some others like Mosek, require that the user provides the SOCP formulation. We present here explicitly these two reformulations, since they are relevant for Chapter 8.

The first example is simple, since it just involves the definition of second order cone. If we have a constraint of the form

$$\|\boldsymbol{x}\|_2 \leq t,$$

it can be rewritten as $(t, x) \in \mathcal{Q}^{n+1}$, by using Eq. (7). If we have a constraint of the form

$$\|\boldsymbol{x}\|_2^2 \leq t,$$

we can write it as $(1/2, t, \boldsymbol{x}) \in \mathcal{Q}_r^{n+2}$, where $\mathcal{Q}_r^{n+2}$ is a *rotated second order cone*, which is just a second order cone transformed by an ortogonal rotation matrix. Its definition is

$$\mathcal{Q}_r^n = \left\{\boldsymbol{x} \in \mathbb{R}^n \text{ s.t. } 2x_1x_2 \geq x_3^2 + \cdots + x_n^2,\ x_1, x_2 \geq 0\right\}. \tag{9}$$

The second example is a bit more complex. Suppose that we have an optimization problem of the form

$$\begin{aligned} \min \quad & \frac{1}{2}\boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{c}^T \boldsymbol{x}, \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0, \end{aligned}$$

where we have a general convex quadratic objective function (i.e. Q is semidefinite positive). As discussed in [94], we can introduce a new variable y and obtain the

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

following equivalent definition

$$\begin{aligned} \min \quad & y + \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} = \boldsymbol{b}, \\ & \boldsymbol{x} \geq 0, \\ & (1, y, F\boldsymbol{x}) \in \mathcal{Q}_r^{k+2}, \end{aligned}$$

where $F \in \mathbb{R}^{k \times n}$ is a matrix such that $Q = F^T F$ and $\mathcal{Q}_r^{k+2}$ is again a rotated second order cone. There are other nonlinearities that can also be written as SOCP and SDP constraints, we refer the reader to [81] and [94] for more examples.

1.6.3. Enhancements based on conic programming concepts. Building upon the above conic programming concepts, in this section we want to briefly discuss something that is not directly part of the RLT algorithm itself, but can contribute to its performance: how tighter relaxations of $PP(\Omega)$ can be obtained by augmenting $LP(\Omega)$ with conic-based constraints. This approach is not directly part of the work presented in this dissertation, but it appears in Chapter 2 and later in Chapter 8, although in this last case in a different context to the RLT technique.

There has been a lot of research on adding conic-based cuts to the relaxations of integer and nonlinear optimization problems. The core idea is that some appropriately-built matrix has to be semidefinite positive at any feasible solution of $PP(\Omega)$. Then, one can directly incorporate these conic constraints or use them to define other constraints such as linear or SOPC cuts. For example, [118] presents an approach to strengthen the RLT relaxations with SDP-based linear cuts. [4, 59] are two more recent contributions that build above the same approach.

What hasn't been done until recently in practice is adding these semidefinite constraints directly. A possible reason is that solvers for conic optimization problems were not very reliable nor efficient. This has changed, and they have reached a level of performance that allows their use in the context of a branch-and-bound algorithm. One of the works that exploits this is [61], adding semidefinite cuts to the RLT relaxation and solving them with Mosek [93].

1.7. Convergence results

We present here the fundamental results required to prove the convergence of the RLT technique. The statements and the proofs can be found in [58], but in this section we present some modified statements for some of them updating the notation to match the one used in this dissertation. We also include the proofs of the lemmas in Appendix B, since we employ them in the proofs in Chapter 4.

We denote by B the variables in Ω that are bounded, i.e., have a finite upper bound. In this chapter, $B = N$, since all variables in Ω have a finite upper bound. The distinction will become relevant in Chapter 4, where some variables may not be bounded. This way, we will be able to use these results in that chapter. Recall that given a degree δ, N^δ denotes the collection of multisets $\{(N, \mu) \text{ s.t. } |(N, \mu)| = \delta\}$, and N^δ_-, the collection $\{(N, \mu) \text{ s.t. } 1 \leq |(N, \mu)| < \delta\}$. We let $\nu[PP(\Omega)]$ denote the objective function at a global optimum of $PP(\Omega)$. If $PP(\Omega)$ is infeasible, we let $\nu[PP(\Omega)] = \infty$, and if $PP(\Omega)$ doesn't have a finite optimum, we let $\nu[PP(\Omega)] = -\infty$.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

The first lemma, Lemma 1.1 and its corollaries, are technical results needed in the proof of Theorem 1.5. In particular, Lemma 1.1 is the result we mentioned in Section 1.3 as one of the motivations in [31] for the introduction of the J-sets. On the other hand, Lemma 1.4 proves one core property of the queue of problems Q in `plainRLT`, that is later employed in Chapter 4. This property is the following: whenever a problem $LP(\Omega^k)$ enters the queue, it is assigned a potential LB^k which comes from the optimal solution of its parent node and represents the best objective function that might be achieved when solving $LP(\Omega^k)$. This causes that if a problem from Q is ever solved by the algorithm, then $LB^k \leq \nu[PP(\Omega)]$, the optimal value of $PP(\Omega)$.

LEMMA 1.1. *Let $B \subseteq N$ be a subset of bounded variables in Ω. Let δ' be such that $1 \leq \delta' < \delta$ and let J_1, J_2 be two multisets such that $J_1 + J_2 \in N^{\delta'}$, $\mathrm{supp}(J_2) \subseteq B$. Then, the bound-factor constraint $[F_{\delta'}(J_1, J_2)]_L \geq 0$ is implied by the bound-factor constraints $[F_\delta(J_3, J_4)]_L \geq 0$ with $J_3 + J_4 \in N^\delta$, $\mathrm{supp}(J_4) \subseteq B$.*

COROLLARY 1.2. *In the feasible region of $LP(\Omega)$, all X_J variables with $J \in B^\delta_-$ are bounded.*

COROLLARY 1.3. *Let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ be a feasible solution of $LP(\Omega)$. If $\bar{x}_p = l_p$ for $p \in B$, then, for all $J \in B^{\delta-1}_-$, $\bar{X}_{J\cup\{p\}} = l_p \bar{X}_J$. Similarly, if $\bar{x}_p = u_p$ for $p \in B$, then, for all $J \in B^{\delta-1}_-$, $\bar{X}_{J\cup\{p\}} = u_p \bar{X}_J$.*

LEMMA 1.4. *Suppose `plainRLT` is applied to solve problem $PP(\Omega)$. Let k correspond to a relaxation $LP(\Omega^k)$ solved during the course of the algorithm. Then, $LB^k \leq \nu[PP(\Omega)]$.*

THEOREM 1.5. *Suppose that `plainRLT` is applied to solve a feasible problem $PP(\Omega)$. Then, one of the following statements holds:*

(1) The algorithm finishes in a finite number of iterations, being the obtained solution a global optimum of $PP(\Omega)$.

(2) An infinite sequence of iterations is generated such that, along any infinite branch of the tree, the accumulation points of the associated (bounded) sequence of variables, $\{\bar{\boldsymbol{x}}\}$, are global optima of $PP(\Omega)$.

1.8. Implementation in RAPOSa

To finish the introduction to the RLT technique, we present here the implementation of this technique that has motivated this PhD dissertation, RAPOSa, and has allowed the different computational experiments presented along the document. This solver has been in development at the University of Santiago de Compostela and CITMAga for some years, being the effort led by my PhD supervisor Julio González Díaz. It has suffered profound changes since the endeavour started, but it's now a tool capable of solving polynomial optimization problems to certified global optimality, and it's used thousands of times each year in the online platform NEOS [29]. Even thought it's not open-source, it can be freely downloaded at `raposa.usc.es`.

As described in [59], it's written entirely in `C++` and it is competitive with the state-of-the-art free and commercial solvers for global optimization. In this section we discuss some implementation choices behind RAPOSa's current version that impact the results we present in the subsequent sections. Thus, this section does

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

not intend to be an exhaustive review of all of the features included in RAPOSa, but just of those relevant to the dissertation. The base algorithm implemented in RAPOSa is the algorithm `plainRLT`, and mostly follows the original algorithm proposed by [120].

In the linearization of $PP(\Omega)$, RAPOSa uses the J-sets approach described in Section 1.3 to reduce its size. The numerical results in [59] show that the use of J-sets is the best performing approach for RAPOSa. Another improvement regarding the linearization that is implemented in RAPOSa is the use of SDP cuts to improve the tightness of the relaxation. This goes beyond the aim of this thesis, and the reader can check [61] for a more detailed review of the use of SDP cuts in RAPOSa. They are disabled by default, as the increase in computational time is not always worth it but just for some specific types of problems that benefit from the strengthened relaxation.

Once the problem $PP(\Omega)$ is linearized, and we have problem $LP(\Omega)$, a core part of the algorithm is solving this problem and the subsequent ones generated in the branching. Something that can seem at first as a simple thing can have a big impact on the performance of the overall algorithm. There are several choices involved here. For example, which auxiliary solver to use, how to interface with it, which information is sent, whether to apply some preprocessing to the problem beforehand, etc. RAPOSa has native integration with Gurobi [64] via their `C++` API and can also be used with any of the solvers available in the Google OR tools [104]. Before sending the problem to the solver, we apply some bound tightening. The techniques implemented are described in Section 1.4. In particular, we have the option to run OBBT and FBBT, and by default we run OBBT only in the root node for a limited amount of time and then FBBT in all of the nodes in the tree. The FBBT technique we implemented builds upon [8], where interval arithmetic and a two phase process are used to infer bounds on the variables. The user can configure not only the nodes at which bound tightening is run but also which type of bound tightening is used each time it's called.

Other important aspect of solving the linear problems is which variable bounds are sent to the solver. In RAPOSa, we send the bounds of all variables (including RLT variables) to the solver, but as discussed in Chapter 3, this seemingly naive decision can have a significant impact on the performance of the algorithm, depending of the solver used and how it handles those bounds. Also regarding the solving of the linear problems, other improvement implemented in RAPOSa is the warm-start. This is an enhancement based on the fact that the only difference from a node to its "children" in the branch-and-bound tree is a bound of one of the variables. This suggests that using the solution or the basis of the parent's optimal point as a starting point for solving the child could be beneficial. There are some instances where the warm-start does not help, as can be seen in [59]. Nonetheless, it's enabled by default in RAPOSa.

Other important point in the algorithm is the effort made to find an upper bound as soon as possible. Theoretically, one would expect that at some point in the algorithm, one of the nodes in the tree would lead to a feasible solution in the original problem and therefore an upper bound of the algorithm. In practice, there's a need to use a local solver in the tree and devote some time for it to search for a local optima of $PP(\Omega)$. RAPOSa does this by default, and any solver that can read .nl files [53] and solve nonlinear optimization problems can be used here.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Each time the local solver is called, the solution associated to the current lower bound is passed to the solver as an initial solution. The local solver is not called at every node, as this would be very consuming, and it's up to the user to find a equilibrium between the frequency of calls to the local solver and the reduction in the optimality gap obtained. The user can modify parameter LS, and the local solver is then used at iteration k if $k \geq LS^l$, where l is a counter for the calls to the local solver.

Lastly, regarding the branching, all of the branching rules described in Section 1.5 are implemented in RAPOSa. The default one is `dual` branching rule, but the user can select any of the others. The selection of the branching rule has a big impact in the performance of RAPOSa, something that was discussed in [59]. The default branching rule is the one that performed best in the numerical results from [59].

In the computational results we present in this thesis, the "baseline" option we usually include does not have all of the enhancements implemented in RAPOSa enabled. This is because, for each improvement added to the algorithm, we want to calculate the "pure" improvement, and starting from an already "enhanced" algorithm distorts the results (and can cause numerical issues in some cases as a consequence of the interaction of the improvements). They are enabled by default when a user executes RAPOSa (as we want to give the user the best possible performance), but we will not consider them as part of the baseline to which we will compare the performance of the proposed changes to the algorithm. For example, warm-start is always disabled, and bound tightening is only included on some comparisons where its use is deemed interesting. The local solver is always used, as it does not interfere with the improvements discussed in the thesis.

1.9. Computational environment and performance metrics

Different chapters in this dissertation contain numerical results. All of them were carried out on similar conditions, so in this section we summarise the common information for chapters in Part 1 and Part 2 to avoid repetition. All of the executions were run on Galicia Supercomputing Center (CESGA), but there is a substancial difference between the chapters: in march 2022, CESGA moved from supercomputer Finisterrae II (FT2) to Finisterrae III (FT3), so the chapters with executions done prior to this date were run on FT2 and the ones done after on FT3. The chapter were FT2 was used is Chapter 5, the other ones used supercomputer FT3.

The nodes in FT2 are powered with 2 ten-core Intel Haswell 2680v3 CPUs with 128GB of RAM and 1TB of hard drive, and the ones in FT3 are powered with two thirty-two-core Intel Xeon Ice Lake 8352Y CPUs with 256GB of RAM and 1TB SSD. All of the executions were carried out using an automated system we developed to run the executions, store the results and compute tables and graphics of the performance metrics. This environment is described in Appendix A.

All of the executions in Part 1 and Part 2, except the ones from Section 5.6, were run using solver RAPOSa, introduced in Section 1.8. The time limit varies by chapter, but all of them have as stopping criterion that the relative or absolute gap is below the threshold 0.001. We use three sets of instances to test the performance of the different approaches under consideration. The first one, DS, is taken from [30] and consists of 180 instances of randomly generated polynomial optimization

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

problems of different degrees, number of variables, and density. The second test set, MINLPLIB, comes from the well-known benchmark MINLPLib [20], a library of mixed-integer nonlinear programming instances. We have selected from MINLPLib those instances that are $PP(\Omega)$ problems with box-constrained and continuous variables, resulting in a total of 166 instances. The third test set, QPLIB, comes from another well-known benchmark, QPLIB [49], 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. In some chapters, some problems have been removed because of numerical issues from the computational studies. In Chapter 5, for the learning results in Section 5.3.1, we disregarded instances solved at the root node, since the branching rule plays no role in them. We also discarded unconstrained problems, as some of the features used for the learning have the number of constraints in the denominator.[2]

To assess the performance of the different approaches studied in this thesis, we use several performance metrics in the results tables that we summarise here:

- **Solved**. Number of instances solved to certified optimality (relative or absolute gap below 0.001).
- **Gap**. Geometric mean of the optimality gap obtained by each approach. We remove instances for which i) at least one approach did not return an optimality gap and ii) all the approaches solved it within the time limit.
- **Time**. Geometric mean of the running time of each approach. We remove instances for which i) every approach solved it in less than 5 seconds and ii) no approach solved it within the time limit.
- **Pace**. Geometric mean of pace^{LB}, as introduced in [54]. It represents the number of seconds needed to improve the lower bound of the algorithm by one unit (the pace at which the lower bound improved). We remove instances solved by every approach in less than 5 seconds. As thoroughly discussed in Section 5.2.1, the main motivation behind this performance measure is that it allows to compare the performance on all instances together, whereas time and gap fail to do so. Time is not informative when comparing performance between instances not solved by any approach (all of them reach the time limit). These instances can be compared using gap which, in turn, is not informative in instances solved by all approaches (all of them close the gap), and where time could be used. Pace, on the other hand, is informative regardless of the number of configurations that might have solved each of the different instances.
- **Nodes**. Geometric mean of the number of nodes explored in the branch-and-bound tree. We only consider the instances solved by all the approaches within the time limit.
- **bounds$^{\mathbf{BT}}$**. Mean of the average improvement of the variable bounds after applying a certain bound tightening approach at the root node.
- **time$^{\mathbf{BT}}$**. Mean of the bound tightening time at the root node.

When the different measures are used, they have, in parenthesis, the number of instances on which each measure is computed. We have chosen to report geometric means in our results, with the exception of bounds$^{\mathrm{BT}}$ and time$^{\mathrm{BT}}$, for which the standard mean is used. Geometric means are known to have the advantage of

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[2]Instances from the three datasets can be downloaded at `https://raposa.usc.es/instances`.

being less sensitive to outliers, which often makes them preferable for performance comparisons. The reason to report the standard mean for bounds$^{\mathrm{BT}}$ is that it measures how much the bounds are tightened with the different approaches, but it is not really a performance measure as gap, time, or pace, for instance. Thus, since bounds$^{\mathrm{BT}}$ measures how much certain parameters of the problem are changed before the branch-and-bound starts, reporting the standard averages allows to get a better picture of the impact of the bound tightening phase on the search space of the algorithm. Similarly, time$^{\mathrm{BT}}$ is not a measure of the overall performance of a given approach, but just a measure of the time spent on bound tightening at the root node. Reporting standard averages provides a more informative quantification of the impact of a given approach during this phase of the algorithm.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 2

Improved domain reduction

Publication derived from this chapter: Ignacio Gómez-Casares et al. *Impact of domain reduction techniques in polynomial optimization: A computational study.* 2024. arXiv: 2403.02823 [math.OC]. URL: https://arxiv.org/abs/2403.02823

2.1. Introduction

The design and implementation of global optimization algorithms for general MINLP problems is a very active field of research, and the number of available solvers has been steadily increasing over the past years. Recently, companies behind state-of-the-art MILP solvers such as Xpress [40] and Gurobi [64] have announced the release of new versions capable of solving general MINLP problems to certified global optimality. Convexifications and domain reduction techniques are probably the two most important elements behind the efficiency of the spatial branching required to handle the nonconvexities as can be seen, for instance, in [108], [126], and [7]. The focus of this chapter is precisely on the joint assessment of the individual impact of different aspects of domain reduction on the performance of a global optimization algorithm. We hope our analysis can guide the efforts in the development and implementation of this type of enhancements in present and future global optimization solvers for nonconvex problems.

Two essential aspects of domain reduction, present in any spatial branching algorithm, are the selection of the branching variable and the selection of the branching point, since both of them are done at each and every node of the branch-and-bound tree as described in Section 1.5. The choice of the branching variable is relatively well understood, partially because of the thorough research for MILP problems (refer, for instance, to the seminal works [84] and [1]) but also due to some specialized papers for spatial branching such as [126] and [7]. In the specific context of polynomial optimization, the impact of the criterion for variable selection is thoroughly discussed in [59] and [54]. On the other hand, the impact of the selection of the branching point has received much less attention and, although different approaches are mentioned in [108], [126], and [7], we are not aware of any numerical analysis assessing how relevant these decisions may be in the performance of the resulting algorithm. In order to bridge this gap, one of the domain reduction aspects we cover in this chapter is precisely the selection of the branching point.

Despite the relevance of the selection of both the branching variable and the branching point, probably the single most important aspect in domain reduction in nonlinear programming, at least in terms of the research it has generated, is bound tightening, discussed in Section 1.4. In [105], the authors present a thorough computational analysis on the huge impact that domain reduction techniques, and bound tightening in particular, have in three state-of-the-art global optimizers for

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

NLP and MINLP problems. [59] reports a similar impact in the context of an RLT-based algorithm for continuous polynomial optimization problems.

Although integrating a basic bound tightening scheme in a global optimization solver is not too demanding, there is a wide range of sophisticated approaches that have been discussed in the literature and whose implementation may be significantly more complex. [5] proposes to enhance FBBT by using convex combinations of constraints instead of only the original ones, while [108] and [55] present FBBT enhancements that use Lagrangian dual information from subproblems previously solved by the algorithm. The latter two approaches can be seen as particular cases of the PU-PR domain reduction framework presented in [126]. These enhanced methods often entail an additional computational overhead node by node, which should then be overcome by the reduction in the size of the resulting branch-and-bound tree. A second contribution of this chapter is devoted to analyze this trade-off, by studying the impact of these sophisticated approaches when integrated into a solver that already has a basic bound tightening scheme in place. More precisely, we study, in the context of the RLT-technique for polynomial optimization [120], the impact of some of the Lagrangian-based enhancements in [108] and [55].

The standard implementations of optimality-based bound tightening require to solve two subproblems for each problem variable, one to adjust its lower bound and one to adjust its upper bound. Moreover, these problems are typically of a difficulty comparable to that of the nodes solved during the branching process itself. A novel contribution of this chapter is to study the use of (nonlinear) conic relaxations, in particular second-order cone programming (SOCP) and semidefinite programming (SDP) ones, to improve the tightening obtained by OBBT. This approach is motivated by the recent analysis in [61] where, in the context of polynomial optimization problems, the authors show the potential of using (nonlinear) conic constraints to tighten the (linear) relaxations solved by an RLT-based algorithm. We adapt their ideas with the goal of improving the performance of the underlying bound tightening techniques.

It is worth noting that domain reduction techniques are not only relevant for theoretical purposes or for the design of general purpose solvers. There have also been important efforts to adapt these techniques to particular classes of problems, directly connected with practical applications. Particularly important are those related to optimization of power networks in general [28] and, more specifically, to the alternating current optimal power flow problem or ACOPF, a very active field of research. We will describe the work we are doing in this field in Part 3 of this dissertation. [26] shows the effectiveness of domain reduction techniques for ACOPF problems, [123] presents an optimality-based bound tightening to improve the bounds in different parts of the network, using a (convex) quadratic relaxation of the problem, and [114] proposes three new methods for reducing the domain, focusing on being efficient on large-scale grids.

Summarizing, the study of domain reduction techniques has been a core topic in global optimization in recent years. In this chapter, we focus on the impact of some of these techniques in the specific context of polynomial optimization problems, building upon an RLT-based algorithm for the analysis [120]. The contributions are the following:

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(1) To study how sensitive the performance of a branch-and-bound algorithm may be with respect to different strategies for choosing the branching point.
(2) To study the impact on performance of some advanced enhancements of FBBT and OBBT on an algorithm already equipped with basic FBBT and OBBT functionalities. In particular, a novel contribution is the study of an OBBT implementation based on the solution of SOCP and SDP relaxations. For a introduction to conic optimization, check Section 1.6.
(3) We embed the above domain reduction enhancements in a learning framework and study its potential to deliver additional improvements on performance, in Section 5.4.

All of the results presented in this chapter are computed using the computational environment described in Section 1.9, using test sets MINLPLIB, DS and QPLIB. This chapter is based on [57], a joint work with Brais González Rodríguez, Julio González Díaz and Pablo Rodríguez Fernández.

2.2. OBBT enhancements based on conic optimization

In recent literature, such as [19] and [18], there has been an increase in the research devoted to the use of different SOCP and SDP constraints to define tighter relaxations of a given nonlinear problem, specially in the study of polynomial optimization problems. A related approach consists in adding linearizations of the SDP constraints, called linear SDP-based cuts, as in [118], [4], and [59]. In [61] the authors use second-order cone and semidefinite constraints in order to tighten the standard RLT linear relaxations along the branch-and-bound tree. Their results show that the use of (nonlinear) conic constraints has significant potential, particularly so for some specific subclasses of problems in the test sets they consider. In this section we build upon the approach in [61], using the same collection of SOCP and SDP constraints but only to tighten the linear relaxations that OBBT has to solve at the root node, so no specialized conic optimization solver is required beyond the root node. The goal is to study to what extent this enhanced OBBT can provide tighter bounds for the variables. Since solving these conic relaxations is usually more difficult than solving the original linear relaxations, there is a trade-off between time consumption and the quality of the bounds. In order to get the best out of this trade-off, when deciding which SOCP and SDP constraints to add to tighten the linear relaxations, one has to be careful to preserve whatever sparsity structure the original problem may have. To do so, in [61] the authors anchor the definition of the second-order cone and semidefinite constraints to the J-sets of $PP(\Omega)$ as follows:

- **SOCP**: For each J-set J and for each pair of variables in J, $x_i \neq x_j$, we add the following second-order cone constraint to the relaxation solved at the OBBT phase:

$$\frac{X_{ii} + X_{jj}}{2} \geq \left\| \begin{pmatrix} X_{ij} \\ \frac{X_{ii} - X_{jj}}{2} \end{pmatrix} \right\|_2 . \tag{10}$$

- **SDP^1**: For each J-set, we define the vector $\boldsymbol{\omega}^1 = (x_j)_{j \in J}$ and we build matrix $W = (\boldsymbol{\omega}^1)^T(\boldsymbol{\omega}^1)$. We denote the linearization of matrix W by $[W]_L$, obtained by replacing all monomials in it by the corresponding

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

RLT variables. Then, we add, to the linear relaxation solved at the OBBT phase, the constraint $[W]_L \in \mathcal{S}_+^{|J|}$ (positive semidefiniteness).

- **SDP²**: We do the same as in SDP¹ but changing the vector $\boldsymbol{\omega}^1$ to $\boldsymbol{\omega}^2 = (1, (x_j)_{j\in J})$.

It is worth noting that, in [61], the authors discuss some additional approaches which, in order to reduce the number of conic constraints added to the standard relaxations and further reduce the computational effort to solve them, check what constraints are binding at the root node and then disregard all the nonbinding ones in the subsequent relaxations. Although such an approach might also be applied to our OBBT relaxations, we believe it is not advisable, since the relaxations solved by the OBBT have completely different objective functions, so there might be no correspondence between the binding constraints of the different subproblems.

2.2.1. Numerical results. The default configuration of RAPOSa relies on Gurobi [64] for the solution of the linear relaxations. For the analysis in this section, we use Gurobi and Mosek [93] for the SOCP relaxations. The SDP ones are solved with Mosek, since Gurobi cannot handle them. With the goal of assessing the potential of an OBBT whose relaxations have been tightened with (nonlinear) conic constraints, we compare the performance of the different conic approaches with a baseline configuration already equipped with standard OBBT and FBBT functionalities:

- `Baseline`: We use the standard OBBT and FBBT used in [59].
- `SOCP`^{G}: We follow approach **SOCP** above and solve the relaxations with Gurobi.
- `SOCP`^{M}: We follow approach **SOCP** above and solve the relaxations with Mosek.
- `SDP`^{1}: We follow approach **SDP¹** above and solve the relaxations with Mosek.
- `SDP`^{2}: We follow approach **SDP²** above and solve the relaxations with Mosek.

TABLE 2.1. Performance of the different conic OBBT approaches.

(403 instances)	(403) Solved	(111) Gap	(153) Time	(268) Pace	(285) Nodes	(403) bounds^{BT}	(403) time^{BT}
`Baseline`	286	0.132	56.95	6.56	104.43	0.338	280.99
`SOCP`^{G}	285	0.133	97.67	9.98	104.38	0.341	391.92
`SOCP`^{M}	286	0.128	65.69	6.99	102.20	0.344	287.92
`SDP`^{1}	286	0.150	114.42	15.19	103.62	0.339	404.08
`SDP`^{2}	286	0.159	132.93	17.89	103.63	0.345	433.57

In Table 2.1 we present the results associated to the aforementioned approaches, where shaded numbers are used to highlight whenever an approach outperforms the baseline for a certain metric. At first sight, it seems that the overall performance of the new conic approaches for OBBT is worse than the one delivered by `Baseline`, specially for the SDP ones. As expected, with respect to `Baseline`, all the conic approaches reduce the number of nodes of the branch and bound tree and improve the tightness of the bounds (bounds^{BT}). Yet, the computational overhead required

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

by OBBT at the root node (time^{BT}) is not compensated by the modest improvement on $\text{bounds}^{\text{BT}}$. This is is also captured by the pace of the different approaches: although $\texttt{SOCP}^G$ and $\texttt{SOCP}^M$ are very competitive in terms of gap, the fact that they are more time consuming than `Baseline` also makes them worse in terms of pace. $\texttt{SOCP}^M$ is clearly the best performing conic approach, being fairly competitive with `Baseline`. Indeed, if we looked at standard means instead of geometric ones (not shown in the table), we would see that $\texttt{SOCP}^M$ and `Baseline` are essentially tied for both time (368.18 vs 369.81) and pace (216108.52 vs 215531.70). The main weakness of the conic approaches, and specially the SDP ones, is that they are more "risky", in the sense that, since conic solvers are not yet as efficient as linear ones, a conic approach may turn out to be particularly slow in some instances, which might overshadow promising behavior on others.

The results in Table 2.1 go along the same lines of those reported in [61], where these conic enhancements are not included at the OBBT stage of the algorithm but, instead, they are added to the RLT relaxation in each and every node of the branch-and-bound tree. The authors go on to show that, despite this initially discouraging aggregate behavior, there are specific instances and subclasses of problems in the tests under study in which some of the SOCP or SDP approaches significantly outperform their baseline configurations. In order to check whether or not we are in a similar situation, in Table 2.2 we present, for each approach under study, how often it is within 5% of the value of the best performing approach according to the given metric.

TABLE 2.2. Frequency with which each conic OBBT approach is within 5% of the best one.

(403 instances)	(111) 5% Gap	(153) 5% Time	(268) 5% Pace	(285) 5% Nodes	(403) 5% $\text{bounds}^{\text{BT}}$	(403) 5% time^{BT}
`Baseline`	100	120	227	208	243	361
$\texttt{SOCP}^G$	88	42	150	209	259	68
$\texttt{SOCP}^M$	103	93	200	224	283	141
$\texttt{SDP}^1$	82	23	116	214	249	54
$\texttt{SDP}^2$	80	10	102	230	358	37

The results are consistent with those in Table 2.1, with `Baseline` and $\texttt{SOCP}^M$ coming out on top. Importantly, Table 2.2 also shows that `Baseline` is not within 5% of the best performing approach in a significant amount of instances: 10% for gap, 22% for time, and 15% for pace. Thus, we could get significant performance improvements if we were able to anticipate the best approach to run on a given instance. This idea is explored in Section 5.4, where we embed these approaches in a machine learning framework, and train a model with the goal of predicting what the best approach would be on an unseen instance.

2.3. FBBT enhancements based on duality

In this section we take the ideas developed in [108] and [55] and study their impact when added on top of the basic bound tightening, OBBT and FBBT, discussed in Section 1.4 for the RLT technique. Both works were developed in the context of global solvers for nonconvex optimization problems, where a convex relaxation is

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

used in a branch-and-bound scheme. They discuss different approaches to improve the performance of the FBBT stages, in the sense of obtaining even tighter bounds, which we adapt below to the context of polynomial optimization.

The core idea is the same in both [108] and [55]: make use of the Lagrangian multipliers associated to previously solved subproblems to obtain valid cuts that can be added to FBBT's constraint propagation scheme. This should lead to (weakly) tighter bounds whenever FBBT is invoked but, at the same time, entail some computational overhead. Both approaches use these new cuts to extend the set of constraints on which the FBBT is run and they do so at all nodes of the branch-and-bound tree. In [55] they use information captured during the initial (or previous) pass of OBBT, while in [108] they use information from the solution of the relaxation at the current node in the tree. For the sake of completeness, we briefly describe below the two approaches and how they have been adapted to our context.

We start with the approach proposed in [55]. After solving all OBBT's problems we have, for each variable, one solution for its maximization problem and another one for its minimization problem. Suppose we have the primal-dual solution $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{\lambda}}, \bar{\mu})$ from solving the minimization problem for variable x_j, where $\bar{\boldsymbol{\lambda}}$ is the vector of multipliers associated with the constraints of the OBBT relaxation and multiplier $\bar{\mu}$ corresponds with an additional constraint bounding the objective function to the best upper bound available when OBBT was applied. Given a valid upper bound UB for $PP(\Omega)$, the following constraint is a valid cut for $PP(\Omega)$:

$$x_j \geq \bar{\boldsymbol{r}}^T \boldsymbol{x} + \bar{\mu} UB + \bar{\boldsymbol{\lambda}}^T \boldsymbol{b}, \tag{11}$$

where $\bar{\boldsymbol{r}}$ is the vector of reduced costs, $\boldsymbol{b}$ is the vector of right-hand sides of the linear relaxation used by OBBT. Similarly, we can use the maximization problem to obtain an upper bound for variable x_j.

Now, we describe the cuts presented in [108]. Consider the relaxation of $PP(\Omega)$ at a given node and suppose that constraint $g_r(\boldsymbol{x}) \leq 0$ is active at the solution of this node. Suppose, moreover, that this constraint has multiplier $\bar{\lambda}_r$, that UB is a valid upper bound for $PP(\Omega)$ and that LB is a lower bound for $PP(\Omega)$. Then, the following constraint is a valid cut for $PP(\Omega)$:

$$g_r(\boldsymbol{x}) \geq -(UB - LB)/\bar{\lambda}_r.$$

If we had an equality constraint, we would use the sign of $\bar{\lambda}_r$ to determine the sense of the new cut's inequality. This is also applied to the constraints of the form $x_j - u_j \leq 0$ and $l_j - x_j \leq 0$, obtaining:

$$x_j \geq u_j - (UB - LB)/\bar{\lambda}_r \quad \text{and} \quad x_j \leq l_j + (UB - LB)/\bar{\lambda}_r. \tag{12}$$

It is worth emphasizing that the values for the bounds UB and LB in constraints (11) and (12) are not static: they are updated as the algorithm progresses. Put differently, whenever any of these constraints are added to the FBBT at a given node, the current values for the bounds are used.

2.3.1. Numerical results. In order to assess the performance of the aforementioned approaches, we have studied different variations, depending on how aggressively we use them. We present here the best performing ones:

- `Baseline`: We use the standard OBBT and FBBT used in [59].
- $\texttt{FBBT}^{OB}$: The OBBT-derived cuts from [55] are added to FBBT at the root node and at nodes whose depth is a multiple of 50.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- FBBT^{NB}: The node by node cuts from [108] are added to FBBT at the root node and at nodes whose depth is a multiple of 10.
- FBBT^{OB+NB}: Both approaches are applied together.

In Section 2.3.2 we report some additional results which, in particular, show that applying these enhancements at all nodes is not desirable. This is because the computational overhead required to perform FBBT's constraint propagation on the extended set of constraints is not compensated by the gains from bound reduction. FBBT^{OB} and FBBT^{NB} provide a good balance between the computational effort and the reduction of the size of the branch-and-bound tree.

TABLE 2.3. Performance of the FBBT enhanced with cuts from [55] and [108].

(403 instances)	(403) Solved	(111) Gap	(134) Time	(248) Pace	(285) Nodes	(403) time^{BT}
Baseline	286	0.132	100.98	8.29	104.43	280.99
FBBT^{OB}	287	0.126	102.80	8.27	101.22	300.36
FBBT^{NB}	287	0.128	100.48	8.21	98.61	299.39
FBBT^{OB+NB}	287	0.128	98.32	8.08	94.52	317.24

The results in Table 2.3 show that the duality-based approaches consistently outperform Baseline for the different performance measures. Yet, the improvements are relatively small and, as expected, far from the ones reported in [59] when basic versions of OBBT and FBBT were added to an RLT implementation without any such functionality. In their numerical analysis, they obtain 70%-90% improvements for gap, time, and nodes in MINLPLib instances. Thus, the results in Table 2.3, combined with those in [59], confirm bound tightening techniques as a fundamental element of global optimization solvers but, at the same time, they show that the effort required to implement sophisticated variants of these techniques might not always be worthwhile. In particular, performance improvements such as the ones in Table 2.3 could be overshadowed by potential stability and robustness issues coming from these additional functions and unforeseen interference with other current or future functionalities.

2.3.2. Additional results for FBBT enhancements. Table 2.4 presents a more comprehensive summary of the numerical results obtained when studying different variations of the FBBT approaches described in this section. We study configurations in which the two approaches are applied at all nodes or just at the root node and at nodes of pre-specified depths. Notation wise, FBBT$^{OB,50+NB,10}$ indicates that constraints associated with approach FBBT^{OB} are added on nodes with a depth value multiple of 50, and constraints associated with FBBT^{NB}, at nodes with a depth value multiple of 10. Note that, under the approach FBBT$^{OB,50+NB,10}$, the cuts from FBBT^{OB} are never applied by themselves, since multiples of 50 are also multiples of 10. In order to assess whether or not this might impact performance, we also studied FBBT$^{OB,50+NB,9}$, under which both families of cuts are jointly applied at the same node much less often.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 2.4. Performance of different versions of the FBBT enhanced with cuts from [55] and [108].

(403 instances)	(403) Solved	(115) Gap	(141) Time	(255) Pace	(281) Nodes	(403) time^{BT}
`Baseline`	286	0.112	82.84	7.17	94.00	280.99
$\texttt{FBBT}^{OB,all}$	281	0.155	112.83	9.84	89.57	819.17
$\texttt{FBBT}^{OB,25}$	286	0.110	84.87	7.28	91.89	414.81
$\texttt{FBBT}^{OB,50}$	287	0.106	85.26	7.19	91.93	300.36
$\texttt{FBBT}^{NB,all}$	287	0.109	82.92	7.13	87.29	329.93
$\texttt{FBBT}^{NB,5}$	287	0.108	83.25	7.14	88.30	303.34
$\texttt{FBBT}^{NB,10}$	287	0.108	82.21	7.09	88.73	299.39
$\texttt{FBBT}^{OB,25+NB,5}$	284	0.117	82.28	7.15	86.04	429.98
$\texttt{FBBT}^{OB,25+NB,9}$	284	0.116	92.54	7.63	86.20	429.16
$\texttt{FBBT}^{OB,25+NB,10}$	284	0.115	82.68	7.17	86.41	428.97
$\texttt{FBBT}^{OB,50+NB,5}$	286	0.110	82.35	7.07	86.08	321.10
$\texttt{FBBT}^{OB,50+NB,9}$	286	0.111	81.70	7.03	86.26	317.24
$\texttt{FBBT}^{OB,50+NB,10}$	287	0.108	81.55	7.03	86.40	317.24

2.4. Impact of the branching point on the branch-and-bound tree

Whenever a subproblem is solved at a node of the branch-and-bound tree, important decisions have to be made. Two crucial ones are the selection of the branching variable and of the precise value of that variable on which branching is done. As mentioned in the introduction, the choice of the branching variable has been thoroughly discussed by past literature, mainly for MILP problems (see, for instance, [84] and [1]), but also in the context of nonlinear optimization and spatial branching in [126] and [7]. When it comes to polynomial optimization and the RLT technique in particular, [59] and [54] offer deep analyses on the impact of variable selection.

The selection of the branching point, on the other hand, has not received much attention by past literature and, importantly, we have seen no numerical analysis studying how the performance of a given solver might be affected by different strategies, which is precisely the main goal of this section. [108] and [7] briefly discuss the main strategies for selecting the branching point in the state-of-the-art solvers Couenne [7] and BARON [110], respectively. Suppose that a branching variable has already been chosen at a certain node. Then, given the current range of that variable, different natural branching points can be considered: the middle point of the interval, the value of the variable in the optimal solution of the subproblem, or some convex combination of these two points. [108] and [7] also suggest to branch on the value of the variable in the best solution already found by the algorithm (provided it belongs to the current range of the variable).

2.4.1. Numerical results. We now present and compare a few simple strategies for the selection of the branching point, which build upon the ideas outlined in [108] and [7]. Given the branching variable, we denote by OV its optimal value at the current node and by MP the middle point of its current range. Since, in some preliminary analysis, we observed that branching whenever possible on the value of

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the variable in the best available solution yields slight performance improvements at no cost, this approach is incorporated in the following five approaches:

- `Baseline`: Branch on OV.
- `0.75·OV+0.25·MP`: Branch on the convex combination $0.75 \cdot OV + 0.25 \cdot MP$.
- `0.5·OV+0.5·MP`: Branch on the convex combination $0.5 \cdot OV + 0.5 \cdot MP$.
- `0.25·OV+0.75·MP`: Branch on the convex combination $0.25 \cdot OV + 0.75 \cdot MP$.
- `MP`: Branch on MP.

It is worth noting that we also studied some other slightly more complex strategies along the lines suggested in [108] and [7], but they did not result in extra performance improvements. Thus, we chose to stick to these simple strategies for the sake of exposition.

TABLE 2.5. Performance of different approaches for branching point selection.

(403 instances)	(403) Solved	(110) Gap	(138) Time	(250) Pace	(280) Nodes
`Baseline`	286	0.124	99.21	8.16	105.70
`0.75·OV+0.25·MP`	287	0.111	93.22	7.68	99.58
`0.5·OV+0.5·MP`	288	0.108	96.75	7.64	102.10
`0.25·OV+0.75·MP`	287	0.108	97.79	7.70	105.23
`MP`	287	0.109	107.71	8.09	107.98

In Table 2.5 we see that all the new approaches beat `Baseline`, which is the worst one according to solved, gap, and pace. Further, `Baseline` is second to last according to time and Nodes, only ahead of `MP`. `Baseline` and `MP` are significantly outperformed by the strict convex combinations, which shows that both `OV` and `MP` should be taken into account to determine the branching point. Although the three strict convex combinations deliver similar results, `0.5·OV+0.5·MP` seems to be slightly superior, as it comes out on top in solved, gap, and pace.

Given the results in Table 2.5, we see that `OV` and `MP` are particularly bad at gap and time, respectively. Thus, one may wonder whether the superior performance of the strict convex combinations comes from the fact they just avoid sticking to `OV` or `MP` in problems where they do not perform well. Put differently, would a hypothetical version of the solver capable of choosing the best performing approach between `OV` and `MP` outperform the strict convex combinations? The answer to this question is negative. Table 2.6 shows, for instance, that the aforementioned hypothetical configuration would be within 5% of the best performing approach in time in, at most, 96 out of the 138 reported instances, whereas approach `0.5·OV+0.5·MP` on its own is within 5% of the best performing one in 102 instances.

2.5. Combining the enhancements

As a complement to the individual analysis in the preceding subsections, regarding the performance of the different approaches to enhance domain reduction techniques, we now study the impact of incorporating all of them together. Note that, although all the enhancements deal with different aspects of domain reduction, they are essentially independent from one another, so there should be no

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 2.6. Frequency with which each branching point approach is within 5% of the best one.

(403 instances)	(110) 5% Gap	(138) 5% Time	(250) 5% Pace	(280) 5% Nodes
`Baseline`	65	59	137	151
`0.75·OV+0.25·MP`	82	102	193	207
`0.5·OV+0.5·MP`	100	81	180	195
`0.25·OV+0.75·MP`	99	72	174	158
`MP`	97	37	140	119

detrimental effect on performance when combining them: i) the enhancements in Section 2.2 only affect the OBBT at the root node, ii) the enhancements in Section 2.3 just affect the FBBT at all nodes, and iii) the enhancements in Section 2.4 deal with the branching point selection, which is independent of the OBBT and FBBT functionalities. For the sake of exposition, we just report the performance of combinations of the best performing approaches for each individual enhancement: $\texttt{SOCP}^{M}$, $\texttt{FBBT}^{OB+NB}$, and `0.5·OV+0.5·MP`. Yet, it is important to note that retrieving dual information for (nonlinear) conic problems is far from straightforward. Thus, whenever a combination uses $\texttt{SOCP}^{M}$, we replace $\texttt{FBBT}^{OB+NB}$ with $\texttt{FBBT}^{NB}$, the best performing approach in Section 2.3 once we disregard the approaches applying $\texttt{FBBT}^{OB}$.

TABLE 2.7. Performance of RAPOSa combining the enhancements.

(403 instances)	(403) Solved	(112) Gap	(146) Time	(257) Pace
`Baseline`	286	0.120	83.07	7.62
$\texttt{SOCP}^{M}$	286	0.116	93.33	8.05
$\texttt{FBBT}^{OB+NB}$	287	0.115	81.45	7.45
`0.5·OV+0.5·MP`	288	0.105	82.48	7.22
$\texttt{SOCP}^{M}$ + $\texttt{FBBT}^{NB}$	287	0.115	94.05	8.09
$\texttt{SOCP}^{M}$ + `0.5·OV+0.5·MP`	289	0.105	90.19	7.65
$\texttt{FBBT}^{OB+NB}$ + `0.5·OV+0.5·MP`	287	0.104	81.73	7.11
$\texttt{SOCP}^{M}$ + $\texttt{FBBT}^{NB}$ + `0.5·OV+0.5·MP`	287	0.106	94.88	7.88

In Table 2.7 we can see that, as expected from the previous results, the combinations that use approach `0.5·OV+0.5·MP` are the most competitive ones. Among them, probably $\texttt{FBBT}^{OB+NB}$ + `0.5·OV+0.5·MP` is slightly superior to the rest, with the raw `0.5·OV+0.5·MP` performing very similarly. Interestingly, the combination $\texttt{SOCP}^{M}$ + `0.5·OV+0.5·MP` is the one that solves most instances, despite not being so competitive in time.

The main takeaway message we have after the numerical results reported in this section, both for the individual enhancements and for their combinations, is that all enhancements seem to improve performance, although not to the same extent. Interestingly, the less sophisticated enhancement (and the easiest to implement), the

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

strategy for the selection of the branching point, is the one with the (significantly) largest impact.

2.6. Conclusions

We have studied the impact of several approaches to enhance the efficacy of domain reduction functionalities within the specific context of the RLT technique for polynomial optimization problems. The presented results showed that, although all these enhancements lead to performance improvements, the overall impact may sometimes be relatively small. Interestingly, the largest impact comes from the approaches to select the branching point which are, by far, the simplest and easiest to implement among all the enhancements under consideration. We believe these kind of insights on the trade-offs between performance improvement and implementation costs are very relevant for the design of more efficient optimization solvers, since they can guide the implementation efforts of the developing teams of state-of-the-art solvers towards the most promising directions. When a new technique or algorithmic enhancement is first studied, it is natural to analyze is impact in isolation, in order to better quantify its impact. Yet, because of this isolated analyses, it is often hard to assess which of a series of potential enhancements should be prioritized when designing a new solver or when evolving an already existing one, since the impacts individually reported for each of them might no be easy to compare to one another.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 3

Interactions of domain reduction with the linear solver

3.1. Introduction

In recent years there has been a rise in the research in global optimization and, in particular, in the development of global optimization solvers for problems that are both nonlinear and nonconvex, for which local optimality conditions do not guarantee good quality solutions. State-of-the-art commercial solvers for MILP problems such as Gurobi [64] and Xpress [40] can now solve MINLP problems to certified global optimality. Octeract [99] is another recent global solver, which has joined the pool of well established ones such as ANTIGONE [91], BARON [110], Couenne [7], LindoGlobal [83], and SCIP [10].

The single most important element of the branch-and-bound algorithms behind the above solvers is probably the approach taken to define the linear and/or convex relaxations of the underlying nonconvex problem. These relaxations are then efficiently solved at the nodes of the tree, along which finer and finer partitions of the feasible region are defined. Devising relaxation strategies that are applicable to general MINLP problems is a challenging task, which has led to a rich literature with a wide variety of mathematical developments (refer to [125] for a comprehensive coverage). In a series of seminal papers (see, for instance, [108, 109, 126, 124]), the research team behind BARON demonstrated the potential of factorable programming [89] to produce linear relaxations whose solution is both computationally efficient and numerically robust.[1] Since its adoption in [108] and [109], factorable programming reformulations and factorable programming relaxations (usually polyhedral) have been at the core of essentially any general MINLP solver.

The growing interest in global optimization is also present in the development of specialized algorithms for particular classes on nonlinear problems. One of the most important such classes, because of the richness of its applications, is polynomial programming, where RLT-POS [30] and, more recently, RAPOSa (introduced in Section 1.8) are two specialized solvers that ensure global optimality. These solvers build upon the reformulation-linearization technique, RLT, introduced in [120] for polynomial optimization problems. This technique defines a reformulation that maps the problem monomials to variables in a lifted space, and which readily leads to linear relaxations of the original problem that are then embedded into the branch-and-bound scheme.

[1] Recently, in [110], the same research team has developed, and integrated in BARON, a framework that combines polyhedral and convex nonlinear relaxations. Importantly, they devised fail-safe techniques to control for numerical issues in the solution of nonlinear relaxations and revert to the more robust polyhedral based relaxations when needed.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Importantly, both of the aforementioned approaches, factorable programming and RLT, require to introduce a large number of auxiliary variables to construct suitable reformulations of the original problem and, hence, the ensuing relaxations are defined in lifted spaces. Since these lifted spaces are typically of a much larger dimension than the original problem, also the search space for the branch-and-bound algorithm is much larger. This observation motivates another fundamental element behind the efficient design of global optimization algorithms: domain reduction and, more specifically, bound tightening. Bound-tightening ([108, 126, 7, 105]) refers to a family of methods designed to reduce the search space by adjusting the bounds of the variables. As described in Section 1.4, the two most prominent methods are 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 exploiting the relations between the problem constraints and the bounds of the variables.

In this chapter we are concerned with an apparently minor aspect of bound tightening in lifted spaces which, to the best of our knowledge, has not been studied by past literature: potential trade-offs when tightening the bounds of the auxiliary variables. Typically, solvers based on factorable reformulations run the branch-and-bound algorithm in the lifted space, and branching decisions make no distinction between original and auxiliary variables. Yet, given the tight connections between them in the reformulated problem, one might expect that the bounds on the original variables implicitly propagate to the auxiliary ones and so one might wonder to what extent it is necessary to explicitly define and subsequently tighten the bounds of the auxiliary variables. On the one hand, tightening them must lead to smaller trees but, on the other hand, it is not clear what is the impact that making all these bounds explicit may have on the performance of the auxiliary solver used at the different nodes of the tree. In order to analyze this trade-off, we focus on the RLT technique for polynomial optimization. There are two main reasons for this choice. First, in the RLT technique branching takes place in the original variables only and, therefore, convergence does not get compromised if bounds on the auxiliary variables are not tightened or even passed to the linear solver. Second, the linear relaxations solved at the nodes of the tree are tightened with the so called bound-factor constraints, which essentially ensure that the bounds on the original variables directly translate into easily computable bounds for the auxiliary variables. This implies, in particular, that the tightening of the auxiliary variables should not impact the size of the resulting branch-and-bound tree (beyond discrepancies due to different branching decisions after solving relaxations with multiple optimal solutions).

Given the above discussion, the main goal of this chapter is to gain understanding on the role of tightening the bounds of the auxiliary variables in spatial branch-and-bound algorithms. In order to do so, we conduct a relatively simple numerical analysis. We compare the performance of different configurations of an RLT-based solver, depending on how the bounds of the auxiliary variables are handled and also depending on whether or not bound-tightening techniques are used. We find that the impact of explicitly incorporating the bounds of the auxiliary variables is significant and, somewhat surprisingly, it can go in any direction with no clear pattern behind it. More precisely, we find that:

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- The direction and the magnitude of the impact on performance varies significantly across linear solvers.
- The direction of the impact varies across test sets. Even after restricting attention to a specific linear solver, the direction of the impact may vary across test sets.
- The direction and the magnitude of the impact also depend on whether or not bound tightening techniques were enabled in the RLT-based solver. Further, the direction of the impact may again vary across solvers and test sets.

Moreover, it is worth noting that the aforementioned impact is significant for different measures of performance such as solve time, gap, number of nodes of the tree, and solve time of the linear relaxations. The fact that we have not been able to find any pattern behind the above behavior definitely calls for further research on this topic. As a first step in this direction, Section 5.2.3 is devoted to study the potential of learning techniques to predict, on a given problem, what approach regarding the bounds of the auxiliary variables will perform better. We build upon the general learning framework developed in [54] and, although we obtain some performance improvements with their approach, it would be worth exploring how much further one might go with learning techniques tailored to this specific setting.

In a nutshell, the contribution of this chapter can be seen as raising awareness on the potentially high impact that an a apparently minor decision such as the specification or not of bounds for the variables in the lifted space can have. Importantly, this impact may heavily depend on the auxiliary solver used to handle the relaxations.

The rest of the chapter is structured as follows. In Section 3.2 we present the general polynomial optimization framework, including the RLT-technique and some considerations regarding bound tightening. Section 3.3 is devoted to the main computational experiments. All of the results presented in this chapter are computed using the computational environment described in Section 1.9, using test sets MINLPLIB and DS. We conclude in Section 3.4. In Section 5.5 we present the learning framework and the associated results. This chapter is a joint work with Brais González Rodríguez and Julio González Díaz.

3.2. Framework for the analysis

3.2.1. Spatial branch-and-bound algorithm. We develop our analysis in the class of polynomial optimization problems and the reformulation-linearization technique described in Chapter 1. Note that, by definition, RLT variables are bounded by the products of the bounds of the original variables present in it. More precisely, for each multiset J, we have that $\prod_{j\in J} l_j \leq X_J \leq \prod_{j\in J} u_j$. Yet, since constraint (1) is not present in $LP(\Omega)$, it might be useful to tighten the relaxations by imposing explicitly these bounds on the RLT variables. Recall that, our main goal is precisely to assess the impact of this tightening of $LP(\Omega)$. Hereafter we use `RLT-looseB` and `RLT-tightB` to refer to configurations of the RLT-based algorithm in which the bounds on RLT variables are omitted or specified, respectively.

3.2.2. Bound tightening. In our computational analysis we are interested in two different aspects related to bound tightening. The first one is the impact of specifying and/or tightening the bounds of the RLT variables and the second one

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

is the interplay of the former with the general bound tightening techniques that may be available in a given RLT-based algorithm. We separately discuss these two aspects below.

3.2.2.1. *Bounds on auxiliary variables.* As discussed in the introduction, moving from `RLT-looseB` to `RLT-tightB` might lead to two main effects: i) impact on the solving time of the linear relaxations due to the large number of additional bounds passed to the linear solver and ii) impact on the size of the resulting branch-and-bound tree due to the additional tightness of the relaxations. We argue below that, for the particular case of RLT-based algorithms, the latter effect should have a relatively limited impact. Importantly, from the theoretical standpoint, no approach, `RLT-looseB` or `RLT-tightB`, should be expected to systematically produce smaller trees.

In Chapter 1 we introduced the following results:

LEMMA (Lemma 1.1). *Let $B \subseteq N$ be a subset of bounded variables in Ω. Let δ' be such that $1 \leq \delta' < \delta$ and let J_1, J_2 be two multisets such that $J_1 + J_2 \in N^{\delta'}$, $\text{supp}(J_2) \subseteq B$. Then, the bound-factor constraint $[F_{\delta'}(J_1, J_2)]_L \geq 0$ is implied by the bound-factor constraints $[F_\delta(J_3, J_4)]_L \geq 0$ with $J_3 + J_4 \in N^\delta$, $\text{supp}(J_4) \subseteq B$.*

COROLLARY (Corollary 1.2). *In the feasible region of $LP(\Omega)$, all X_J variables with $J \in B^\delta_\leq$ are bounded.*

Now, it is straightforward to show that, for quadratic problems, the RLT variables have, as lower (upper) bounds the products of the lower (upper) bounds of the variables present in them.

COROLLARY 3.1. *The bound-factor constraints of degree 2 imply that, for each monomial J of degree 2, $\prod_{j\in J} l_j \leq X_J \leq \prod_{j\in J} u_j$.*

PROOF. It follows immediately from the proof of Corollary 1.2 for the case $|J| = 2$, since $X_j \geq l_{j_2}x_{j_1} + l_{j_1}x_{j_2} - l_{j_1}l_{j_2} \geq l_{j_2}l_{j_1} + l_{j_1}l_{j_2} - l_{j_1}l_{j_2} = l_{j_1}l_{j_2}$. For the upper bound, a similar argument can be used to show that $X_J \leq u_{j_2}x_{j_1} + l_{j_1}x_{j_2} - l_{j_1}u_{j_2} \leq u_{j_2}u_{j_1} + l_{j_1}u_{j_2} - l_{j_1}u_{j_2} = u_{j_2}u_{j_1}$. □

Corollary 3.1 has important implications for our analysis, since it implies that, for quadratic problems, explicitly specifying the bounds of the RLT variables does not change the feasible set of $LP(\Omega)$, since these additional bounds are redundant. Therefore, if all subproblems solved by the branch-and-bound algorithm had a unique optimum, the explicit inclusion of the bounds of the RLT variables would not change at all the resulting tree. Further, it should not be hard to build upon the induction arguments in the proof of Corollary 1.2 to generalize Corollary 3.1 to variables of any degree. Yet, in practice, even for quadratic problems and under uniqueness of the optimal solution of the linear relaxations, one might get different trees because of numerical precision when solving the two equivalent formulations of $LP(\Omega)$ (with and without explicit bounds). More importantly, the high dimensionality of $LP(\Omega)$, induced precisely by the RLT variables, facilitates the existence of multiple optima. Therefore, despite Corollary 3.1 and its potential generalization to RLT variables of any degree, moving from `RLT-looseB` to `RLT-tightB` might still impact not only the solving time of the relaxations, but also the size of the resulting tree and, moreover, these differences might go in any direction.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

3.2.2.2. *OBBT and FBBT.* As already discussed in the introduction (and in Section 1.4), OBBT and FBBT are usually combined in such a way that OBBT is only run at specified points of the algorithm and FBBT is run at all nodes. This is the approach taken by the RLT-based algorithm used in our computational analysis:

OBBT.: It is run only at the root node by solving two optimization problems for each variable $j \in N$. These problems consist of minimizing and maximizing x_j subject to the feasible region of problem $LP(\Omega)$.

FBBT.: It is run at all nodes of the tree by using constraint propagation techniques on $PP(\Omega)$, following the interval arithmetic approach in [8].

Note that, given that OBBT is run on problems with the same feasible region of $LP(\Omega)$, Corollary 3.1 and the above discussion would again imply that, at least for quadratic problems, the bounds obtained for the original variables during the OBBT phase are the same with `RLT-looseB` and with `RLT-tightB`. Moreover, since FBBT is run directly on the nonlinear problem $PP(\Omega)$, the RLT variables are not present and, therefore, the resulting bounds are the same with `RLT-looseB` and with `RLT-tightB`.

Therefore, there is no reason to expect that the potential differences in performance between `RLT-looseB` and `RLT-tightB` might depend on whether or not bound tightening is enabled in the RLT-based algorithm.

3.3. Computational experiments

In this section we compare the performance of different configurations of the RLT-based algorithm RAPOSa, with the focus being on the differences in performance between approaches `RLT-looseB` and `RLT-tightB`. Further, in order to assess whether the differences between these two approaches might be dependent on the auxiliary linear solver, all instances are solved with the linear solvers available from Google OR-Tools [104]. More precisely, we use Gurobi [64], Google's proprietary LP solver Glop [104], Clp [41], Xpress [40], and CPLEX [70]. Lastly, in order to assess potential interactions of `RLT-looseB` and `RLT-tightB` with bound tightening, the numerical analysis is run both with and without enabling RAPOSa's bound tightening capabilities, as described in Section 3.2.2.

Table 3.1 presents the results for the executions without bound tightening, separated by test set, MINLP or DS, and by linear solver. We report the results taking `RLT-looseB` as the reference approach, and so the numbers in the table represent the relative change in performance when moving from `RLT-looseB` to `RLT-tightB`: positive values imply that an increase, deterioration, in performance for `RLT-tightB` and negative values (in bold), indicate that `RLT-tightB` outperformed `RLT-looseB`. Column "Instances" contains the number of instances used to compute each performance measure in each test set, after applying the exclusions described in Section 1.9. In particular, the number of instances in row "Unsolved" is the total number of instances used for that test set (some instances that turned out to be problematic for some solver were removed).

It is worth noting that we have chosen to report the results in this manner because it does not allow to compare the relative performance of the different linear solvers with respect to each other, which might distract attention from the goal of the study. To this end, the results in Table 3.1 just allow to compare the

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 3.1. Results without bound tightening.

	Instances		Gurobi		Glop	
	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	165	179	**-3.9**	21.4	9.1	**-38.5**
Gap	57	55	**-18.0**	12.3	41.9	**-69.0**
Time	89	145	45.0	72.9	24.8	**-39.7**
Pace	131	158	36.8	65.4	61.4	**-39.1**
Nodes	101	124	**-0.1**	**-0.5**	**-5.7**	2.8
L-Time	101	124	573.7	169.4	27.2	**-39.1**
	Clp		Xpress		CPLEX	
	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	**-4.6**	**-20.5**	0.0	**-18.8**	0.0	0.0
Gap	**-18.3**	**-58.2**	12.2	**-36.9**	0.6	3.1
Time	39.1	**-26.6**	6.0	**-3.5**	**-3.9**	8.8
Pace	21.3	**-26.7**	**-10.8**	**-3.8**	**-3.1**	8.1
Nodes	1.0	3.1	8.9	0.3	**-1.7**	0.5
L-Time	3.3	**-42.5**	**-3.0**	28.3	**-6.6**	9.0

relative impact of the tightening of the RLT bounds for the different solvers, but no comparison of their absolute performance is possible.

Table 3.1 shows that anything can happen. With Gurobi, `RLT-looseB` is preferable for DS, but results are somewhat mixed for MINLP. Yet, for both test sets, `RLT-tightB` increases the solve time of the linear relaxation quite significantly. The situation practically gets reversed with Glop, for which `RLT-tightB` clearly outperforms `RLT-looseB` for DS, but gets clearly outperformed in MINLP. Regarding linear solve time, it gets down in DS but it goes up in MINLP. Looking at the results for Clp, Xpress, and CPLEX just confirm that the impact of tightening the bounds of the RLT variables can go in any direction which, moreover, may depend on the test set and on the specific performance metric. The only additional insight may be that Xpress and, particularly CPLEX, seem to be more insensitive to the use of `RLT-looseB` or `RLT-tightB`.

These results suggest that the way in which variable bounds are handled internally by the different solvers has a big impact on performance. Yet, we have not been able to find any pattern that may explain the disparate performance results for `RLT-looseB` and `RLT-tightB` in Table 3.1.

Table 3.2 contains the results of the same numerical analysis, but with bound tightening enabled in RAPOSa. The results now look completely different from those in Table 3.1 but, at the same time, they are qualitatively identical, since anything can happen in terms of the relative performance of `RLT-looseB` and `RLT-tightB`. Interestingly, with bound tightening it is now Gurobi the solver less sensitive to the approach taken for the bounds of the RLT variables. Glop and Clp's results are quite similar with and without bound tightening but, on the other hand, CPLEX is now very sensitive to the use of `RLT-looseB` or `RLT-tightB`, with the latter leading to substantial improvements in performance.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 3.2. Results with bound tightening.

	Instances		Gurobi		Glop	
	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	165	174	2.4	18.8	5.4	**-27.8**
Gap	58	49	0.9	9.0	21.7	**-60.0**
Time	113	155	3.6	5.5	9.4	**-37.8**
Pace	150	170	0.0	5.1	8.2	**-30.4**
Nodes	101	125	**-1.2**	0.0	**-1.0**	**-0.1**
L-Time	101	125	**-8.6**	3.4	**-14.8**	**-39.7**
	Clp		Xpress		CPLEX	
	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	2.6	**-13.5**	0.0	0.0	0.0	0.0
Gap	7.2	**-44.1**	**-10.9**	13.6	0.1	1.7
Time	123.0	**-12.2**	8.1	3.3	**-52.4**	**-21.0**
Pace	82.8	**-4.1**	**-2.6**	3.9	**-42.8**	**-19.3**
Nodes	**-0.6**	0.4	**-0.4**	**-0.1**	**-1.3**	**-0.4**
L-Time	1.3	**-34.4**	0.3	6.0	**-1.2**	**-1.3**

3.4. Conclusions

We have not been able to find any compelling rationale behind the results in Table 3.1 and Table 3.2. Yet, given the size of the variability in the different measures of performance, it definitely seems important to develop additional studies to gain some understanding on the observed behavior. These additional studies might even go beyond the RLT technique and polynomial optimization problems, and study whether similar effects might arise for other classes of problems and branch-and-bound schemes. In Section 5.5 we try to shed some light on this by applying ML to try and anticipate which of the approaches works best for a particular problem. Analyzing the resulting models could allow the extraction of information to shed some light on what makes the performance of either of the approaches so different for different solvers and problems.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 4

Extending RLT to solve unbounded problems

4.1. Introduction

In Chapter 1 the RLT technique was introduced. It is an algorithm designed to solve general polynomial optimization problems, as long as the problems have finite bounds on all the variables. The convergence of the algorithm is only proven for that case. A priori, this requirement may be seen as a reasonable compromise: most problems in real life applications have finite bounds on the variables imposed by the resources available or the physics of the problem at hand. Nonetheless, from the point of view of solver development, having a solver that can handle problems as general as possible is an attractive goal. For instance, BARON [110] (a state-of-the-art general nonlinear solver) allows the user to enter problems without bounds for all the variables, although it does not guarantee convergence to a global optimum if "the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued".

There is a first approach to allow unbounded variables that is just to run bound-tightening on the problem, hoping to bound the unbounded variables, and if that is not successful just stop solving and return an error message to the user. Even if something more complex is implemented, this should be the first step when dealing with problems with unbounded variables, as it may remove the unboundedness altogether and the solver may be able to proceed without any additional modifications. But it can happen bound-tightening does not find bounds for the unbounded variables. Then the aim should be for the solver to proceed and return some result to the user.

For a polynomial problem with unbounded variables, and in the context of a branch-and-bound base algorithm, one of the following scenarios must occur:

(1) Both the problem and the root-node relaxation have finite optima.
(2) The problem has a finite optimum, but the relaxation is unbounded.
(3) Both the problem and the relaxation are unbounded.

The ideal situation for a solver would be to be able to handle these three cases. For Case (3), the solver needs to detect and prove the unboundedness of the problem; for Case (2) it needs to "fix" the relaxation and continue with the solving process and return the global optimum; and for Case (1) it needs to solve the problem and return the global optimum. In this chapter, we focus on Case (1), adapting the algorithm `plainRLT` to work with unbounded variables and proving the optimality of the solution if one is found.

In our proposal there is a substancial difference with respect to `plainRLT`. Since we are no longer working on a feasible region contained in a compact set, we don't have theoretical guarantees that the algorithm converges, something we did have for `plainRLT`. Our hope is that, in practice, convergence is attained, but it may be the

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

case that, even if the problem has a finite optimum, the algorithm does not converge to any of the optima of the problem. Nonetheless, to the best of our knowledge, this is the first time a global optimization algorithm for unbounded problems comes with some theoretical results substantiating its convergence guarantees and the requires assumptions.

The rest of the chapter is structured as follows. In Section 4.2 we present the modifications made to `plainRLT` for it to be able to handle unbounded variables; in Section 4.3 we adapt the proof in [58] to this modified algorithm; in Section 4.4 we analyze the possible impact the enhancements to the RLT technique discussed in Chapter 1 and Chapter 2 can have on the theoretical guarantees of convergence; and finally in Section 4.5 we present some numerical results on the performance of the variation of RAPOSa to handle problems with unbounded variables. This chapter is a joint work with Brais González Rodríguez, Julio González Díaz and Pietro Belotti.

4.2. Extension to unbounded problems

As we have seen in Section 1.2, the RLT technique `plainRLT`, introduced in [120], was designed to solve polynomial optimization problems with bounded variables, that is, with the feasible region contained in a hyperrectangle. In this chapter we propose an extension to this algorithm to handle polynomial optimization problems with possibly unbounded domains. We show that with minor modifications to the algorithm it can be proved that, if convergence is attained, the limit point is a global optimum of the polynomial problem. For the sake of exposition, throughout this section we assume that the upper bounds are the only source of unboundedness, i.e., all variables have finite lower bounds. All the results presented in this chapter can be easily extended to the general case in which variables may be unbounded from below. Then, we work with a modified version of $PP(\Omega)$, where the upper bounds of the variables may not be finite:

$$PP(\dot{\Omega}) \qquad \begin{aligned} \text{minimize} \quad & \phi_0(\mathbf{x}) \\ \text{subject to} \quad & \phi_r(\mathbf{x}) \geq b_r, \qquad r = 1, 2, \ldots, m_1 \\ & \phi_r(\mathbf{x}) = b_r, \qquad r = m_1 + 1, \ldots, m \\ & \mathbf{x} \in \dot{\Omega} \subset \mathbb{R}^{|N|}, \end{aligned}$$

where $\dot{\Omega} = \{\mathbf{x} \in \mathbb{R}^{|N|} : 0 \leq l_j \leq x_j \leq u_j \leq \infty, \text{ for all } j \in N\} \subset \mathbb{R}^{|N|}$ is the feasible region. We no longer have a hyperrectangle but an "improper" hyperrectangle as some of the bounds may be infinite. We abuse terminology and also refer to $\dot{\Omega}$ as a hyperrectangle.

We introduce two modifications to `plainRLT`. The first one is just a necessity, since it makes no sense to define bound-factors associated to infinite bounds. The second one is where the key to the extension resides, as it will ensure that, along the infinite branches of the tree, the desired boundedness is recovered. The resulting modified algorithm, which hereafter is referred to as `unboundedRLT`, is summarized in Figure 4.1. The modifications with respect to algorithm `plainRLT` in Figure 1.1 are shown in gray.

Modification 1. Let $U = \{j \in N \text{ s.t. } u_j = \infty\}$ be the set of variables without a finite upper bound and $B = \{j \in N \text{ s.t. } u_j < \infty\}$. Then, in the linearization $LP(\dot{\Omega})$, only bound-factor constraints that involve variables in B are included.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Parameter initialization. Take $\Delta > 0$.

Initialization. Let $LB := -\infty$ and $UB := +\infty$ be the lower and upper bounds of the algorithm. Let $\dot{\Omega}^1 := \dot{\Omega}$ be the root node's hyperrectangle; $LB^1 := \infty$ the root node's potential; $Q := \{1\}$ the queue of problems to solve; $\mathcal{N}^1 := \emptyset$ a set of variables for the hyperrectangle expansion phase; $\sigma^1 := 1$ the current reference node; $\gamma^1 := 0$ the node's depth; $\tau := 1$ a counter for the problems; and $\boldsymbol{x}^{best} := \emptyset$ the current best solution.

For $j \in N$ and $k \in \mathbb{N}$, let l_j^k and u_j^k denote the lower and upper bound of variable x_j in $LP(\dot{\Omega}^k)$ and let $U^k := \{j \in N \text{ s.t. } u_j^k = \infty\} \subseteq N$.

Stage 1 ***(main).*** Choose $k \in Q$ such that $LB^k = \min_{s \in Q} LB^s$. Let $Q := Q \backslash \{k\}$. Solve problem $LP(\dot{\Omega}^k)$.

- If $LP(\dot{\Omega}^k)$ is infeasible, go to **Stage 2**.
- If $LP(\dot{\Omega}^k)$ is feasible, an optimal solution, $(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k)$, and an optimal value of the objective function, $\bar{z}^k$, are obtained. Compute θ_j^k for $j \in N$ as in (3).
  - If $\bar{z}^k < UB$ and $\theta_j^k = 0$ for all $j \in N$:
    - ◇ *Update upper bound.* $UB := \bar{z}^k$. Let $\boldsymbol{x}^{best} := \bar{\boldsymbol{x}}^k$.
    - ◇ *Prune.* Remove all $s \in Q$ such that $LB^s \geq UB$.
    - ◇ Go to **Stage 2**.
  - If $\bar{z}^k < UB$ and $\theta_j^k > 0$ for some $j \in N$:
    - ◇ If $U^k \neq \emptyset$, $\mathcal{N}^k = \emptyset$ and $\gamma^k = d \cdot |N|$ for some $d \in \mathbb{N}$: let $\mathcal{N}^k := \{j \in U^k \text{ s.t. } |l_j^k - l_j^{\sigma^k}| < \Delta\}$.
    - ◇ *Select branching variable.*
      - · *Hyperrectangle expansion.* If $\mathcal{N}^k \neq \emptyset$, choose $p \in \mathcal{N}^k$. Branch at $\beta := l_p^k + \Delta$.
      - · *Standard branching.* If $\mathcal{N}^k = \emptyset$, choose $p \in N$ such that $\theta_p^k = \max_{j \in N} \theta_j^k$. Branch at $\beta := \bar{x}_p^k$.
    - ◇ *Branch.* Let
    $$\dot{\Omega}^{\tau+1} := \dot{\Omega}^k \cap \{\boldsymbol{x} \in \mathbb{R}^{|N|} \text{ s.t. } l_p^k \leq x_p \leq \beta\},$$
    $$\dot{\Omega}^{\tau+2} := \dot{\Omega}^k \cap \{\boldsymbol{x} \in \mathbb{R}^{|N|} \text{ s.t. } \beta \leq x_p \leq u_p^k\}.$$
    - ◇ *Define reference nodes.*
      - · If $\mathcal{N}^k \neq \emptyset$: let $\sigma^{\tau+1} := \tau + 1$, $\sigma^{\tau+2} := \tau + 2$. Let $\mathcal{N}^k := \mathcal{N}^k \backslash \{p\}$.
      - · If $\mathcal{N}^k = \emptyset$: let $\sigma^{\tau+1} = \sigma^{\tau+2} := \sigma^k$.
    - ◇ *Update depth.*
      - · If $\mathcal{N}^k \neq \emptyset$: let $\gamma^{\tau+1} = \gamma^{\tau+2} := \gamma^k$.
      - · If $\mathcal{N}^k = \emptyset$: let $\gamma^{\tau+1} = \gamma^{\tau+2} := \gamma^k + 1$.
    - ◇ Let $\mathcal{N}^{\tau+1} = \mathcal{N}^{\tau+2} := \mathcal{N}^k$. Let $Q := Q \cup \{\tau+1, \tau+2\}$ and $\tau := \tau + 2$. Let $LB^{\tau+1} = LB^{\tau+2} := \bar{z}^k$.
    - ◇ Go to **Stage 2**.
  - If $\bar{z}^j \geq UB$. *Prune branch.* Go to **Stage 2**.

Stage 2 ***(control).*** *Update lower bound.* $LB := \min\{\min_{k \in Q} LB^k, UB\}$.[a]

- If $LB = UB$, END:
  - If $\boldsymbol{x}^{best} = \emptyset$, $PP(\Omega)$ is infeasible.
  - If $\boldsymbol{x}^{best} \neq \emptyset$, $\boldsymbol{x}^{best}$ is a global optimum and UB is the optimal value.
- In other case, go to **Stage 1**.

[a]Note that the minimum over an empty set is defined as infinite.

FIGURE 4.1. Algorithm `unboundedRLT`

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Whenever a variable in U gets assigned a finite upper bound at some node, the bound-factor constraints are updated adding the newly available ones from that node onwards. The linearization of $PP(\dot{\Omega})$ is then defined as:

$$
LP(\dot{\Omega}) \quad \begin{array}{lll}
\text{minimize} & [\phi_0(\mathbf{x})]_L & \\
\text{subject to} & [\phi_r(\mathbf{x})]_L \geq b_r, & r = 1, 2, \ldots, m_1 \\
& [\phi_r(\mathbf{x})]_L = b_r, & r = m_1 + 1, \ldots, m \\
& [F_\delta(J_1, J_2)]_L \geq 0, & J_1 + J_2 \in N^\delta,\ \mathrm{supp}(J_2) \subseteq B \\
& \mathbf{x} \in \dot{\Omega} \subset \mathbb{R}^{|N|}. &
\end{array}
$$

Note that $\dot{\Omega}$ has a double role in $LP(\dot{\Omega})$, since it not only represents the hyperrectangle containing the feasible region, but also determines the bound-factor constraints.

Modification 2 (Hyperrectangle expansion phase). In the branch-and-bound tree, in nodes with depth $d \cdot |N|$ with $d \in \mathbb{N}$, we do the following:

- For each variable x_j such that $u_j = \infty$, we check the increment in the lower bound (increment with respect to the last node of the current branch in which hyperrectangle expansion was applied). We select the variables where the increment was smaller than a fixed $\Delta > 0$.
- For all those variables, we use them as branching variables (ignoring their RLT violations), branching at $l_j^k + \Delta$.
- This ensures that, every $|N|$ iterations in each branch, the lower bound of the variables without upper bound raises at least Δ.

Importantly, when computing the depth of a node, the nodes in hyperrectangle expansion phases are not taken into account.

As we already mentioned, Modification 2 is necessary to enforce some bounding of the variables as reflected in the following lemma.

Lemma 4.1. *Suppose that algorithm* ***unboundedRLT*** *is applied to* $PP(\dot{\Omega})$ *and let* $\Gamma > 0$. *Then, there exists* $\tilde{\gamma} \in \mathbb{N}$ *such that, for all nodes* $k \in \mathbb{N}$ *with depth* $\gamma^k > \tilde{\gamma}$ *and for all variables* $j \in N$, *if* $u_j^k = \infty$, *then* $l_j^k > \Gamma$.

Proof. By construction, along any branch of the tree, every $|N|$ nodes not belonging to the hyperrectangle expansion phase, the lower bound of each unbounded variable increases by at least Δ. For each $j \in N$, let $\Gamma_j = \Gamma - l_j^1$, and let $\gamma_j = |N| \cdot \lceil \Gamma_j / \Delta \rceil$. Now, consider a node k at depth $\gamma^k \geq \gamma_j$. Then, either variable j is bounded at the current node, or its lower bound has increased by at least Γ_j, and so $l_j^k \geq \Gamma$. To conclude the proof it suffices to take $\tilde{\gamma} = \max_{j \in N} \gamma_j$. □

Lemma 4.1, combined with results Lemma 1.1, Corollary 1.2, Corollary 1.3 and Lemma 1.4, provide all the necesary ingredients to prove the convergence result for `unboundedRLT`, i.e., the extension of Theorem 1.5 to unbounded problems. Lemma 1.1, Corollary 1.2 and Corollary 1.3 can all be applied directly in the proof, since we apply them when all of the variables already have a finite upper bound. On the other hand, Lemma 1.4 can also be applied, as the arguments used in the proof in Appendix B are also applicable to `unboundedRLT`, just noticing that the selection of problems from the queue and the branches' pruning is not modified from `plainRLT`. Nonetheless, we include here an updated formulation:

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

LEMMA 4.2. *Suppose* ***unboundedRLT*** *is applied to solve problem* $PP(\dot{\Omega})$. *Let* k *correspond to a relaxation* $LP(\dot{\Omega}^k)$ *solved during the course of the algorithm. Then,* $LB^k \leq \nu[PP(\dot{\Omega})]$.

4.3. Convergence result

The aim of this section is to introduce a result similar to Theorem 1.5 but extended for the algorithm `unboundedRLT`. Here we will encounter two main issues. The first one is that, even thought we are assuming $PP(\dot{\Omega})$ is feasible (as was also done in Theorem 1.5), we don't have guarantees that the problem has a finite global optimum. In $PP(\Omega)$, since the feasible region was bounded, once the problem is feasible we know there is at least one finite optimum. But when removing some or all bounds from the variables this statement is no longer true. Consequently, we need to impose some additional condition to $PP(\dot{\Omega})$, and in particular we will put the condition on $LP(\dot{\Omega})$, since we also need to have a finite optimum in the relaxed problem for the algorithm to be able to continue.

The second issue we need to deal with is that there may not be any accumulation point in any of the infinite branches of the tree. In Theorem 1.5 we implicitly assume there is at least one (its existence is proven in [58]), but because we are not working inside a compact region anymore, we can't assume the same here. That is the reason why we move from "the accumulation points" to "any accumulation point" and therefore arrive to a less strong result.

For both these issues, there is still work that can be done to strengthen the result. For example, to get around the need of a finite optimum in $LP(\dot{\Omega})$, some strategies could be devised to check if the region on which the objective function can grow to infinity is feasible or not in $PP(\dot{\Omega})$. If it is the latter, that region could be excluded from the relaxation with some appropriate cuts and still run the algorithm. Also, regarding the existence of accumulation points, there could be some additional conditions that could be imposed on the problem that would ensure the existence of such points, and so prove the convergence of the algorithm at least for some classes of problems.

We introduce now the convergence result for `unboundedRLT` and its proof.

THEOREM 4.3. *Suppose that* ***unboundedRLT*** *is applied to solve a feasible and possibly unbounded problem* $PP(\dot{\Omega})$. *Then, if* $LP(\dot{\Omega})$ *has a finite optimum, one of the following statements holds:*

(1) The algorithm finishes in a finite number of iterations, being the obtained solution a global optimum of $PP(\dot{\Omega})$.

(2) An infinite sequence of iterations is generated such that, along any infinite branch of the tree, any accumulation point of the associated sequence of variables, $\{\bar{\boldsymbol{x}}\}$, *is a global optimum of* $PP(\dot{\Omega})$.

PROOF. Since we are assuming the root node of the tree has a finite optimum, once the root node is solved we have $LB > -\infty$.

Case (1). Since the algorithm has finished, $UB = LB$ (this only happens when Q is empty). If $UB < \infty$, $\boldsymbol{x}^{best}$ is the global optimum of $PP(\dot{\Omega})$. Otherwise, $UB = LB = \infty$ and $PP(\dot{\Omega})$ is infeasible.

Case (2). Consider an infinite branch and denote by $\{\dot{\Omega}^k\}_{k\in K}$, $K \subset \mathbb{N}$, the associated nested sequence of hyperrectangles. We denote the solution of node $LP(\dot{\Omega}^k)$ by $(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k)$, and consider the sequence $\{\bar{\boldsymbol{x}}^k\}_{k\in K}$. Suppose that $\boldsymbol{x}^*$ is an

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

accumulation point of $\{\bar{\boldsymbol{x}}^k\}_{k\in K}$. We have to prove that $\boldsymbol{x}^*$ is a global optimum of $PP(\dot{\Omega})$.

Since $\boldsymbol{x}^*$ is an accumulation point, there exists a subsequence $\{\bar{\boldsymbol{x}}^k\}_{k\in K_1\subseteq K}$ that converges to $\boldsymbol{x}^*$. Since this subsequence is infinite, there is at least one variable x_p on which the algorithm branches infinitely often in K_1. Let $K_2 \subseteq K_1$ be the set of iterations where the branching variable is x_p.

Now, there is $K_3 \subseteq K_2$ such that, at iterations in K_3, $\theta_p \geq \theta_j$ for all $j \in N$. Thus, for each $k \in K_3$, there exists $J' \in N_-^\delta$ such that

$$\left|\bar{X}^k_{J'\cup\{p\}} - \bar{x}^k_p \bar{X}^k_{J'}\right| \geq \left|\bar{X}^k_{J\cup\{j\}} - \bar{x}^k_j \bar{X}^k_J\right| \tag{13}$$

for all $J \in N_-^\delta$ and for all $j \in N$.

Next, we prove that the sequence $\left\{\left(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k, \boldsymbol{l}^k, \boldsymbol{u}^k\right)\right\}_{k\in K_3}$ is bounded and therefore that it has a convergent subsequence.

- $\{\bar{\boldsymbol{x}}^k\}$ is bounded for $k \in K_3$ as $\{\bar{\boldsymbol{x}}^k\}_{k\in K_3} \to \boldsymbol{x}^*$.
- Because of Lemma 4.1 and the convergence of $\{\bar{\boldsymbol{x}}^k\}_{k\in K_3}$, there exists $\tilde{k} \in \mathbb{N}$ such that, for all $k > \tilde{k}$, $u_j^k < \infty$ for all $j \in N$. Since $\{\boldsymbol{u}^k\}$ is monotonically decreasing and bounded below by $\boldsymbol{l}^k$, it's therefore bounded. This also means that, for $k > \tilde{k}$, $U^k = \emptyset$.
- Consequently, $\{\boldsymbol{l}^k\}$ is bounded since $\boldsymbol{l}_j^k \leq \boldsymbol{u}_j^k$ for all $j \in N$.
- Finally, using Corollary 1.2 and given that once a finite upper bound is set for a variable we add the corresponding bound-factor constraints, $\{\bar{\boldsymbol{X}}^k\}$ is also bounded.

Take the convergent subsequence $\{(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k, \boldsymbol{l}^k, \boldsymbol{u}^k)\}_{k\in K_4\subseteq K_3}$ and its limit point $(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{l}^*, \boldsymbol{u}^*)$. We have that $(\boldsymbol{x}^*, \boldsymbol{X}^*)$ is a feasible point for $LP(\dot{\Omega}^*)$, where $\dot{\Omega}^* = \{\mathbf{x} \in \mathbb{R}^{|N|} : 0 \leq l_j^* \leq x_j \leq u_j^*, \text{ for all } j \in N\} \subseteq \dot{\Omega}$, and therefore feasible in $LP(\dot{\Omega})$. Next, we prove feasibility and optimality of $\boldsymbol{x}^*$ in $PP(\dot{\Omega})$.

Because of the way branching is done, it holds that for each $k, k' \in K_4$, with $k' > k$, $\bar{x}_p^k \notin (l_p^{k'}, u_p^{k'})$. Therefore, since $x_p^* \in [l_p^*, u_p^*]$, we have that $x_p^* = l_p^*$ or $x_p^* = u_p^*$. By Corollary 1.3, we therefore get $X^*_{J'\cup\{p\}} = x_p^* X^*_{J'}$.

Consequently, using Eq. (13) and making k tend to infinity, it holds that $X^*_{J\cup\{j\}} = x_j^* X_J^*$ for each $J \in N_-^\delta$, $j \in N$. Thus, $\boldsymbol{x}^*$ is feasible in $PP(\dot{\Omega})$ and $[\phi_0(\boldsymbol{x}^*, \boldsymbol{X}^*)]_L = \phi_0(\boldsymbol{x}^*) \geq \nu[PP(\dot{\Omega})]$, where $\nu[PP(\dot{\Omega})]$ is the global optimum (we now know that there is one, since we have found a feasible solution and $LP(\dot{\Omega})$ has a finite optimum).

Since all problems in K are eventually solved by the algorithm, by Lemma 4.2, we have that for all $k \in K$, $\nu[PP(\dot{\Omega})] \geq LB^k$. Because for all $k \in K$ with $k > 1$, there exists $s \in K$ with $s < k$, such that $[\phi_0(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k)]_L = LB^s$, it holds that $\nu[PP(\dot{\Omega})] \geq [\phi_0(\bar{\boldsymbol{x}}^k, \bar{\boldsymbol{X}}^k)]_L$ for all $k \in K_4$. Since $[\phi_0(\cdot,\cdot)]_L$ is continuous, it holds that $\nu[PP(\dot{\Omega})] \geq [\phi_0(\boldsymbol{x}^*, \boldsymbol{X}^*)]_L = \phi_0(\boldsymbol{x}^*)$. Thus, $\boldsymbol{x}^*$ is the global optimum of $PP(\dot{\Omega})$. □

The proof of Theorem 4.3 is similar to the one of Theorem 1.5 in [58]. The key point that changes between the two is that in the unbounded case, in order to apply the same arguments to guarantee the global optimality of an accumulation point, we need to ensure we are working in a "bounded" setting. Modification 2 enters here, with its hypercube expansion phase, and causes that after some depth is reached in a branch, we are indeed inside a hyperrectangle as proven in Lemma 4.1.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

From that point onward, the same arguments applied in the proof of Theorem 1.5 can be used here.

4.4. Interaction with other enhancements

To finish up the theoretical part of this chapter, we study how the changes applied to `plainRLT` to extend it for problems with unbounded variables impact the use of some of the enhancements described in Chapter 1. In particular, we want to check if the proof of Theorem 4.3 can be affected by the use of such enhancements.

4.4.1. *J*-sets. The first enhancement discussed is the use of J-sets. As we discussed in Section 1.3, the idea behind the J-sets is to add just the bound-factor constraints necessary to guarantee convergence. The objective is to reduce the size of the linear relaxation as much as possible, since the number of possible bound-factor constraints increases exponentially with the number of variables in the problem. The bound-factors added are only those associated with the monomials that appear in $PP(\dot{\Omega})$, and then this is refined removing those which are not maximal (i.e. their monomials are part of other higher-degree monomials). Regarding the proof of Theorem 4.3, since we are invoking Corollary 1.2 only for bounded variables (using Lemma 4.1, we select those iterators in the sequence that lie inside a hyperrectangle), the result is used the same way as in the proof of Theorem 1.5. This fact implies that the use of J-sets, which didn't compromise the result of Theorem 1.5, doesn't compromise the result of Theorem 4.3 either.

4.4.2. Bound tightening. The second enhancement mentioned in Chapter 1 is bound tightening. This technique allows to infer new bounds for the variables using the information from the bounds of the other variables and from the constraints of the corresponding optimization problem. We apply both the OBBT and the FBBT techniques described in Section 1.4. OBBT computes the best bounds for the variables in $PP(\dot{\Omega})$ (and therefore in $LP(\dot{\Omega})$), removing points that are outside of the feasible region of $LP(\dot{\Omega})$. FBBT also computes bounds for the variables in $PP(\dot{\Omega})$ (and consequently in $LP(\dot{\Omega})$), but using the information from $PP(\dot{\Omega})$ and removing points that lie outside the feasible region of the original problem. The impact these techniques have is therefore on the speed at which the linear relaxations are solved and also on the tightness of the relaxations. Both impacts don't affect the proof of convergence result Theorem 4.3, since all the arguments remain the same and the only change is to the tree that is generated when using the algorithm.

Nonetheless, the employment of bound tightening techniques does have a more relevant place in the algorithm `unboundedRLT` compared to the algorithm `plainRLT`. Recall that in `plainRLT` we start with a bounded problem, $PP(\Omega)$, and therefore we already have bounds for the variables and $LP(\Omega)$ has a finite optimum if feasible. In `unboundedRLT` we start with a possibly unbounded problem $PP(\dot{\Omega})$ and the linear relaxation $LP(\dot{\Omega})$ may not have a finite optimum. If that was the case, we wouldn't be able to continue with the algorithm. But, if somehow we could infer new bounds for the variables using the information from $PP(\dot{\Omega})$ and those new bounds excluded the points that lead to the infinite improvement of the objective function from the feasible region of $LP(\dot{\Omega})$, we would be able to at least use the branch-and-bound scheme for that problem. This is what we do in RAPOSa, since our implementation

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

of bound tightening employs information from $PP(\dot{\Omega})$ and therefore we may be able to solve more problems by using the bound tightening techniques than otherwise.

4.4.3. Conic-based enhancements. We have the same situation with the conic programming-based cuts discussed briefly in Section 1.6.3. These enhancements add new constraints to problem $LP(\dot{\Omega})$, removing points in the space that are outside the feasible region of $PP(\dot{\Omega})$. As before, applying these enhancements to $PP(\dot{\Omega})$ doesn't affect the convergence and can make the linearization more tight. It may even happen in some cases that adding this new constraints could make the linear relaxation $LP(\dot{\Omega})$ have a finite optimum than otherwise would not.

4.4.4. Branching enhancements. Finally, the other big enhancement is regarding the selection of the branching variable. Since this enhancement only affects to which variable is chosen, and not to the point that is branched upon, the only part of the proof of Theorem 4.3 where this impacts is in Eq. (13). We are changing from branching rule (3) to branching rule (4). Consequently, Eq. (13) could be written now as:

$$\sum_{J\in N_-^\delta} w(p,J)\cdot\left|\bar{X}^k_{J\cup\{p\}}-\bar{x}^k_p\bar{X}^k_J\right| \geq \sum_{J\in N_-^\delta} w(j,J)\cdot\left|\bar{X}^k_{J\cup\{j\}}-\bar{x}^k_j\bar{X}^k_J\right|, \tag{14}$$

for all $j \in N$. But, as the weights of all the criteria defined in Chapter 1 are positive, we can also write:

$$\sum_{J\in N_-^\delta} w(p,J)\cdot\left|\bar{X}^k_{J\cup\{p\}}-\bar{x}^k_p\bar{X}^k_J\right| \geq \left|\bar{X}^k_{J\cup\{j\}}-\bar{x}^k_j\bar{X}^k_J\right|, \tag{15}$$

for all $J \in N_-^\delta$ and for all $j \in N$. Using Eq. (15) and Corollary 1.3, the same argument made in the proof can be used as we will have $X^*_{J\cup\{p\}} = x^*_p X^*_J$ for all $J \in N_-^\delta$ and therefore $\sum_{J\in N_-^\delta} w(p,J)\cdot|\bar{X}^k_{J\cup\{p\}}-\bar{x}^k_p\bar{X}^k_J| \to 0$.

We said that the enhancements to the branching didn't change the branching point, but in Chapter 2 we do in fact introduce an enhancement that changes the selection of the branching point in the algorithm. In this case, not all of the arguments made in the proof of Theorem 4.3 hold, and an adaptation is indeed needed. We don't tackle the modifications as this goes beyond the aim of this chapter.

4.5. Numerical results

In this section we present some preliminary results on the performance of RAPOSa, modified as described in Section 4.2. There is still much work to be done, these results constitute only a first test that the results of the current implementation are consistent with the theoretical results. The instances we have run are taken from the MINLPLIB [20] test set, selecting those polynomial problems that we excluded for the rest of the chapters because of the presence of unbounded variables. Note, that although the theory developed in the chapter does not contemplate having an infinite lower bound, the implementation in RAPOSa is able to handle having infinite lower or upper bounds for some variables. For our numerical analysis we remove all the problems for which RAPOSa does not manage to compute a lower bound when solving the root node. This can respond to two reasons: either the linear relaxation is unbounded or it can't be solved within the

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

time limit. We end up with 87 problems (from a total of 216), all run with a 10 minutes maximum running time.

TABLE 4.1. Performance of `unboundedRLT` and comparison to BARON.

(87 instances)	(87) Unsolved	(37) Gap	(35) Time
`unbounded`	56	1.238	115.586
`unbounded-BT`	46	0.364	27.857
BARON	30	0.028	1.957

In Table 4.1 we can see the preliminary results. We have defined two configurations, `unbounded` and `unbounded-BT`. The first one is running RAPOSa without the use of bound-tightening, and the second one is adding the bound-tightening to the execution. This is done to analyze the impact the bound tightening has on problems without bounds for some of the variables, which is very significant as can be seen in the results. In the last row of the table we have the results of solving those problems with BARON, just to compare how far our implementation is from a state-of-the-art nonlinear solver. Note that we also disable the warm start in RAPOSa.

The performance of RAPOSa is not good compared to BARON's. Even when using bound tightening techniques, we are only able to solve 41 problems, compared to BARON being able to solve 57 of them. Also, the gap obtained is noticeably worse than the one from BARON, and the same happens for the running time. Additionally, BARON is able to find upper and lower bounds for more problems than RAPOSa (as mentioned already, we remove those problems where RAPOSa can't find a lower bound).

There are several reasons for this not good performance, since there is still work to do in the implementation. For example, compared to RAPOSa's default implementation, there is a significant difference, related to the generation of the bound-factor constraints. When all the variables have finite bounds, the bound-factor constraints are generated when the root node relaxation is computed, and then, each time a bound changes (because of the branching step or because of the bound tightening) they are updated by changing only the corresponding constraints to that variable. However, when a variable has an infinite bound, this option of "warm start" for the bound-factor constraints is not available, as the current structure of the code does not allow for it in its current state. This leads to RAPOSa generating the bound-factor constraints entirely each time a bound is changed, which leads to a bigger computational cost associated to this core process in the algorithm. Also, some improvements to the domain reduction phase in the root node can be implemented. For instance, when dealing with an unbounded relaxation, most solvers do some "aggresive probbing", consisting on putting some large lower bound for each of the unbounded variables and using the FBBT to hopefully prove $PP(\dot{\Omega})$'s feasible region is empty with those new bounds, which would allow to bound the relaxation.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Finally, there are still some decisions made in the implementation that can be refined to achieve a better performance of the algorithm. The first one is a decision

on the branching point, both during the hyperrectangle expansion phase and in the regular branching. For the hyperrectangle expansion phase, if an unbounded variable has a finite upper or lower bound, we branch as described in Section 4.2. But, if the variable doesn't have any finite bound, we branch on 0. Also, if the variable has an infinite bound, we cannot employ the `MP` (or any of the others) option from Chapter 2, as we can't compute the middle point of a nonexistent interval. In this case, we branch on the optimal value of the variable in that node, i.e, we default to the `Baseline` option from Chapter 2. The second one is in the interaction with the linear solver. For example, Gurobi has a hard limit on what it considers "infinite" values, which is 10^{30}. Any value with absolute value bigger than 10^{30} is considered infinite. This could change with other auxiliary solvers, which would require an adaptation of RAPOSa's code, but also, the decision on wether the variables are sent to the solver as unbounded or with a hard limit of 10^{30} (or smaller) can be addressed, in the same spirit of the discussion regarding the bounds for the RLT variables in Chapter 3.

4.6. Conclusions

We have introduced an extension of `plainRLT` to deal with variables with infinte upper bounds. We also proved the result guaranteeing the optimality of the result obtained by the algorithm, in case of convergence, and discussed the impact other enhancements could have on this convergence. Lastly, we presented some preliminary numerical results that showcase the implementation works but there is still work to be done to obtain a competitive implementation.

As future work, the refinement of the implementation, for example re-adding the "warm start" generation of the bound-factor constraints or refining the selection of the branching point is a natural next step. Also running the algorithm on more test problems from different test sets such as QPLIB [49]. But, more importantly, an effort must be done to implement some process to bound the linear relaxation for those problems where it's unbounded. Doing this is crucial to be able to solve more problems where the base linear relaxation may not be bounded, but adding some appropriate cuts, for example some conic programming-based ones, or using probing to discard the parts of the feasible region that lead to the unboundedness because they are infeasible in the original problem, leads to a bounded relaxation.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Part 2

Machine Learning for Optimization

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Objectives

Part 2 builds upon what was done in Part 1, since machine learning techniques are used to select between the different enhancements described depending on the particular characteristics of the problem that needs to be solved. The aims of the part are:

- Formally introduce the pace, a performance metric that generalizes time and gap and allows for a simultaneous comparison of solved and unsolved instances, crucial for the use of any learning technique.
- Introduce a learning framework to select between different solvers or configurations of the same solver depending on features computed on the optimization problem at hand.
- Study the performance of the learning framework for choosing between branching rules (configurations) and between different solvers.
- Apply the learning framework to select between the enhancements described in Part 1, to further improve their impact.
- Analyze if incorporating some "dynamic" aspect to the learning can lead to better performance of the framework in the context of a spatial branching algorithm.

Chapter 5 introduces the learning framework and discusses its application to the different scenarios of selecting between configurations and solvers. Chapter 6 studies the extension of the framework to the "dynamic" setting.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 5

General learning framework: Learning on a static setting

Publication derived from this chapter: Bissan Ghaddar et al. "Learning for Spatial Branching: An Algorithm Selection Approach". In: *INFORMS Journal on Computing* 35.5 (2023), pp. 1024–1043. DOI: 10.1287/ijoc.2022.0090

5.1. Introduction

This chapter introduces a learning technique introduced in [54, 58] designed to select between different configurations of an algorithm designed to solve polynomial optimization problems. In [54], the authors introduced the technique to be used for the selection of the branching rule for the branch-and-bound scheme, but it can be applied to any setting in which an `Optimal` selection between different algorithms or different variants or "configurations" of an algorithm is desired.

In recent years there has been a significant increase in the literature devoted to the study of applying learning methods to improve the performance of optimization algorithms [85, 9, 112]. Learning can be applied at different stages of the optimization problem. The most common approach is to use the learning to improve the performance of some decision already being made by the algorithm, exploiting the historical information stored in the "learned" model to (hopefully) make a better decision than otherwise could be made (without a big computational overhead). A natural example, which has been exhaustively studied for MILP problems, is the selection of the branching variable. The most naive way to select the branching variable in this setting is just choosing the variable that is further away from being integer. But if one could branch on all of the variables in the problem, and then check which branching performed the best, one would choose that "`Optimal`" variable to branch on. This is what *strong branching* does, branching on all of the variables and looking at the improvement in the lower bound of the tree. The variable with the largest improvement is chosen. In practice, strong branching has been shown to lead to significantly smaller trees [84], but it has a big computational cost and is therefore highly inefficient in real problems. In this context, the idea of the learning is that there may be some underlying information in the structure of the problem or the branching tree that can be used to predict the variable that will lead to a bigger improvement in the lower bound (and would be chosen by strong branching if it was employed). With a learning scheme, one can decide which variable to choose, based on information from past nodes and their characteristics. In recent years, there has been an explosion of literature devoted to this, see for example [85], but mostly for MILP problems. In particular, in Chapter 6 we explore, in the context of spatial branching, similar ideas to those already studied in the context of integer branching.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

More generally, for the design of optimization algorithms, learning can be used to choose between different configurations of the same algorithm, and even between different algorithms to solve the same class of problems. State of the art solvers have numerous options that can be configured by the user, and learning can be used to predict which configuration is the best for a new problem. Although there is a wide variety of approaches to tackle this kind of learning, in this thesis we build upon the literature on "algorithm portfolios" [56, 82], already explored for MILP problems to select between branching rules [33]. In this chapter we start presenting the analysis in [54], in which we adapt the aforementioned ideas to develop a learning framework for NLP problems and, in particular, spatial branch-and-bound algorithms. Importantly, although in [54] attention is restricted to branching rule selection, we push this approach a bit further and show it can be effective at learning other decisions or configurations of an algorithm such as the type of bound-tightening to apply or the selection of the branching point and, finally we show that it can even be applied to the selection between different solvers.

Up until now, we have not discussed the meaning of "static" as mentioned in the title of the chapter (and neither the meaning of "dynamic" in Chapter 6). When it comes to the comparison of learning methodologies, one can establish different taxonomies. An important distinction is between *offline* learning and *online* learning. Offline learning requires that the training is separated from the solution of a new optimization problem: the (possibly) computationally demanding training phase takes place upfront, and there is no computational overhead at the moment of execution of the optimization algorithm. On the other hand, online learning requires gathering data during the execution ot the algorithm, for example gathering data node by node in a branch-and-bound based algorithm, and the learning is dynamically updated as new data becomes available, but there is a trade-off between the potential improvements in performance and the computational cost of learning, which is now part of the cost of execution. As far as features are concerned, one can distinguish between *static* and *dynamic* features. Static features only depend on the specific characteristics of each optimization problem, such as the number of variables, the number of constraints, some descriptive statistics on the problem coefficients, or the sparsity of the problem. Dynamic features, on the other hand, can be much richer, capturing information about the evolution of the branch-and-bound algorithm such as the progress in the bounds of the problem, bounds on the variables, and even properties of the branch-and-bound tree itself.

In this chapter we deal with offline learning: i) we have a set of configurations whose performance we want to predict; ii) there is a set of available problems; iii) all these problems are solved with all configurations; and iv) the resulting performance metrics are taken as the input of the learning methodology. Once the learning is done, the resulting model is used before running the algorithm to select the best configuration for the particular problem at hand. Therefore, in this chapter, we only use static features, since we don't incorporate information obtained on the nodes of the tree but on the problem itself. In Chapter 6 we will study the potential of incorporating dynamic features to the learning in the context of RLT and the selection of the branching variable. In the references mentioned above, specially those on branching variable selection in integer programming, a mixture of static and dynamic features are used, as well as techniques based on offline and online

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

learning. For example, [72] explore an online framework that ranks variables to replicate strong branching.

The structure of the chapter is as follows. In Section 5.2 we described the learning technique introduced in [54, 58], defining the KPI, the explanatory variables and the learning techniques used. In Section 5.3 we describe the application of this learning technique to the selection of the branching rule in the RLT technique, also part of [54, 58]. These two sections form the preliminaries of the chapter, allowing the reader to grasp the potential of the learning framework described as well as its inner structure. In the subsequent sections, we apply the learning technique to the several enhancements described on Part 1 of the thesis. In Section 5.4 we try to choose between the different domain reduction strategies described in Chapter 2; in Section 5.5 we do the same, but for the interaction with the linear solver described in Chapter 3. Finally, in Section 5.6, we apply the learning framework to a different setting: selecting between different commercial nonlinear optimization solvers. The aim of this last section is to showcase the robustness of the learning technique from [54, 58]. Section 5.2 and Section 5.3 are based on the content from [54], also included in [58], a joint work with Bissan Ghaddar, Julio González Díaz, Brais González Rodríguez, Beatriz Pateiro López and Sofía Rodríguez Ballesteros.

5.2. Proposed Methodology

In this preliminary section, we described the methodology introduced in [54, 58]. We describe the KPI used for the learning, the features employed as explanatory variables and lastly the learning technique itself. We remind the reader that we are in a setting of offline learning and using static features, that is, features computed on the instance.

5.2.1. 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", and
- $C =$ "Instances not solved by any algorithm 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 differentiating between algorithms for all instances 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 [2] 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

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

existing one is removed. In [63], when discussing limitations of their work stemming from the 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". Other recent papers [127, 62] use theoretical methods based on survival analysis to define a KPI for selecting the most suitable algorithm for a given problem. The proposed methods take into account the running time as well as the times when the algorithm fails to complete within the time limit (i.e., timeouts).

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 all branching rules. 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 all of the rules. 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 all of the rules. 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 than those that have not, since they will have both a smaller computational time and a larger reduction in the `Optimality` gap.

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:

$$\text{pace}^{LB} = \frac{\text{time}}{LB^{\texttt{end}} - LB^{\texttt{root}} + \varepsilon}. \tag{16}$$

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}, will guide both

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the learning process described in Section 5.2.3 and the illustration of the results in the rest of the chapter.

5.2.2. 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 configuration of the algorithm. Following some of the ideas in [33], [72], and [2] for MILP and our knowledge of $PP(\Omega)$ problems, we have considered 34 features. They are `sum`marized in Table 5.1. Features are categorized depending on whether they represent characteristics of 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 1.5.1. Many of the chosen features consist on counts or statistical measures (like 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.

TABLE 5.1. List of features used by the different learning techniques.

Variables	No. of variables, variance of the density of the variables[1] Average/median/variance of the `range`s of the variables[2] Average/variance of the no. of appearances of each variable[3] Pct. of variables not present in any monomial with degree greater than one Pct. of variables not present in any monomial with degree greater than two
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 objective function[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 an number of variables might have.
[6] We do not include transitivity for CMIG since, given its structure, its value is always 0.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

5.2.3. Quantile Regression Learning Techniques. In supervised learning, data consists of input–response pairs. Given an optimization problem of the form $PP(\Omega)$ we use, as input, the information from the features vector. Regarding the response, we compute for each of the configurations of the algorithm that are to be compared 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 normalized pace as npace^{LB}. Thus, the closer this value is to one for a given configuration, the better the performance of it. Note that the pace^{LB} of a given configuration is independent of the other configurations, but its npace^{LB} is not.

While our ultimate goal is the selection of the best configuration, our variables of interest are quantitative responses. Therefore, we address the problem as a regression problem. The task is to learn, for each configuration, a real-valued function that models and predicts npace^{LB}. Then, given an instance, the configuration 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 [54], they motivate the use of quantile regression as an asymmetric behavior of y (negative skewness), as well as the presence of outliers, was observed. 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, $Q_\tau(\boldsymbol{x}) = \inf\{y \colon F(y|\boldsymbol{x}) \geq \tau\}$, 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 [74] and [75] for a detailed overview of quantile regression.

There are several quantile regression methods in the statistics and ML literature. Just as classical regression methods use the fact that $\mu(\boldsymbol{x})$ minimizes the expectation of the quadratic loss function, quantile regression methods are based on the fact that $Q_\tau(\boldsymbol{x})$ minimizes the expectation of an assymmetric loss function. More specifically, $Q_\tau(\boldsymbol{x}) = \arg\min_{q \in \mathbb{R}} E(\rho_\tau(y - q)|\boldsymbol{x})$, being $\rho_\tau(m) = m(\tau - \mathbb{I}\{m < 0\})$ the so-called pinball loss function and $\mathbb{I}$ is an indicator function. Also, as with classical regression, quantile regression methods `range` from linear to nonlinear and from parametric to nonparametric, depending on the assumptions. 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 $\mu(\boldsymbol{x}) = texttttsum_{j=1}^{m} f_j(\boldsymbol{x})$, where the m additive terms represent predictors. The f_j's are nonparametric functions that can be modeled using an expansion of basis functions; spline basis functions in our case. This leaves behind the traditional linear model that fails to represent the real effects of the inputs, which are normally nonlinear.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

For detailed information on generalized additive models we refer the reader to [66]. Quantile generalized additive models extend the previous idea to $Q_\tau(\boldsymbol{x})$ [39].

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 [46]. The stochastic gradient boosting algorithm is a modification of the gradient boosting algorithm by [45], 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 [76].

Quantile regression forests. Random forests are supervised learning algorithms that can be used both in classification and regression problems. They were introduced in [15] 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 [90], 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 other ensemble methods that use bagging, is that we can use the complete dataset of size n to evaluate the model using the out-of-bag predictions, without randomly splitting the data into training and test set. To do so, a large number m of bootstrap samples of size n are obtained from the dataset by sampling with replacement. Each bootstrap sample leaves out, on average, about 37% of the observations (the left-out observations constitute the so-called out-of-bag sample).[1] Then, m individual trees are grown using these m bootstrap samples. For each observation in the dataset, the corresponding out-of-bag prediction is obtained by taking into account only those trees fitted on bootstrap samples that leave out that particular observation.

5.3. Choosing a branching rule

In this second preliminary section, we include the use case of the learning framework in Section 5.2 described in [54, 58], aimed at selecting between different branching rules implemented in RAPOSa. The objective is to be able to select the best branching rule for a given instance using the learning, as there are significant differences in performance between the branching rules, as can be seen in Figure 5.1. This serves as a showcase of the performance of the learning framework.

5.3.1. Branching Rules. We use the rules defined in Section 1.5, in particular, the rules `max`, `sum`, `dual`, `range`, `eig-VI` and `eig-CMI`. Figure 5.1 shows the results of a preliminary computational experiment to assess the performance of the different branching rules on $PP(\Omega)$ problems. More precisely, we used three different sets of instances (refer to Section 1.9 for details): DS [30], MINLPlib [20], and

[1] By sampling with replacement, a given observation of the dataset of size n does not appear in the bootstrap sample with probability $(1 - 1/n)^n$ (which tends to $1/e \approx 0.3678$).

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

QPLIB [49]. For our comparison of the branching rules we use KPI npace^{LB}. The closer the normalized KPI is to 1 for a given branching rule, the better the performance of that rule. Figure 5.1a 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 MINLP instances. These observations already hint at the promising potential of the integration of learning techniques in the branching rule selection process. In Figure 5.1b we rank, for each problem, the branching rules from 1 (best) to 6 (worst), according to the normalized KPI. The stacked bar graphs represent the percentage of problems in each rank position per branching rule and the numbers inside the bars indicate the average value of the normalized KPI for the problems in each rank position. Note that it is important not only to choose the best possible branching rule, but also to ensure that, in case of not doing so, the selected rule still performs reasonably well. For instance, for those MINLPLib instances where the `max` rule is not the best one, its performance is on average significantly worse than other methods, ranging from 0.21 to 0.65. In those cases it would be preferable to choose, for instance, the `range` rule, even when it is not the best possible choice, since its performance then `range`s from 0.53 to 0.91.

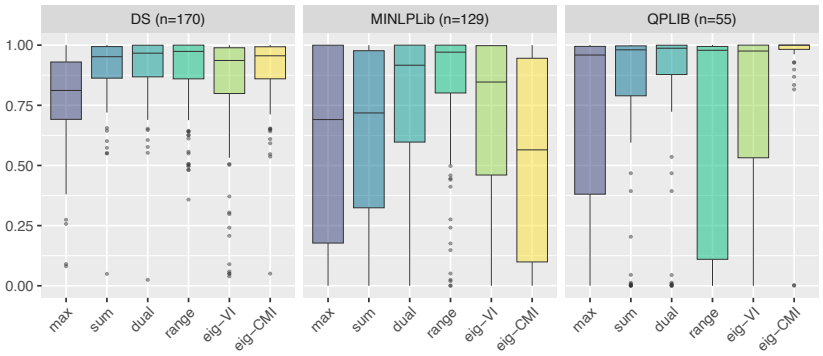
In addition to the above branching rules, in Section 5.3.4 we also study the impact in our learning framework of adding three new branching rules based on adaptations of reliability branching and violation transfer to our setting, as explained in Section 1.5.

5.3.2. Experimental Setup. As described in Section 1.9, we used the three sets of instances DS, MINLPLIB and QPLIB to test the performance of the learning approach on a variety of instances with different characteristics. In this case, we have disregarded instances solved at the root node, since the branching rule plays no role in them. We also discarded unconstrained problems, as some of the features have the number of constraints in the denominator. Our final dataset consists of a total of 354 instances. Interestingly, although the size of our dataset is relatively small for current ML standards, the improvements reported in Section 5.3.3 and Section 5.3.4 show the potential of our methodology.

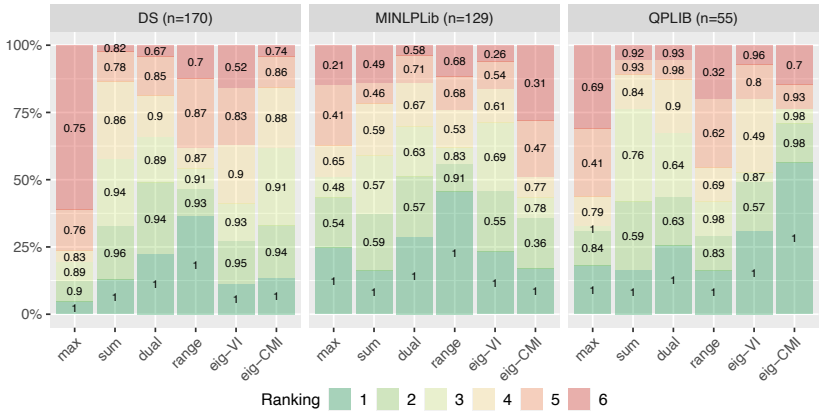
The branching rules defined in Section 5.3.1 have been coded in RAPOSa [59], a state-of-the-art implementation of the RLT outlined in Section 1.8. The analyses are performed using a configuration of RAPOSa with a minimal number of enhancements: calls to an NLP local solver and J-sets [31]; refer to Section 1.8 or [59] for details. Each instance of our final dataset was run with six different configurations of RAPOSa, based on the six different branching rules defined in Section 5.3.1. The time limit of each execution was set to one hour.

Learning is performed jointly on all sets of instances following the standard learning procedure. We randomly split the complete set of instances into two disjoint sets: the training set (70%) and the test 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

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of the normalized KPI for each of the branching rules.



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

FIGURE 5.1. Comparative analysis of the performance of the different branching rules.

is maintained through the splitting process. Moreover, in order to gain robustness, we construct 10 partitions of the dataset into training and test data and report aggregate results over all the partitions. It is worth noting that different variations of these experiments were also carried out when preparing the manuscript such as different time limits and variations in the sets of instances. The findings in all these experiments were qualitatively very similar to the ones we report.

As discussed in Section 5.2.3, when performing quantile regression one has to specify the τ-conditional quantile to be estimated. Given $\tau \in (0,1)$, let $\hat{Q}_\tau(\boldsymbol{x})$ denote the estimated τ-conditional quantile of npace^{LB} for a given branching rule. This means that $(1-\tau)100\%$ of the time the performance of that rule is expected to be larger than $\hat{Q}_\tau(\boldsymbol{x})$. Due to the negative asymmetry present in the KPI (see Figure 5.1), we have opted for values below the median ($\tau \leq 0.5$), where more differentiated behaviors can be observed between the branching rules and where it is more relevant to make a good selection. Moreover, if we fix a large value of τ we would select a branching rule that is estimated to have a very good performance but with a small probability. Our goal is to select for each instance a branching

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

rule that, with high probability, achieves a reasonable performance. Although the different values evaluated ($\tau = 0.3$, $\tau = 0.4$ and $\tau = 0.5$) led to similar results, the best ones were obtained with $\tau = 0.3$, which is therefore the value used for the numerical analysis.

5.3.3. Main 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.

5.3.3.1. *Assessing the different learning methodologies.* As a first step we compare the three ML-based rules obtained with the learning methodologies described in Section 5.2.3, that is, quantile generalized additive models (Q-GAM), stochastic gradient boosting for quantile regression (SGB-QR), and quantile regression forests (Q-RF). In order to get a more thorough comparison, we study the ML-based branching for three configurations of RAPOSa: first a baseline configuration, then a configuration with warm start (WS) of the LP solver and, finally, a configuration with warm start and bound tightening (WS + BT); refer to Chapter 1 or [59] for details on these enhancements.

Table 5.2 presents a first comparison, based on the performance of the resulting ML-based branching rules at solving the optimization problems in the test sets. The reported numbers correspond to the averages, across the 10 test sets, of the geometric means of pace^{LB} in the instances of each test set.[2] We use `Best` to refer to the original branching rule where the aforementioned average is smaller, that is, the one performing best. For the Baseline configuration, `Best` corresponds with `range`, while for the other two it corresponds with `dual`. `Optimal` is an instance-specific oracle, i.e., it chooses, 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 with respect to pace^{LB}, although Q-RF performs slightly better.

TABLE 5.2. ML-based rules' performance with respect to pace^{LB} (average over test sets).

	Baseline	**WS**	**WS + BT**
`Best` (across all instances)	1.875	1.534	0.819
ML-based branching rule:			
Q-GAM	1.439	1.181	0.782
SGB-QR	1.439	1.174	0.763
Q-RF	1.415	1.153	0.745
`Optimal` (instance-specific oracle)	1.187	0.956	0.653
Improvement after learning (with Q-RF)	-24.5%	-24.8%	-9.0%
`Optimal` **improvement** (upper bound learning)	-36.7%	-37.6%	-20.2%

We also compare the learning approaches in a more principled statistical manner. Recall that, for each learning approach, the ML-based branching rule of a given instance is the one that corresponds to the highest predicted τ-conditional quantile of the normalized KPI. Thus, we can evaluate the predictive performance of the three methods by measuring the accuracy on the predicted quantiles. We

[2]We take $\varepsilon = 0.001$ in Eq. (16).

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

do so by using the prediction error measurement for quantile regression methods introduced in [130]. More specifically, for each learning approach, we have computed the mean absolute prediction error for each branching rule over the ten test subsamples. The overall average errors of Q-GAM, SGB-QR, and Q-RF are 1.53, 1.16, and 0.95, respectively. Since Q-RF is also the best from this point of view, hereafter we use it as the reference learning method.

5.3.3.2. *Assessing the impact of the learning.* Going back to Table 5.2, we can see that, for the Baseline configuration, Q-RF improves the pace by 24.5% when compared to `Best`. Importantly, this is not far from the improvement obtained by `Optimal`, 36.7%, which is an upper bound for our algorithm selection approach. For WS + BT, the improvement with Q-RF is 9% and the `Optimal` one is 20.2%. 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, WS enhancement improves the KPI from 1.875 to 1.534, which is 18.2%, falling short of the 24.5% improvement obtained using Q-RF in the baseline model. This implies that, given the choice between implementing a warm start enhancement or a machine-learning branching approach, the latter might be preferable (and with no computational overhead). Table 5.3 shows that, when evaluating the performance 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 dataset and `Best` was always `dual`).

TABLE 5.3. ML-based rule's performance with respect to pace^{LB} (Out-of-bag).

	Baseline	**WS**	**WS + BT**
`Best` (across all instances) – `dual`	1.736	1.399	0.733
Out-of-bag Q-RF	1.191	1.050	0.678
`Optimal` (instance-specific oracle)	0.983	0.810	0.601
Improvement after learning (with Q-RF)	-31.4%	-24.9%	-7.5%
`Optimal` **improvement** (upper bound learning)	-43.4%	-42.1%	–18.0%

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.[3]

In Figure 5.2, 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 the average value of npace^{LB} for the problems in each rank position. Note that Q-RF always chooses one of the six original rules and, hence, for its ranking in a given instance we just take the ranking of the branching rule it chooses for that instance. 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 all the other six branching rules (ranging from 0.43 to 0.69). In particular, the plots

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

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

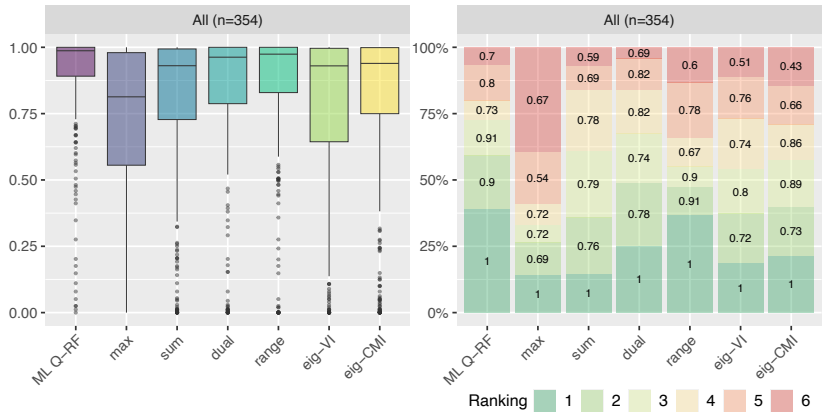


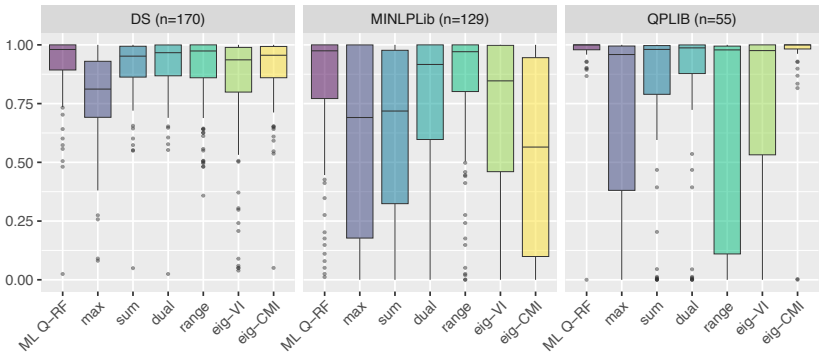
FIGURE 5.2. Performance in Baseline configuration. Left, npace^{LB} boxplots of all the branching rules. Right, branching rules ranked from 1 (best) to 6 (worst), according to npace^{LB}.

in Figure 5.2 show that, although the learned criterion 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 5.2 and 5.3, is close to the `Optimal` improvement.

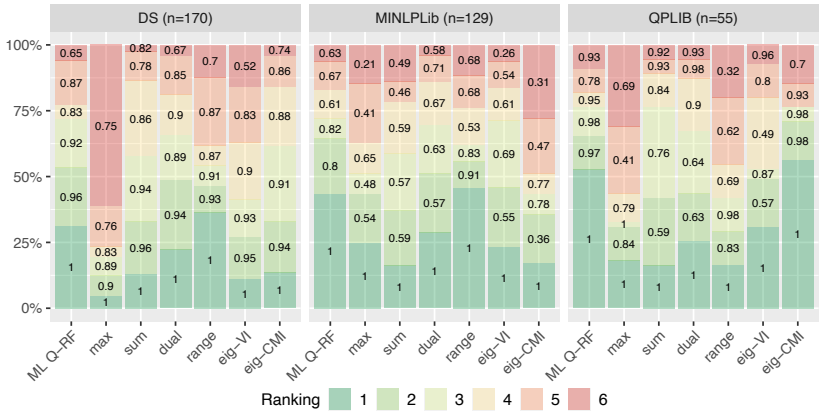
In Figure 5.3 we separate the results in Figure 5.2 for the three sets of instances, DS, MINLPLib, and QPLIB. The number of instances within each library is indicated in parenthesis. There are some differences across datasets. For instance, the box plots show little variation in npace^{LB} across the branching rules for DS set compared to the other two sets. The `max` rule is the worst one, while the other rules are close in terms of normalized pace. For MINLPLib, `range` performs slightly better than Q-RF as the number of times that `range` ranked first is 45.73% compared to 43.41% for Q-RF. For QPLIB, both Q-RF and `eig-CMI` strongly dominate in terms of performance, ranking first in 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 behave well in all three sets of instances, which suggests that it is not overfitting to a particular one.

Next, we further explore to what extent the ML-based rule manages to learn the patterns behind `Optimal`. Figure 5.4 presents a comparison of the frequencies with which the original rules are chosen by the ML-based rule and `Optimal`. The behavior of the ML-based branching rule strongly resembles that of `Optimal` in all three datasets. 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` rule 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 `Optimal`. Interestingly, in this last set of instances, the `range` is rarely selected. Overall, these findings are consistent with the results presented in Figure 5.3. Recall that our offline learning is performed jointly on all sets of instances, so the results in

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of the normalized pace using the described branching rules.



(B) Branching rules ranked from 1 (best) to 6 (worst), according to the normalized pace.

FIGURE 5.3. Performance in Baseline configuration. Comparison across the sets of instances.

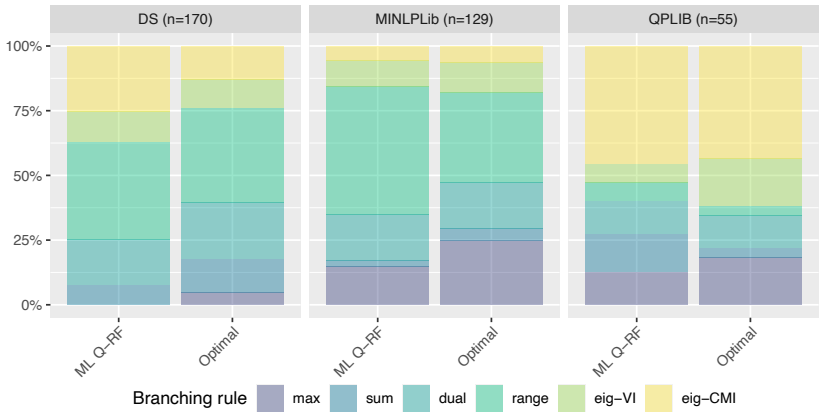


FIGURE 5.4. Stacked bar graphs represent percentage of times that each branching rule is selected by Q-RF and `Optimal`.

Figure 5.4, showing that the ML-based rule matches the patterns of `Optimal` in 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 hetereogeneous instances.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

5.3.3.3. *Interpretability.* One advantage of the 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 5.5 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 8: CMIG.mod, VIG.mod, and CMIG.Dens. Further, CMIG.mod, the modularity on CMIG, is in second 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: CMIG.mod and CMIG.Dens are particularly important for `eig-CMI` and VIG.mod score is very high for `eig-VI`. Additionally, the most important features for learning `range` and `dual` branching rules are those related to the `range`s of variables.

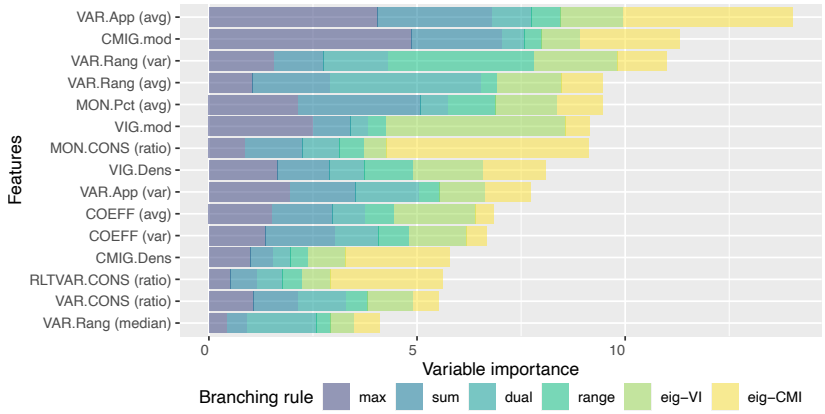


FIGURE 5.5. Stacked bar graphs represent, for each branching rule, variable importance score of the features obtained using Q-RF.

The highest scores accross the individual regressions are obtained by variables MON.CONS and Var.App for `eig-CMI`, VIG.mod for `eig-VI`, and CMIG.mod for `max`. Whereas VIG.mod's score for `eig-VI` might be expected, the mechanisms behind the others would require further investigation, and in particular, CMIG.mod's score for `max`, which might not be expected since `max` is not a graph-based branching rule. Although a deep analysis along these lines is definitely worth pursuing, it is beyond the scope of this thesis.

5.3.3.4. *Performance profiles.* To conclude, we compare the performance of the different branching rules by showing the corresponding performance profiles [35]. In Figure 5.6 we present the performance profiles for the three sets of instances together and individually. In the x-axis we represent ratios of the pace, 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 the pace of each configuration by the pace of the best configuration in that instance.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Consistently with all the results we have already reported so far, the ML-based rule Q-RF performs noticeably better than all the original branching rules. From the DS library plot, Q-RF performs slightly better than 4 of the 6 branching rules (while `max` and `eig-VI` are significantly worse). For MINLPLib library, Q-RF is clearly the best and `range` is in second position. For QPLIB, 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, Q-RF manages to slightly outperform the best rules in each individual set and solidly outperforms all the original rules overall.

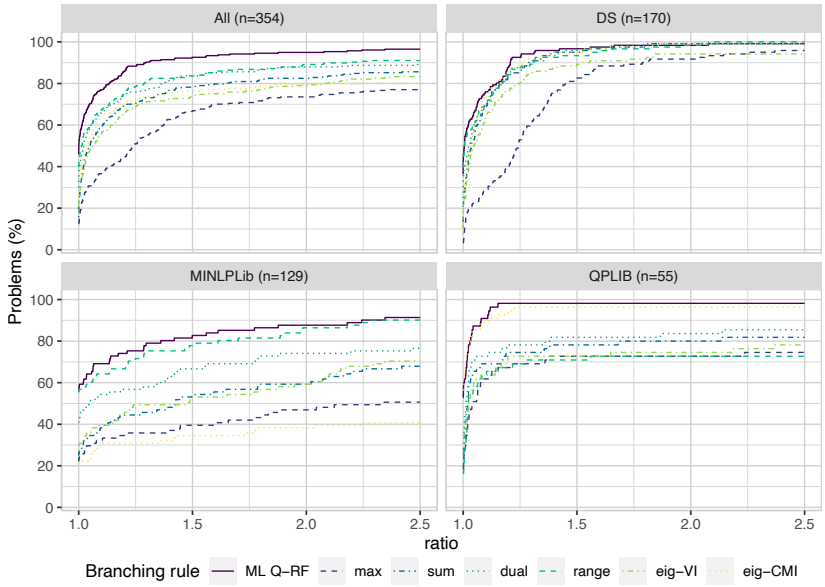


FIGURE 5.6. Performance profiles using the pace KPI for different sets of instances.

5.3.4. Enhancing the Portfolio of Branching Rules. By definition, the larger the portfolio of branching rules is, the better the performance of `Optimal` will be, i.e., the larger the potential for improvement with the learning method we have developed. Similarly, for a given number of rules in the portfolio, one might also expect that the more diverse they are, the larger the potential for improvement should be. In order to assess these two assertions, we incorporate into our analysis three new rules: an adaptation of reliability branching [1], `rb`, and two variants of violation transfer [126], $\texttt{vt}_1$ and $\texttt{vt}_2$. The rule `rb` corresponds to what we called `sum rel.` in Section 1.5, where the violation transfer rules are also introduced. We define two new portfolios of branching rules:

- **Portfolio-6.** Contains six branching rules, as in the previous analysis, obtained by combining the latter three rules with the three best performing ones in the original portfolio: `dual`, `range`, and `eig-VI`. By preserving the size of the portfolio, we can ensure that the differences in performance of the resulting ML-based rule stem from the additional richness of the rules in the portfolio.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

- **Portfolio-9.** A portfolio of nine branching rules, in which the three new rules are added to the original portfolio. This allows to study the joint impact of size and richness of the portfolio.

TABLE 5.4. ML-based rule's performance with respect to pace^{LB} (Out-of-bag) in Portfolio-6.

	Baseline	**WS**	**WS + BT**
`Best` (across all instances) – `dual`	1.736	1.399	0.733
Out-of-bag Q-RF	1.131	0.908	0.630
`Optimal` (instance-specific oracle)	0.900	0.741	0.532
Improvement after learning (with Q-RF)	-34.9%	-35.1%	-14.1%
`Optimal` **improvement** (upper bound learning)	-48.2%	-47.0%	-27.4%

TABLE 5.5. ML-based rule's performance with respect to pace^{LB} (Out-of-bag) in Portfolio-9.

	Baseline	**WS**	**WS + BT**
`Best` (across all instances) – `dual`	1.736	1.399	0.733
Out-of-bag Q-RF	1.089	0.964	0.632
`Optimal` (instance-specific oracle)	0.891	0.733	0.526
Improvement after learning (with Q-RF)	-37.3%	-31.1%	-13.8%
`Optimal` **improvement** (upper bound learning)	-48.7%	-47.6%	-28.2%

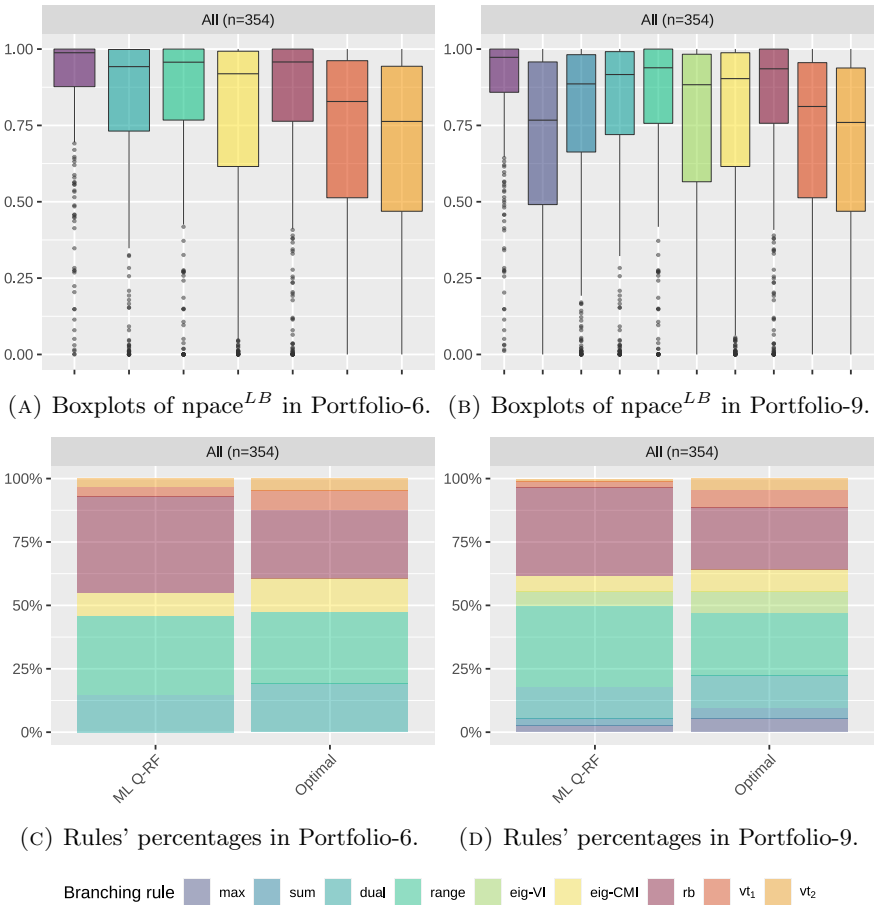
Table 5.4 shows that, in Portfolio-6, the improvement achieved by `Optimal` increases between 4.8% and 9.4% with respect to the old portfolio. Similarly, the improvement of the ML-based rule, Q-RF, increases between 3.5% and 10.2%. This shows that, as expected, our ML-framework benefits from the quality and richness of the underlying portfolio. Then, the results in Table 5.5 show that adding back `max`, `sum`, and `eig-VI` to the portfolio barely improves the performance of `Optimal`, since we just get an additional 0.5%-0.8% with respect to Portfolio 6. Although the Q-RF rules obtained for both Portfolio 6 and Portfolio 9 clearly dominate the one obtained in Section 5.3.3, the comparison between them is not so clear. The rule obtained for Portfolio 9 is 2.4% better in Baseline, but 4% and 0.3% worse in WS and WS+BT, respectively. Thus, although the results for Portfolio-6 show that the quality of the rules in the portfolio should definitely have a positive impact in the performance of the ML-based rule, the results for Portfolio-9 suggest that adding poor rules to the portfolio might even have a detrimental effect on its performance.

The plots in Figure 5.7 provide some additional insights. Clearly, among the new branching rules, the main impact comes from `rb`, which is competitive with `dual` and `range`: it is the best rule in around 25% of the instances in both portfolios and it is chosen by ML-based Q-RF slightly under 40%. On the other hand, $\texttt{vt}_1$ and $\texttt{vt}_2$ combined are the best in around 12% of the instances and are chosen by ML-based Q-RF in just around 5%.

5.4. Choosing a domain reduction strategy

In this section we take the enhancements of Chapter 2 one step further. The goal is to study the potential for additional performance improvements by using machine learning techniques to predict the best combination of enhancements for a

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of npace^{LB} in Portfolio-6. (B) Boxplots of npace^{LB} in Portfolio-9.

(C) Rules' percentages in Portfolio-6. (D) Rules' percentages in Portfolio-9.

FIGURE 5.7. Performance of the ML-based rules in Portfolio-6 and Portfolio-9.

given instance, and in particular the learning framework described in Section 5.2. To this end, we also analyze the improvement on performance we would observe if we had an oracle signalling the best combination for each instance and, then, we use it as a reference to assess the quality of the learning.

Following recent trends in the field, we have built upon the learning framework developed in [54] to show that machine/statistical learning can be used to further improve performance. The promising results obtained in Section 5.4.3, with the direct application of the methodology in [54], suggests that one could get even better results with tailor-made modifications of it. In particular, a relatively simple avenue for future research would be to extend the set of features used for the learning, including additional ones that may be suitable for domain reduction.

5.4.1. Machine learning framework. We use the machine learning framework described in Section 5.2. In particular, for our analysis we follow the best performing approach among those reported in [54], choosing pace as the measure of performance (KPI) and using quantile regression forests for the learning [90]. The learning is carried out on the same set of instances used in Chapter 2, using the statistical language R [106] with the library `ranger` [133].

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

It is important to note that, not only we follow the approach in [54], but we also use the same set of features and the same parameters for quantile regression (as described in Section 5.2), so there is no tailor-made adaptation to the current setting. Thus, our results can be seen as a lower bound on what might be achieved by considering additional features that capture relevant aspects for domain reduction.

In the results' tables in this section we report the performance of a special approach: `Optimal`, which represents the performance obtained when we choose, for each instance, the best configuration according to pace, i.e., it's an ideal version representing what we would get if we had an oracle signalling the best approach for each instance. If, for a given instance, there are two options with the same pace, we choose the one with the smallest `Optimal`ity gap.

5.4.2. Impact of machine learning on the individual enhancements. Table 5.6 `sum`marizes the results of our learning methodology when applied individually to each one of the three different enhancements discussed in Chapter 2. More precisely, for each enhancement, the goal is to predict which one of the considered approaches will have a better performance on a given instance. To simplify the tables of results, among the original approaches we just represent the best performing one for the given enhancement, `Best`. "out-of-bag Q-RF" is the result of choosing, for each instance, the approach selected by the machine learning framework. Finally, "Improvement after learning" is the percentage of relative improvement from `Best` to out-of-bag Q-RF and "`Optimal` Improvement" is the same but comparing `Best` to `Optimal`.

The results in Table 5.6 are somewhat mixed. For the conic bound tightening from Section 2.2, we see modest improvements in gap and time, but relatively far from the ones delivered by `Optimal`. Learning seems to perform particularly well for the FBBT enhancements from Section 2.3, with out-of-bag Q-RF significantly outperforming `Best` and getting around half-way of the improvements `Optimal` would achieve. Finally, regarding the selection of the branching point from Section 2.4, the learning does not seem to obtain much since, although it improves almost by 2% in time with respect to `Best`, its performance deteriorates in gap and pace by 0.4% and 1.19%, respectively. It seems as if the inherent randomness behind the impact of the branching point on the generated branch-and-bound trees is hard to learn upon.

5.4.3. Impact of machine learning on the combined enhancements. Despite the mixed results of the previous section, one would expect that, when adding all of the enhancements and its combinations together, there is more potential for learning, given the additional richness and diversity of the underlying approaches/configurations. For the sake of exposition, learning is not carried out on all approaches and combinations, but just on those reported in Section 2.5.

The results in Table 5.7 show that machine learning performs fairly well. The learning approach outperforms `Best` ($\texttt{FBBT}^{OB+NB}$ + `0.5·OV+0.5·MP`) according to all the metrics: it solves three more instances and obtains improvements of around 8% and 3% in time and gap. As expected, having a richer set of configurations to choose from also helps `Optimal` to obtain larger improvements.

Figure 5.8 represents, side by side, how often each of the eight configurations is selected by out-of-bag Q-RF and by `Optimal`. We can see that out-of-bag Q-RF mimics quite well the behavior of `Optimal`. Importantly, Figure 5.8 also shows

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 5.6. Machine Learning impact on the different enhancements.

Section 2.2: OBBT enhancements based on conic optimization				
(403 instances)	(403) Solved	(111) Gap	(153) Time	(268) Pace
`Best` (`Baseline`)	286	0.132	56.95	6.56
Out-of-bag Q-RF	286	0.131	56.75	6.57
`Optimal`	287	0.128	53.68	6.19
Improvement after learning	0.0%	−1.13%	−0.35%	0.01%
`Optimal` improvement	0.35%	−3.42%	−5.74%	−5.62%
Section 2.3: FBBT enhancements based on duality				
(403 instances)	(403) Solved	(111) Gap	(134) Time	(248) Pace
`Best` ($\texttt{FBBT}^{OB+NB}$)	287	0.128	98.32	8.08
Out-of-bag Q-RF	288	0.125	94.72	7.90
`Optimal`	288	0.124	89.65	7.65
Improvement after learning	0.35%	−1.86%	−3.66%	−2.22%
`Optimal` improvement	0.35%	−2.52%	−8.81%	−5.33%
Section 2.4: Impact of the branching point on the branch-and-bound tree				
(403 instances)	(403) Solved	(110) Gap	(138) Time	(250) Pace
`Best` (`0.5·OV+0.5·MP`)	288	0.108	96.75	7.64
Out-of-bag Q-RF	288	0.108	94.85	7.73
`Optimal`	290	0.104	82.66	6.95
Improvement after learning	0.0%	0.4%	−1.96%	1.19%
`Optimal` improvement	0.69%	−3.6%	−14.56%	−9.14%

TABLE 5.7. Machine Learning impact on all of the enhancements together.

(403 instances)	(403) Solved	(112) Gap	(146) Time	(257) Pace
`Best` ($\texttt{FBBT}^{OB+NB}$ + `0.5·OV+0.5·MP`)	287	0.104	81.73	7.11
Out-of-bag Q-RF	290	0.104	75.21	6.89
`Optimal`	291	0.099	66.13	6.28
Improvement after learning	1.05%	−0.32%	−7.97%	−3.13%
`Optimal` improvement	1.39%	−4.41%	−19.09%	−11.71%

that a configuration with $\texttt{SOCP}^{M}$ is chosen in around one third of the instances, demonstrating the potential of using conic-based relaxations in the OBBT stages of global optimization solvers.

Despite the apparently nice behavior of out-of-bag Q-RF displayed in Figure 5.8, it might even be the case that it never chose the best configuration and

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

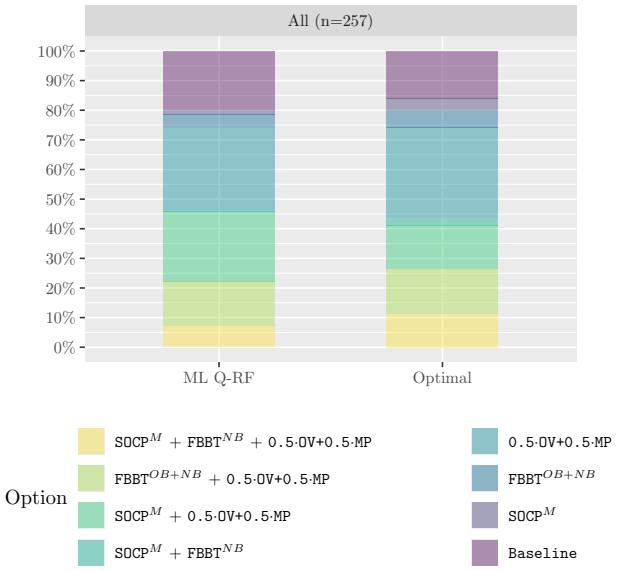


FIGURE 5.8. Percentages in which each version is selected by out-of-bag Q-RF and by `Optimal`.

just happened to use the different configurations with the right frequencies. Figure 5.9 presents detailed information regarding the performance of the different configurations which, in particular, allows to assess how often each of them is the best one. The bar chart represents, for each configuration, the percentage of instances in which that configuration was the best one, the second best, and so on, for the eight ranking positions it might occupy depending on its performance relative to the rest.[4] Moreover, the numbers inside the bars indicate how close that configuration is, on average, to the best one for the instances in which it occupies the corresponding ranking position. More specifically, these numbers go from zero to one and are computed by dividing the best (smallest) pace among the different configurations by the pace of the current one, and then taking the average over the instances in each ranking position. Thus, Figure 5.9 allows to assess not only how often each approach is in each ranking position, but also how rapidly its performance deteriorates as it goes from the top positions to the bottom ones.

Figure 5.9 shows that out-of-bag Q-RF comes on top more frequently than any other configuration, with `0.5·OV+0.5·MP` close behind. Yet, the main difference between them comes from the fact that the relative performance of out-of-bag Q-RF deteriorates more slowly as it moves from being the best configuration to the second or third best, for example. In Figure 5.9 we can also see that the configurations that use `SOCP`^{M} are the ones in which performance tends to deteriorate more quickly, which is consistent with the discussion in Section 2.2 about their "riskiness". In this respect, since out-of-bag Q-RF uses `SOCP`^{M} in nearly one third of the instances, the fact that it performs so well even when it does not choose the top performing configuration suggests that it is successful at learning to avoid using `SOCP`^{M} in instances in which it does not perform well. Yet, the difference in performance

[4]Note that we have 8 configurations plus out-of-bag Q-RF, and we just consider ranking positions from 1 to 8. This is because out-of-bag Q-RF always coincides with one of the eight configurations on which the learning is performed.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

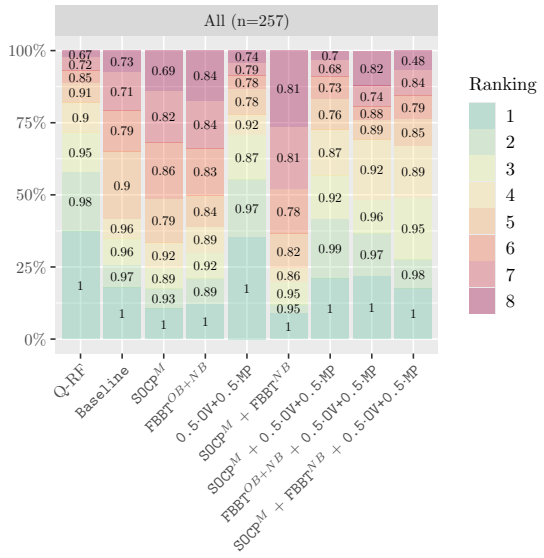


FIGURE 5.9. Domain reduction enhancements ranked from 1 (best) to 8 (worst), according to pace.

between out-of-bag Q-RF and `Optimal` also suggests that there is room for further improvement. A natural direction would be to enhance the set of features of the instances, with the goal of capturing additional aspects that might be relevant for domain reduction.

5.5. Anticipating the impact of domain reduction on the linear solver

Given that we have not been able to obtain any clear pattern in the computational experiments in Chapter 3, a natural approach is to use statistical learning techniques with the goal of anticipating, on a given instance, which of the two approaches, `RLT-looseB` or `RLT-tightB`, will perform better. To that end, we employ the ML framework described in Section 5.2 of this chapter. We perform the analysis on both the results with bound tightening and the results without bound tightening, for solver Gurobi. The learning is done together for both test sets, DS and MINLPLIB, using the quantile regression forests as described in Section 5.2, but in the tables we separate the results by test set. The results displayed were obtained using the out-of-bag predictions from the learned method.

TABLE 5.8. ML without bound tightening.

	Instances		`RLT-looseB`		`RLT-tightB`		ML		`Optimal`	
	MINLP	DS	MINLP	DS	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	165	179	52	14	**-3.9**	21.4	**-1.9**	0.0	**-3.9**	0.0
Gap	50	17	0.6	0.0	**-21.8**	45.9	**-15.5**	0.0	**-24.3**	0.0
Time	46	103	54.6	83.4	**-4.7**	51.3	**-5.7**	0.0	**-12.9**	**-0.1**
Pace	96	117	183.3	0.1	6.2	44.2	**-6.1**	0.0	**-11.5**	**-0.1**
Nodes	113	162	73.0	163.4	**-1.2**	**-0.4**	**-1.5**	0.0	**-3.9**	**-0.0**
L-Time	113	162	0.0	0.0	455.0	126.4	1.5	0.0	3.0	0.2

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 5.9. ML selections without bound tightening.

	RLT-looseB				RLT-tightB			
	MINLP		DS		MINLP		DS	
ML	128	78.0%	179	100.0%	37	22.0%	0	0.0%
Optimal	123	75.0%	175	98.0%	42	25.0%	4	2.0%

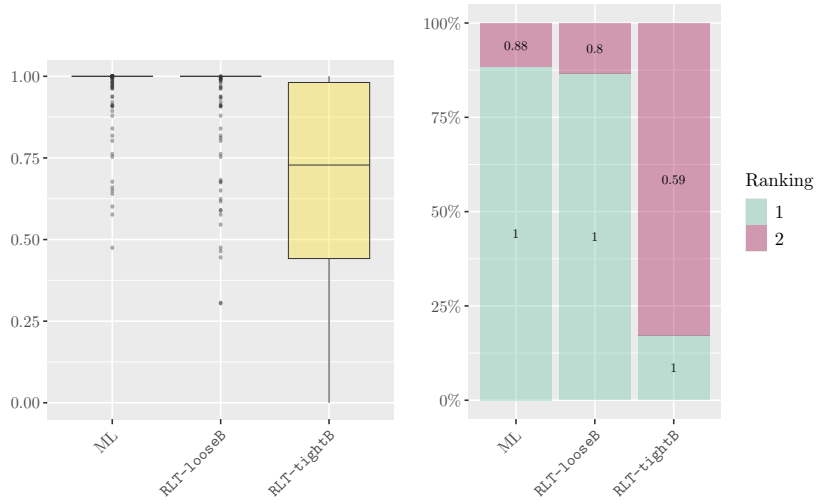
On Table 5.8 we can see the performance of ML on the results without bound tightening. After the "Instances" columns, on the following two columns, we have the number of unsolved problems and the geometric means of the gap, time, pace, nodes and linear solving time for the RLT-looseB configuration, and on the remaining columns we have the percentage of change compared to RLT-looseB for the RLT-tightB configuration, for the ML, and the for the Optimal "auxiliary" configuration built by choosing the best performing configuration according to pace for each problem. There is a clear difference in the results between DS and MINLPLIB. Looking at the Optimal column, it's clear that choosing RLT-tightB does not improve the performance for almost any problem in the DS library, i.e., for those problems the margin for learning is nonexistent. Nonetheless, the ML learning technique is able to select practically always the "correct" option as we see in Table 5.9 where the amount of times each option is selected is shown for ML and Optimal. For DS, the option RLT-tightB is not chosen for any of the problems in the test set, which speaks of the robustness of the technique. For MINLPLIB, there is a substantial gain when selecting the best option for each problem, as we see in Optimal. We see a 24.3% improvement in gap, a 12.9% improvement in time and a 11.5% improvement in pace. ML is able to capture this correctly, delivering a 15.5% improvement in gap and a 5.7% improvement in time. Furthermore, it does seem to detect when RLT-tightB performs badly in pace and avoid selecting that configuration in those cases, as an improvement of 6.1% is achieved in pace. In Figure 5.10 we can clearly see the superiority of RLT-looseB and how ML is able to capture this and even improve upon that configuration, as we clearly see in Figure 5.10b.

TABLE 5.10. ML with bound tightening.

	Instances		RLT-looseB		RLT-tightB		ML		Optimal	
	MINLP	DS	MINLP	DS	MINLP	DS	MINLP	DS	MINLP	DS
Unsolved	165	174	42	16	2.4	18.8	2.4	0.0	0.0	0.0
Gap	41	18	0.2	0.0	1.6	26.3	2.6	0.0	**-5.3**	**-0.9**
Time	50	102	48.7	105.8	5.8	8.1	3.9	**-0.1**	**-9.5**	**-1.5**
Pace	92	117	79.4	0.2	**-1.3**	7.2	**-2.1**	**-0.1**	**-10.7**	**-1.3**
Nodes	122	155	45.7	120.5	**-2.3**	**-0.1**	**-2.2**	**-0.2**	**-2.7**	**-0.1**
L-Time	122	155	0.0	0.0	**-7.0**	4.6	**-5.5**	0.0	**-5.7**	**-0.1**

We analyze now the ML performance on the results with bound tightening. We already mentioned in Section 3.3 that, when RAPOSa was run with bound tightening, Gurobi was the solver less sensitive to the approach selected. This leaves ML with a more challenging task as the "best" option is now not so clear. In Table 5.10 we see the ML performance for the results with bound tightening. At

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of the normalized pace, without bound tightening.

(B) Ranking plot, without bound tightening.

FIGURE 5.10. Performance of ML without bound tightening.

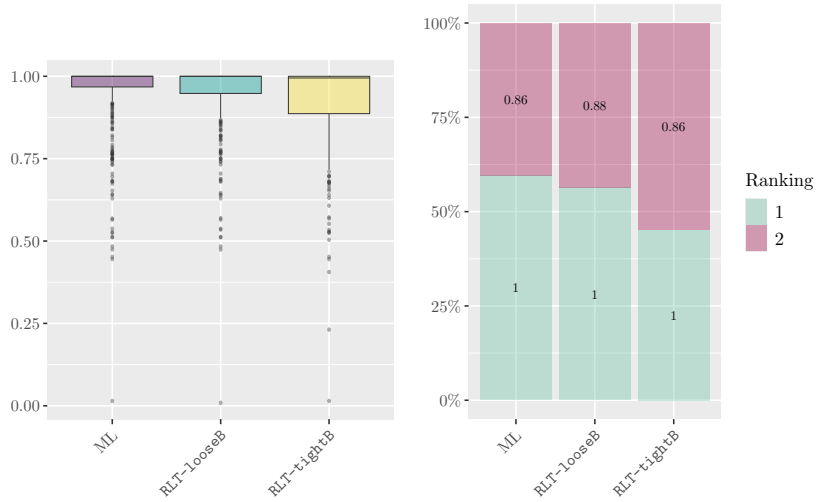
TABLE 5.11. ML selections with bound tightening.

	RLT-looseB				RLT-tightB			
	MINLP		DS		MINLP		DS	
ML	76	46.0%	153	88.0%	89	54.0%	21	12.0%
Optimal	75	45.0%	116	67.0%	90	55.0%	58	33.0%

first sight, we notice that there is less margin for the learning in MINLPLIB, but more in the case of DS. For the latter, even though there is more room to learn, the ML doesn't seem to be able to understand these changes in performance, as there is almost no improvement in gap, time and pace, but the Optimal option manages to get a 0.9% improvement in gap, a 1.5% improvement in time and a 1.3% improvement in pace. Nonetheless, the performance for DS is close to the one of Optimal. In the case of MINLPLIB, we see that ML does improve in pace and in nodes, something we also see in Figure 5.11, but it's not able to improve in gap nor in time. Note that the learning is performed on the pace, so an improvement there is expected and show the ML approach works in this setting also. Finally, As we can see in Table 5.11, in the case of MINLPLIB the percentage of selections for one approach and the other are close, in the case of DS, the numbers differ notably. This clashes with the results in Table 5.10, as its for DS where the ML is able to avoid selecting RLT-tightB in the problems where performance is worse, but for MINLPLIIB it caused the performance in time and gap to worsen by 3.9% and 2.6% respectively. This is probably due to ML selecting the worst performing option for some of the problems, even though it agrees with the Optimal selection when looking at the global percentages.

To summarize this section, it seems that ML can anticipate to some extent the best approach for the bounds of the auxiliary variables, but there is still some improvement possible for these techniques. Also, another interesting thing to study

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of the normalized pace, with bound tightening.

(B) Ranking plot, with bound tightening.

FIGURE 5.11. Performance of ML without bound tightening.

are the variables that enter the learning and which are the most relevant for the resulting models. Analyzing these interactions may allow to uncover some patterns behind the disparate behavior observed in the numerical results in Chapter 3.

5.6. Selecting between solvers

In this section, we apply the learning technique described in Section 5.2 to a different setting: selecting the best solver to solve a given problem. The aim of this section is to showcase the robustness of the technique, since it can be used outside of the main objective it was designed for. In particular, what we do is solve problems from the MINLPLIB [20] test set using different state-of-the-art solvers, such as BARON [109], Octeract [99], Couenne [7] and LindoGlobal [83] and then use the learning technique to select the best performing one for each of the problems, comparing with the `Optimal` selection and analyzing the improvement attained with the learning.

5.6.1. Computational setup. We use the entire test set MINLPLIB, since we are using general nonlinear solvers. We remove those problems where some solver didn't return an upper or lower bound, for which we can't compute the KPI. There can be different reasons behind such lack of lower or upper bound, such as numerical issues or inability of the solver to handle some specific nonlinearities. We also remove those problems where there is inconsistency between the bounds return by the different solvers, since the KPI may not be accurate for this problems. From the 1603 problems in the entire MINLPLIB test set, we end up with 514 problems.

All of the solvers are called with a 1 hour time limit, fixing the stopping criteria for the relative gap to 0.001, and fixing the `max`imum number of threads to 1, so all of them run sequentially (not all of the solvers use support parallel computation). Then, we extract the lower bound and the upper bound returned by the solver, as well as the running time. For the learning, if the running time is greater than 1 hour (sometimes solvers don't stop exactly at the time limit, and we don't want

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

this to impact the results), we modify the result and change it to 1 hour, and, if the gap is lower than 0.001 (similarly to time, we don't want the KPI to be affected by gaps smaller than the threshold set since the problem is considered to be "solved" regardless), we fix it to 0.001.

For this study, we cannot use the pace as KPI, because even if we could get the lower bound from the root node relaxation, the obtained lower bounds would not be comparable, since the different solvers use distinct reformulations and relaxations techniques. Therefore, we need to use a different KPI, but one that allows for a simultaneous comparison of the gap (for not solved problems) and the time (for solved problems). To that end, we use following KPI:

$$(1 + \text{Time}) \cdot (1 + \text{Gap}),$$

where time is the running time of the solver and gap is the `Optimality` gap. To reward those solvers that solve the problem, we divide the KPI by two if the problem is solved by that solver (i.e., if the gap is 0.001). With this KPI, and normalizing, we are in the same conditions as described above for the learning framework.

For the explanatory variables for the model, we use the properties of the instances provided with the MINLPLIB test set. In particular, we use the ones enumerated in Table 5.12. The learning is performed following the same setup as in Section 5.3, that is, using quantile regression with $\tau = 0.3$ and employing the out-of-bag predictions to present the results.

TABLE 5.12. List of properties from the MINLPLIB test set used.

Variables	No. of variables, No. of binary variables, No. of integer (non binary) variables No. of nonlinear variables, No. of nonlinear binary variables, No. of nonlinear integer (non binary) variables
Constraints	No. of constraints, No. of linear constraints, No. of quadratic constraints, No. of polynomial constraints, No. of signomial constraints, No. of general nonlinear constraints No. of nonzeros in Jacobian, No. of nonlinear nonzeros in Jacobian, Minimal coefficient, maximal coeficient

TABLE 5.13. Performance of the different solvers.

(514 instances)	(514) Solved	(159) Gap	(283) Time	(514) KPI
Solver 1	346	0.002	2.78	2.37
Solver 2	470	0.003	9.57	3.75
Solver 3	457	0.007	24.84	6.46
Solver 4	458	0.016	33.96	8.11

5.6.2. Numerical results. In Table 5.13 we have the performance results for the different solvers. We call the solvers Solver 1, 2, 3 and 4. This is done to avoid diverting the attention towards a comparison of the solvers' performance, since we want to study how these solvers can complement each other instead. We

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

are representing the number of solved problems and the geometric mean of the gap, time and the new KPI. The exclusions used for each metric are the same as described in Section 1.9. For the new KPI we consider all instances. What stands out in the results is that, the best performing solver according to gap, time and the KPI, is the one that solves less problems overall. This is one reason behind what led us to use a KPI that rewards solving the problem, as we want to capture this behavior and use the learning to improve the time, the gap but also select a solver that solves the problem and consequently improve the number of solved problems.

TABLE 5.14. Machine Learning performance selecting between solvers.

(514 instances)	(514) Solved	(159) Gap	(283) Time	(514) KPI
`Best` (Solver 1)	346	0.0020	2.78	2.37
Out-of-bag Q-RF	457	0.0021	1.84	1.76
`Optimal`	479	0.0015	1.12	1.34
Improvement after learning	32.08%	3.47%	−33.72%	−25.78%
`Optimal` improvement	38.44%	−23.23%	−59.68%	−43.45%

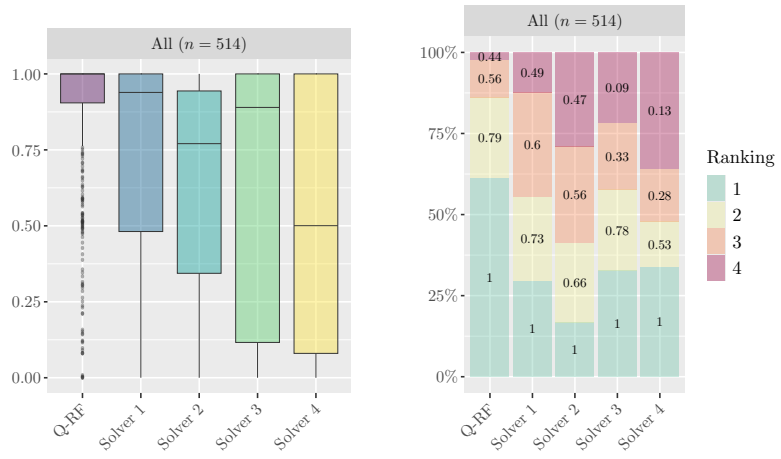
In Table 5.14 we can see the results of the learning. First of all, we see that there is ample room for improvement. Selecting the appropriate solver for each instance can deliver up to a 59.68% improvement in the solving time, a 23.23% improvement in the `Optimal`ity gap and solve 38.44% more instances than the best solver overall (according to the KPI). Additionally, we improve upon the best results in every performance metric with respect to Table 5.13. This shows the potential for learning in this setting. Regarding the learning itself, we obtain good results, as we are able to improve a 33.72% in time (57% of the total improvement possible), 25.78% in KPI (59% of the total improvement possible), and solve 111 instances more than `Best`. We don't improve the gap, but note that the values are very small and so the margin for improvement is not large. It might happen that with a different KPI we could get an improvement in gap by sacrificing for example the improvement in the number of solved problems. Changing the KPI to reward more the improvement in gap and less solving the problem could lead to different results on that regard.

In Figure 5.12, we can analyze a bit further the learning results. First, in Figure 5.12a we can see a representation of the normalized KPI, and check that the learning improves significantly upon the other solvers. The same can be seen in Figure 5.12b, where the "learned solver" is the best for 60% of the instances, while performing well (according to the KPI) when it's the second or third best. Finally, in Figure 5.12, we see that all the solvers are selected by `Optimal` for some instances, and the selection made by Q-RF is comparable to `Optimal`, which indicates that we are indeed been able to learn which is the best solver for each instance.

5.7. Conclusions

In this chapter we have described the learning framework from [54, 58], including the KPI used, the explanatory variables considered and the learning technique chosen. This learning technique defines a general framework that can be used for

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Boxplots of the normalized KPI. (B) Ranking plot.

(C) Percentage selected solver for Q-RF and `Optimal`.

FIGURE 5.12. Performance of ML for selecting between solvers.

example to select between different configurations of an optimization algorithm (as the RLT technique), but also between different solvers for the same type problems, as long as an adequate KPI is defined and appropriate explanatory variables selected. In this chapter we have shown the robustness of the technique, applying it to the results from the different chapters in Part 1. As general as this framework is, adapting it to the different contexts on which it's applied could deliver even better results.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 6

Extending the learning framework to a dynamic setting

Publication derived from this chapter: Brais González-Rodríguez et al. *Learning in Spatial Branching: Limitations of Strong Branching Imitation*. 2024. arXiv: 2406.03626 [math.OC]. URL: https://arxiv.org/abs/2406.03626

6.1. Introduction

In the field of optimization and, more specifically, in the context of branch-and-bound algorithms, branching decisions significantly affect the efficiency and speed of finding the optimal solution. Strong branching is a specific type of branching technique that involves evaluating all possible branching variables by simulating the branching process for each potential variable and measuring the impact on the optimality gap. Although strong branching is known to be highly effective in reducing the size of the search tree and accelerating the convergence to an optimal solution, it is computationally prohibitive because it requires solving, at each and every node of the branch-and-bound tree, two optimization problems for each candidate variable. Learning to branch leverages machine learning techniques to imitate or enhance traditional branching strategies like strong branching. By using historical data on past branching decisions and their outcomes, machine learning models are trained to approximate the decision-making quality of strong branching without incurring high computational costs, thus balancing efficiency and computational speed. As can be seen in the recent surveys [85, 9, 112], most of the research so far has centered on mixed integer linear programming due to its wide range of applications. This chapter contributes to the same strand of literature, but in the context of nonlinear problems.

Regarding the specific learning task of branching variable selection, a common approach involves using a known expert system as a model and training on past data to imitate it. As discussed above, given that strong branching is (experimentally) known to produce significantly smaller trees than any other branching rule, it is not surprising that, in this context, most literature has focused precisely on learning to imitate strong branching, as in [72, 2, 3, 52, 63]. For instance, [2] present an offline supervised learning method to predict strong branching scores in branch-and-bound algorithms for solving mixed-integer linear programming problems. [72] explore an online framework that ranks variables to replicate strong branching. On the other hand, [52] use offline learning to encode the branching policies into a graph convolutional neural network and have fast approximations for strong branching. [63] extend this work to a hybrid offline learning model that combines the capabilities of graph neural networks with the less computationally demanding multi-layer perceptrons for more efficient branching.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Despite the widely spread use of strong branching as an expert to imitate, recent research has highlighted some limitations associated with this approach. As discussed in [112], "The strong branching heuristic has been used by many as an expert from which we can learn effective decision-making. The claim that strong branching is a desirable strategy to follow has recently been challenged, with some notable examples of instances where strong branching scores provide no useful information". [50] argue that, when measuring the impact of strong branching on metrics such as the number of nodes of the branch-and-bound tree, careful consideration is needed to obtain fair comparisons. In particular, since strong branching can identify infeasibilities (and does not perform branching at nodes where it identifies such infeasibilities) and provide faster dual-bound improvements, standard implementations of strong branching effectively reduce the size of the resulting tree in a way that no rule aimed at imitating strong branching can ever achieve. Second, as also highlighted in [112], "strong branching relies on the LP relaxation for scoring variables, which can provide little information in cases when the optimal LP objective value does not change with branching. In such cases, strong branching cannot be considered a reliable expert." These concerns have motivated research on expert-free approaches, in which reinforcement learning is used to learn a branching rule from scratch as in [36], [103], and [111].

Although the use of learning in nonlinear programming (NLP) has also been on the rise in recent years (see, for instance, [4], [14], and [71]), not much has been done on learning to branch. An exception is [54], where the authors develop an offline learning framework for spatial branching based on ideas of algorithm selection [56], similar to those used in [33] for MILP problems. In this chapter we extend the work in [54], in which learning takes place at the instance level with static features only, and study to what extent learning at the node level with dynamic features can outperform the baseline static approach. On the one hand, for a given set of optimization problems, the dynamic setting provides many more observations for the training. Each instance results in as many observations as nodes of the branch-and-bound tree, instead of having just one observation associated with each instance. On the other hand, the information obtained on a node-by-node basis is much noisier, which can definitely be challenging for the learning.

The contribution of this chapter is not to assess the performance of a specific learning technique, but to determine the potential of a learning framework itself. More specifically, we evaluate the potential of three learning frameworks, by analyzing the performance of three different experts. The first one, $\texttt{ORule}^{\texttt{S}}$, is an expert capable of choosing, for each instance, the best performing branching rule from a predefined pool; it represents the best that can be achieved with the static learning in [54]. The second one, $\texttt{BRule}^{\texttt{D}}$, is an expert capable of choosing, for each node of the branch-and-bound tree, the best performing branching rule (at that node) from a predefined pool; it represents the best that can be achieved by extending the static learning setting to accommodate dynamic features and learn at the node level. The third one, $\texttt{BVar}^{\texttt{D}}$, is an expert capable of choosing, for each node of the branch-and-bound tree, the best variable (at that node), in the sense of perfectly imitating strong branching choices. Importantly, note that experts $\texttt{BRule}^{\texttt{D}}$ and $\texttt{BVar}^{\texttt{D}}$ can be seen as targets both for offline and online learning and, indeed, as we discussed above, $\texttt{BVar}^{\texttt{D}}$ has been the most common choice by past literature under

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

both settings. Therefore, our findings apply to both offline and online learning frameworks.

We frame our contribution in the context of continuous polynomial optimization problems, where we study branching variable selection for a spatial branch-and-bound algorithm that builds upon the RLT technique [120]. Our numerical results show that $\texttt{ORule}^{\texttt{S}}$ is the best performing expert: both $\texttt{BRule}^{\texttt{D}}$ and $\texttt{BVar}^{\texttt{D}}$ fall behind for the various performance metrics under consideration. This finding might seem counter intuitive at first, since the last two experts should benefit from the richness of the dynamic setting and also from the possibilities of using online learning or hybrid offline-online approaches. Yet, what seems to be limiting these experts is the fact that they make, node by node, the best myopic branching decision based on the achieved lower bound improvement at the current node. Thus, there is no guarantee that this locally optimal choice will result in a good enough overall performance and, indeed, we observe that $\texttt{ORule}^{\texttt{S}}$, the expert of the (offline) static framework is clearly superior.

It is worth noting that the use of an RLT-based algorithm for polynomial optimization problems is merely a means of assessing the potential of the different experts and learning frameworks. Thus, although polynomial optimization is an important class of problems on its own right, we believe that the implications and insights of our findings can be extrapolated to general NLP and MINLP problems. In particular, the relatively weak performance obtained for $\texttt{BVar}^{\texttt{D}}$, might guide future efforts in the design of branching rules for spatial branching in directions whose goal is not necessarily to approximate strong branching decisions. This chapter is based on [60], a joint work with Brais González Rodríguez, Bissan Ghaddar, Julio González Díaz and Beatriz Pateiro López.

6.2. Framework for the analysis

In this section, we describe the elements required to develop the computational study to assess the potential of learning schemes based on the imitation of three different experts: $\texttt{ORule}^{\texttt{S}}$, $\texttt{BRule}^{\texttt{D}}$, and $\texttt{BVar}^{\texttt{D}}$.

6.2.1. Porfolio of branching rules. We use rules `dual`, `range`, `eig-CMI`, `dual rel.`, `range rel.` and `eig-CMI rel.` as described in Section 1.5.

Once we have defined the portfolio of branching rules, we move to the learning frameworks and associated experts.

6.2.2. Learning frameworks and experts. In this section, we formally present the three learning frameworks and the three associated experts whose potential is the main object of analysis in this chapter. Each of them represents, in a sense that we clarify below, the best that might ever be achieved within a certain learning framework. Therefore, as already mentioned, the contribution of this chapter is not to assess the performance of a specific learning technique, but to determine the potential of a learning framework itself. More precisely, we want to understand how much a branch-and-bound algorithm for nonlinear optimization might possibly benefit by enhancing branching variable selection with machine learning. This question is studied for three different learning frameworks, which we describe below.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

6.2.2.1. *Instance-by-instance rule learning with static features.* Consider an offline learning setting in which there is a pool of branching rules available and the goal is to learn to choose, instance by instance, the best performing branching rule according to a given metric. Since the learning is performed at the instance level, it can only use features that depend on the specific characteristics of each optimization problem, i.e., static features. Hereafter, we refer to this learning framework as *static rule learning*.

The absolute best that can be achieved with any learning scheme in the static learning setting is to perfectly predict the best performing branching rule on all instances. $\texttt{ORule}^{\texttt{S}}$ represents an expert capable of making these perfect predictions. Note that this expert is optimal in the following sense: no learning scheme designed to assign a branching rule from the pool to each instance can ever outperform $\texttt{ORule}^{\texttt{S}}$.

A limitation of the static learning framework is that the learning is not adaptive, since one commits to applying the same branching rule at all the nodes of the branch-and-bound tree.

6.2.2.2. *Node-by-node rule learning with dynamic features.* A natural extension of the above setting is to carry out the learning scheme on a node-by-node basis. This approach allows the incorporation of dynamic features into the learning process using, for instance, information about past branching decisions, the evolution of the bounds, or the structure of the branch-and-bound tree. In order to measure the performance at a given node, we take as *key performance indicator*, KPI, the improvement with respect to the lower bound of the parent node. Therefore, the goal is to learn to choose, node by node, the branching rule from the pool that will deliver the best KPI. We refer to this framework as *dynamic rule learning*.

We use $\texttt{BRule}^{\texttt{D}}$ to refer to an expert capable of perfectly predicting the branching rule with the best KPI at each node. Note that, for this expert, we have chosen to refer to it as "Best" instead of "Optimal", because the chosen KPI is a myopic one, since it is only a measure of the improvement at the current node, i.e., it does not capture the performance further down the tree. Expert $\texttt{BRule}^{\texttt{D}}$ can be seen as the target to imitate in any setting with node-by-node learning, regardless of whether the learning takes place offline or online.

Despite the advantages of the additional richness of the dynamic setting, node-by-node information can also be very noisy, which poses new challenges. Further, it is uncertain whether the optimal node-by-node decisions of $\texttt{BRule}^{\texttt{D}}$ will result in a good overall performance, since they might be limited by the myopic nature of the KPI. This potential limitation of $\texttt{BRule}^{\texttt{D}}$ is also shared by the next expert, strong branching imitation. Nonetheless, the latter is still the most common approach in the learning-to-branch literature (as discussed in Section 6.1).

6.2.2.3. *Node-by-node variable learning with dynamic features.* The two experts defined above, $\texttt{ORule}^{\texttt{S}}$ and $\texttt{BRule}^{\texttt{D}}$, focus on learning to select the best rule from a given pool or branching rules. Yet, most of the research in the field focuses instead on learning the best branching variable and, more precisely, on strong branching imitation, which is the approach behind our third expert. Consider again the learning setting with dynamic features and the KPI used to introduce $\texttt{BRule}^{\texttt{D}}$. The goal now is to choose, node by node, the branching variable that provides the largest improvement with respect to the lower bound of the parent node. We refer to this framework as *dynamic variable learning*.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

We use $\texttt{BVar}^{\texttt{D}}$ to refer to an expert capable of perfectly predicting the branching variable with the best KPI at each node. Note that, as defined, $\texttt{BVar}^{\texttt{D}}$ coincides with (perfect) strong branching imitation. The above discussion for $\texttt{BRule}^{\texttt{D}}$ on the use of "Best" instead of "Optimal" and on its potential limitations because of the myopia of the KPI also apply to $\texttt{BVar}^{\texttt{D}}$. Expert $\texttt{BVar}^{\texttt{D}}$ can be seen as the target to imitate in any setting with node-by-node learning, regardless of whether the learning takes place offline or online.

6.3. Assessing learning frameworks

In this section, we present the computational analysis of the three learning frameworks discussed in Section 6.2.2: static rule learning, dynamic rule learning, and dynamic variable learning. In order to analyze the potential of these three learning frameworks, we study the performance of the associated experts: $\texttt{ORule}^{\texttt{S}}$, $\texttt{BRule}^{\texttt{D}}$, and $\texttt{BVar}^{\texttt{D}}$, respectively.

6.3.1. Static rule learning. We take as our starting point the learning framework developed in [54] in the context of what we have called static rule learning, discussed in Section 5.2. Table 6.1 reports the results of the application of that learning methodology to the pool of rules comprised of the six branching rules defined in Section 6.2.1. In particular, the learning is conducted jointly on all sets of instances using quantile regression forests [90], and the rule resulting from the learning process is referred to as `ML-Q-RF`. The statistical analysis was developed in programming language `R` [106], using library `ranger` [133].

TABLE 6.1. Pool of branching rules and resulting machine learning rule.

	(409) Solved	(159) Gap	(120) Time	(265) Pace	(235) Nodes
`dual`	254	0.114	46.02	9.34	98.99
`range`	249	0.114	51.20	9.76	94.74
`eig-CMI`	243	0.136	93.85	11.21	135.71
`dual rel.`	249	0.130	45.25	9.46	89.52
`range rel.`	252	0.115	46.25	6.73	88.89
`eig-CMI rel.`	242	0.145	78.12	10.23	118.09
`ML-Q-RF`	257	0.097	39.70	4.56	89.10

The results in Table 6.1 show that `ML-Q-RF` improves upon the best performing rule according to pace, `range rel.`, by 32% (from 6.73 to 4.56) and delivers also significant improvements in solved, time, and gap.[1] In Table 6.2 we assess the performance of our first expert, $\texttt{ORule}^{\texttt{S}}$, by comparing it with `range rel.` and `ML-Q-RF`.[2] In this case, the difference in pace between `range rel.` and `ML-Q-RF` is 33%, whereas $\texttt{ORule}^{\texttt{S}}$ delivers an additional 14%, almost reducing pace to half

[1] Our results do not reproduce exactly those reported in [54], since there are some changes in the computational setup, such as the version of RAPOSa used in the analysis.

[2] Recall that the results in Table 6.2 for `range rel.` and `ML-Q-RF` do not need to coincide with those in Table 6.1. This is because of the exclusions applied to compute the values for gap, time, pace, and nodes described in Section 1.9, which leads to slightly different numbers of instances considered for each metric.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

(from 9.19 to 4.87) and producing substantial improvements on the five performance metrics under study.

TABLE 6.2. Assessing the potential of $\texttt{ORule}^{\texttt{S}}$.

	(409) Solved	(145) Gap	(108) Time	(253) Pace	(250) Nodes
`range rel.`	252	0.182	69.78	9.19	115.47
`ML-Q-RF`	257	0.150	59.57	6.15	108.25
$\texttt{ORule}^{\texttt{S}}$	259	0.134	45.58	4.87	92.76

Given the potential of both `ML-Q-RF` and $\texttt{ORule}^{\texttt{S}}$ illustrated in Tables 6.1 and 6.2, one may want to explore to what extent the results could be further improved by enriching the relatively simple static rule learning framework above. A first approach would be to add more diverse and competitive branching rules to the pool since, by construction, the more rules in the pool, the better $\texttt{ORule}^{\texttt{S}}$ will perform. This idea was already explored in [54], where the authors found that `ML-Q-RF` also benefits from the additional richness. A second approach, which we consider here, consists of adapting the algorithm selection approach to the dynamic rule learning framework, i.e., learning to select the best rule on a node-by-node basis using dynamic features.

6.3.2. Dynamic rule learning. As discussed, the promising results of the previous section have been obtained in a setting in which features are restricted to be static and where the size of the training set is limited by the available number of instances. Thus, it is reasonable to question how much further one might go in a setting where learning takes place at the node level, where new features would be available to capture the impact of past decisions and the evolution of the branch-and-bound algorithm and where, moreover, one might also rely on online learning. This is what we referred to as dynamic rule learning in Section 6.2.2.2, and $\texttt{BRule}^{\texttt{D}}$ is an expert capable of perfectly predicting the branching rule with the best KPI at each node.

It is important to note that there will be nodes along the branch-and-bound tree for which different rules select different branching variables and, still, have the same KPI. Probably the most common (and relevant) situation in which this happens is when all rules lead to a KPI of 0, i.e., no improvement with respect to the lower bound of the parent node. Therefore, in order to perfectly define $\texttt{BRule}^{\texttt{D}}$, we have to specify how to proceed when such ties arise. Since the main goal of this section is to assess the extra potential of the dynamic rule framework with respect to the static one, we explore two possibilities, $\texttt{BRule}^{\texttt{D,Q-RF}}$ and $\texttt{BRule}^{\texttt{D,opt}}$, which in case of a tie follow, respectively, the choice that `ML-Q-RF` and $\texttt{ORule}^{\texttt{S}}$ would make for the instance under consideration. In particular, $\texttt{BRule}^{\texttt{D,opt}}$ allows the expert of the dynamic setting to build upon the expert of the static learning framework in situations where the chosen KPI produces ties.

Table 6.3 presents the performance of $\texttt{BRule}^{\texttt{D,Q-RF}}$ and $\texttt{BRule}^{\texttt{D,opt}}$ side by side with the performance of `ML-Q-RF` and $\texttt{ORule}^{\texttt{S}}$. Surprisingly, we see that $\texttt{ORule}^{\texttt{S}}$ is significantly superior to both $\texttt{BRule}^{\texttt{D,Q-RF}}$ and $\texttt{BRule}^{\texttt{D,opt}}$ in terms of pace, gap, and time, while falling narrowly behind in the number of solved instances. Thus,

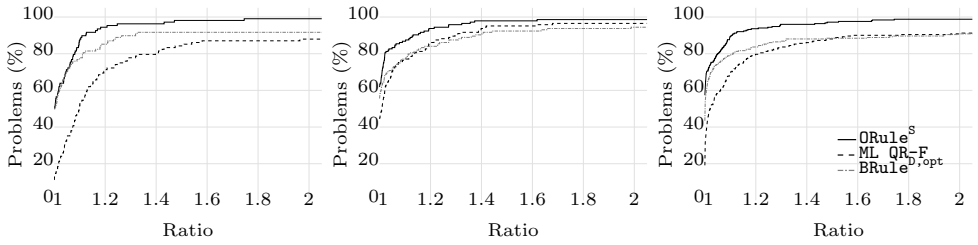
USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

TABLE 6.3. Assessing the potential of $\texttt{BRule}^{\texttt{D}}$.

	(409) Solved	(142) Gap	(109) Time	(252) Pace	(256) Nodes
ML-Q-RF	257	0.191	63.34	6.48	125.22
$\texttt{ORule}^{\texttt{S}}$	259	0.170	48.36	5.12	105.14
$\texttt{BRule}^{\texttt{D,Q-RF}}$	261	0.183	53.78	6.38	104.51
$\texttt{BRule}^{\texttt{D,opt}}$	260	0.194	53.63	6.13	105.19

it seems that the myopic nature of the chosen KPI represents a ceiling on what can be achieved in the dynamic rule framework, whose potential is below what can be achieved by $\texttt{ORule}^{\texttt{S}}$. Interestingly, even ML-Q-RF is highly competitive with both variants of $\texttt{BRule}^{\texttt{D}}$, and specially so in terms of pace, the KPI on which it was trained. These findings suggest that it may be more promising to enhance the relatively simple approaches in static rule learning, rather than pursuing what are likely to be much more complex techniques in the context of dynamic rule learning.

Figure 6.2 presents three performance profiles [35], which provide additional information to compare ML-Q-RF, $\texttt{ORule}^{\texttt{S}}$, and $\texttt{BRule}^{\texttt{D,opt}}$, according to running time, optimality gap, and pace. The three of them are consistent with the information in Table 6.3, but the superiority of $\texttt{ORule}^{\texttt{S}}$ is even more noticeable.



(A) Running time. (B) Optimality gap. (C) Pace.

FIGURE 6.1. Performance profiles.

6.3.3. Dynamic variable learning. Given the negative results obtained for $\texttt{BRule}^{\texttt{D}}$ in the previous section, it is important to get a better understanding on why the dynamic rule learning framework has a more limited potential than the much simpler static one. As discussed above, a natural explanation is that the underlying KPI is completely myopic, so even an expert capable of following the rule with the best KPI at each node may not exhibit a good overall performance. Alternatively, one may argue that the limitation comes from the algorithm selection approach, under which the only candidate variables for branching at a given node are those chosen by at least one of the branching rules from the pool. In the latter case, we should observe that an expert capable of choosing, at each node, the branching variable with the best KPI, should outperform $\texttt{BRule}^{\texttt{D}}$. This is exactly what we study in this section. Evaluating the performance of such an expert, which we have called $\texttt{BVar}^{\texttt{D}}$, is equivalent to evaluating the performance of perfectly imitating strong branching, which, as discussed in the introduction, has been the goal of most of the literature on learning to branch in MILP problems.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Similar to the above section with $\texttt{BRule}^{\texttt{D}}$, we also need to specify what variables are selected for branching when ties in KPI occur. Again, we consider two variants of $\texttt{BVar}^{\texttt{D}}$: $\texttt{BVar}^{\texttt{D,Q-RF}}$ and $\texttt{BVar}^{\texttt{D,opt}}$, which in case of a tie follow, respectively, the choice that ML-Q-RF and $\texttt{ORule}^{\texttt{S}}$ would make for the instance under consideration. There is another aspect regarding ties that will be relevant in the ensuing analysis, namely, whether or not we should include some tolerance when considering ties. To this end, $\texttt{BVar}_0^{\texttt{D,Q-RF}}$ and $\texttt{BVar}_0^{\texttt{D,opt}}$ denote the versions of $\texttt{BRule}^{\texttt{D,Q-RF}}$ and $\texttt{BRule}^{\texttt{D,opt}}$ in which only a perfect coincidence of the KPI (to machine accuracy) is considered a tie. On the other hand, we later consider as well variants such as $\texttt{BVar}_{0.005}^{\texttt{D,opt}}$ or $\texttt{BVar}_{0.01}^{\texttt{D,opt}}$, to follow $\texttt{ORule}^{\texttt{S}}$ also in situations where the best KPI delivers an improvement no superior to 0.005 or 0.01, respectively.

Given that strong branching is computationally very intensive, and given that this burden rapidly increases with the number of variables, for the results reported in this section we have restricted our test set to instances with 20 variables or less. This reduces the size of the test set from 409 instances to 240. It is worth noting that, for the numeric results below, we assume that $\texttt{BVar}^{\texttt{D}}$ and its variants can perfectly imitate strong branching with no computational overhead, i.e., they are just assumed to know, for each and every node of the branch-and-bound tree, the KPI associated to each and every variable.

TABLE 6.4. Assessing the potential of $\texttt{BVar}^{\texttt{D}}$.

	(240) Solved	(43) Gap	(89) Time	(124) Pace	(192) Nodes
ML-Q-RF	197	0.018	71.82	0.35	95.87
$\texttt{ORule}^{\texttt{S}}$	198	0.014	58.41	0.28	84.90
$\texttt{BRule}^{\texttt{D,opt}}$	199	0.018	61.73	0.28	83.85
$\texttt{BVar}_0^{\texttt{D,Q-RF}}$	196	0.031	70.74	0.36	79.91
$\texttt{BVar}_0^{\texttt{D,opt}}$	196	0.032	72.27	0.37	79.72

Table 6.4 presents the performance of $\texttt{BVar}_0^{\texttt{D,Q-RF}}$ and $\texttt{BVar}_0^{\texttt{D,opt}}$. The results show that, once again, the potential of the dynamic learning setting is inferior to that of the static one. Yet, before drawing any conclusions, it is convenient to check if this behavior may be driven by the zero tolerance considered for ties.

TABLE 6.5. Assessing the potential of $\texttt{BVar}^{\texttt{D}}$ with different thresholds on ties.

	(240) Solved	(46) Gap	(90) Time	(129) Pace	(188) Nodes
$\texttt{BVar}_0^{\texttt{D,opt}}$	196	0.025	51.89	0.33	74.00
$\texttt{BVar}_{0.005}^{\texttt{D,opt}}$	193	0.024	53.65	0.31	74.06
$\texttt{BVar}_{0.01}^{\texttt{D,opt}}$	192	0.023	53.58	0.31	74.89
$\texttt{BVar}_{0.02}^{\texttt{D,opt}}$	192	0.023	54.99	0.32	76.42
$\texttt{BVar}_{0.05}^{\texttt{D,opt}}$	194	0.022	59.86	0.36	78.00

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Table 6.5 represents a comparison between five difference values for the tolerance on ties, ranging from 0 to 0.05. Although all 5 versions of $\texttt{BVar}^{\texttt{D}}$ perform

comparably well, $\texttt{BVar}^{\texttt{D},\texttt{opt}}_{0.01}$ delivers some improvement in terms of gap, without compromising much on time and pace. Since the performance according to gap was probably the weakest point of $\texttt{BVar}^{\texttt{D},\texttt{opt}}_{0}$ in Table 6.4, we present a last table comparing the performance of $\texttt{BVar}^{\texttt{D},\texttt{opt}}_{0.01}$ with respect to ML-Q-RF, $\texttt{ORule}^{\texttt{S}}$, and $\texttt{BRule}^{\texttt{D}}$.

TABLE 6.6. Comparing the potential of $\texttt{ORule}^{\texttt{S}}$ with the best configurations of $\texttt{BRule}^{\texttt{D}}$ and $\texttt{BVar}^{\texttt{D}}$.

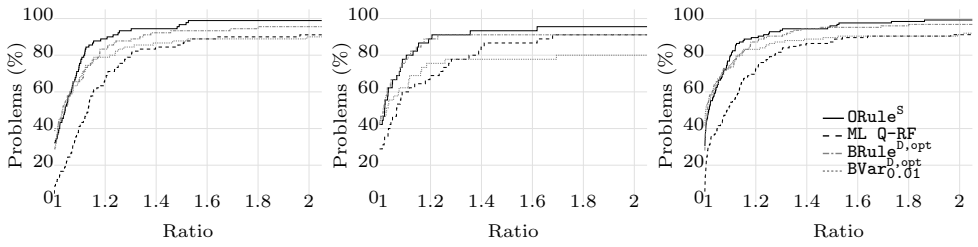
	(240)	(44)	(90)	(125)	(190)	(190)
	Solved	Gap	Time	Pace	Nodes	Mean nodes
ML-Q-RF	197	0.017	69.07	0.34	87.81	835.88
$\texttt{ORule}^{\texttt{S}}$	198	0.013	56.31	0.27	79.33	672.48
$\texttt{BRule}^{\texttt{D},\texttt{opt}}$	199	0.017	59.66	0.28	79.09	803.02
$\texttt{BVar}^{\texttt{D},\texttt{opt}}_{0.01}$	192	0.026	72.37	0.34	76.22	844.89

The results in Table 6.6 confirm that, not only $\texttt{BVar}^{\texttt{D}}$ is inferior to $\texttt{ORule}^{\texttt{S}}$, but it is also clearly inferior to $\texttt{BRule}^{\texttt{D}}$. We believe that this finding is important on its own. In the context of MILP problems, it has been widely documented that strong branching produces the smallest trees, with no other rule becoming close to it [1, 50]. In order to assess this in our setting, Table 6.6 contains one additional column, reporting the standard mean of the average number of nodes, which complements the information about the geometric mean in Nodes, a measure less sensitive to outliers. The geometric means reported in nodes show that, also in our NLP setting, strong branching imitation tends to produce the smallest trees and, yet, its standard mean is the largest one. A deeper examination of the numerical results behind Table 6.6 confirms that this is precisely because, although $\texttt{BRule}^{\texttt{D}}$ performs very well on most instances, there is a non-negligible number of them in which it performs poorly. Thus, despite being the best expert according to Nodes, $\texttt{BVar}^{\texttt{D}}$ falls clearly behind $\texttt{ORule}^{\texttt{S}}$ and $\texttt{BRule}^{\texttt{D}}$ according to all other metrics. Interestingly, even ML-Q-RF is very competitive with $\texttt{BVar}^{\texttt{D}}$.

It is natural to wonder what may be the mechanism behind the negative results for $\texttt{BVar}^{\texttt{D}}$, i.e., strong branching imitation, in our NLP setting. When compared to $\texttt{BVar}^{\texttt{D}}$, $\texttt{BRule}^{\texttt{D}}$ has the advantage of using information about the magnitudes of the violations of RLT defining identities in Eq. (1), and also the weights in Eq. (4) can capture information about the evolution of the branching process. For instance, `range` and `range rel.` take into account how much the bounds of the variables have changed from their values at the root node. This information allows $\texttt{BRule}^{\texttt{D}}$ to disregard some variables that might then be selected by strong branching based on their "myopic" KPI, which just focuses on LB improvement. The violations of the RLT defining identities and the $w(j, J)$ weights convey additional information that may pay off deeper in the tree. In this sense, although $\texttt{BVar}^{\texttt{D}}$ may be improving the lower bound faster at the current node compared to $\texttt{ORule}^{\texttt{S}}$ and $\texttt{BRule}^{\texttt{D}}$, it may not be reducing so well the size of the infeasibilities.

Figure 6.2 presents performance profiles comparing ML-Q-RF, $\texttt{ORule}^{\texttt{S}}$, $\texttt{BRule}^{\texttt{D}}$, and $\texttt{BVar}^{\texttt{D}}$ according to time, gap, and pace. The results are consistent with those reported in Table 6.6, but they also allow us to get additional insights. We can see that, for small ratios in the performance profiles, $\texttt{BVar}^{\texttt{D}}$ is very competitive, and

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA



(A) Running time. (B) Optimality gap. (C) Pace.

FIGURE 6.2. Performance profiles.

these small ratios already account for a relatively large percentage of the instances. This means that $\texttt{BVar}^{\texttt{D}}$ is very effective in most of the instances but, as the ratios become larger (i.e., relative differences in performance become larger), it starts falling significantly below $\texttt{ORule}^{\texttt{S}}$ and $\texttt{BRule}^{\texttt{D}}$, and, eventually, it falls even below ML-Q-RF. This behavior confirms what we argued above: although $\texttt{BRule}^{\texttt{D}}$ performs well in most instances, it also performs poorly on a relevant amount of them, which suffices to significantly deteriorate its overall performance.

6.4. Conclusions

This chapter reveals interesting insights into the effectiveness of different branching rules and variable selection strategies for general nonlinear programming problems when using a spatial branch-and-bound algorithm. The analysis was done on RLT for polynomial programming problems however everything can be extended to general nonlinear programming. In particular, we analyze three different experts that learning approaches can try to imitate and show the limitations of two of them ($\texttt{BVar}^{\texttt{D}}$ and $\texttt{BRule}^{\texttt{D}}$), which rely on dynamic features compared to the one that is based on static features ($\texttt{ORule}^{\texttt{S}}$). These limitations primarily stem from the shortsighted decisions made using local information from the branch-and-bound tree. The dynamic use of features, while seemingly adaptive, often fails to provide long-term benefits, rendering it less effective against the $\texttt{ORule}^{\texttt{S}}$ expert. This underscores the challenge of relying too heavily on information that only pertains to the current state of the tree, without considering broader implications.

Among the three experts, $\texttt{ORule}^{\texttt{S}}$ outperforms the others while $\texttt{BRule}^{\texttt{D}}$, which relies on dynamic branching rule selection, performs less effectively because of the myopic nature of the KPI it uses which sets a limit on the attainable outcomes within the dynamic rule framework. Meanwhile, $\texttt{BVar}^{\texttt{D}}$ is the least effective of the three. Although $\texttt{BRule}^{\texttt{D}}$ might not always choose the best immediate options, it benefits from incorporating insights into the branching process' progression. This allows $\texttt{BRule}^{\texttt{D}}$ to exclude certain variables that $\texttt{BVar}^{\texttt{D}}$ might select based on their immediate, myopic, KPI focused solely on bound improvement.

The underperformance of $\texttt{BVar}^{\texttt{D}}$, which attempts to mimic strong branching, suggests that future efforts in developing branching rules for spatial branching in the context of nonlinear optimization should perhaps not focus solely on approximating strong branching. Instead, these efforts might aim to explore new directions such as extending reinforcement learning to spatial branching which can potentially lead to more effective branching strategies in complex optimization environments. Other

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

directions worth exploring are finding a better KPI to learn in the dynamic setting and enriching the portfolio of branching rules in the static learning setting.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Part 3

Optimizing Power Networks

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

List of Symbols

Sets		
N	set of buses	
G	set of generators	
$L \subset N \times N$	set of connections	L
$H_i \subset G, i \in N$	generators per bus	H_i
T	time periods	
Parameters		
C_i^0	operating cost 0 of generator i	$i \in G$
C_i^1	operating cost 1 of generator i	$i \in G$
C_i^2	operating cost 2 of generator i	$i \in G$
C_i^{SU}	start-up cost of generator i	$i \in G$
P_{it}^L	real power load bus i at time t	$i \in N, t \in T$
Q_{it}^L	reactive power load bus i at time t	$i \in N, t \in T$
R_i^{SU}	ramp-up rate of generator i	$i \in G$
R_i^{SD}	shut-down rate of generator i	$i \in G$
P_i^{SU}	start-up capacity of generator i	$i \in G$
P_i^{SD}	shut-down capacity of generator i	$i \in G$
$P_i^{\min}$	min real power of generator i	$i \in G$
$P_i^{\max}$	max real power of generator i	$i \in G$
$Q_i^{\min}$	min reactive power of generator i	$i \in G$
$Q_i^{\max}$	max reactive power of generator i	$i \in G$
T_i^U	minimum up time of generator i	$i \in G$
T_i^D	minimum down time of generator i	$i \in G$
$V_i^{\min}$	min voltage of bus i	$i \in N, t \in T$
$V_i^{\max}$	max voltage of bus i	$i \in N, t \in T$
$S_{ij}^{\max}$	max apparent power of branch ij	$(i,j) \in L$
$\tilde{\boldsymbol{Y}}$	admittance matrix	
$\boldsymbol{Y}_i^G$	auxiliary matrix	$i \in N$
$\boldsymbol{Y}_{ij}^B$	auxiliary matrix	$(i,j) \in L$
$\boldsymbol{E}_i^G$	auxiliary matrix	$i \in N$
$\bar{\boldsymbol{E}}_i^G$	auxiliary matrix	$i \in N$
$\boldsymbol{E}_{ij}^B$	auxiliary matrix	$(i,j) \in L$
$\bar{\boldsymbol{E}}_{ij}^B$	auxiliary matrix	$(i,j) \in L$
$\boldsymbol{M}_i$	auxiliary matrix	$i \in N$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Variables

P_{it}^G	real power gen. i at period t	$i \in G, t \in T$
Q_{it}^G	reactive power gen. i at period t	$i \in G, t \in T$
P_{it}	real power node i at period t	$i \in N$ s.t. $H_i \neq \emptyset, t \in T$
Q_{it}	reactive power node i at period t	$i \in N$ s.t. $H_i \neq \emptyset, t \in T$
$\tilde{P}_{ijt}$	real power branch (i,j) at period t	$(i,j) \in L, t \in T$
$\tilde{Q}_{ijt}$	reactive power branch (i,j) at period t	$(i,j) \in L, t \in T$
v_{it}	gen. i on at period t	$i \in G, t \in T$
y_{it}	gen. i turned on beginning period t	$i \in G, t \in T \backslash \{0\}$
z_{it}	gen. i turned off beginning period t	$i \in G, t \in T \backslash \{0\}$
$\boldsymbol{X}_t$	voltage time t $(2 \cdot \lvert N \rvert \times 1)$	$t \in T$

Other

$\mathcal{S}^n$	space of symmetric matrices of order n
$\mathcal{S}_+^n$	positive semidefinite matrices
$\mathcal{Q}^n$	second-order cone
$\mathcal{Q}_r^n$	rotated second-order cone

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Objectives

Part 3 discusses three problems usually employed in the power networks optimization field. The study of these problems is not disconnected from Part 1 and Part 2, since they are polynomial optimization problems and the motivation for including this two chapters in the thesis is for RAPOSa to be able to tackle them in the future. For the time being, we study the properties and solution methods for these problems using an ad-hoc framework designed to solve them. The objectives of the part are:

- Summarize the basic concepts in electrical circuits used subsequently in the definition of the optimization problems.
- Introduce the Optimal Power Flow and the Unit Commitment optimization problems, building blocks of the main subject of study, the Security-Constrained Unit Commitment problem.
- Define the SCUC problem and the formulation employed in our case.
- Describe the framework developed to solve the SCUC problem and obtain solutions with theoretical guarantees of optimality.
- Present some numerical results and the computational setup used.

Note that the notation in this part is not related to the notation used in Part 1 nor Part 2.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

CHAPTER 7

Optimal Power Flow and Unit Commitment

7.1. Introduction

This introductory chapter to the final part of this thesis serves two purposes. The first one is to introduce the necessary concepts taken from the electrical circuits theory to understand the optimization problems used in this part. This introduction to the concepts does not intend to be exhaustive but just give the basic ideas needed to follow the subsequent sections. The second objective is to describe the OPF and the UC optimization problems, which will be used in Chapter 8. Both problems have been extensively studied in the literature, and here we summarize the main ideas we need to build the SCUC problem and a framework to solve it in Chapter 8.

The first of these problems, the Optimal Power Flow or OPF optimization problem, was introduced in [24]. Several formulations exist for this class of problems, but the common objective of all of them is to decide how to optimally operate a power distribution network given the constraints imposed by physics' laws and other limits imposed during the operation. The OPF problem can be adapted to be used for long-term or short-term planning, but regardless of the formulation employed, the use of the power flow equations is what characterizes this problem and also what makes it hard to solve. It is a nonlinear, nonconvex optimization problem, and as mentioned in [80] this problem is NP-hard. There have been numerous works proposing different ways of solving it and obtaining a good solution with optimality guarantees. Part of this research has been oriented to finding good relaxations of the optimization problem. In this chapter we describe the relaxation introduced in [78] which we will employ later in Chapter 8. We also discuss how the sparsity of the problem can be used to simplify this relaxation and consequently improve the solving performance.

The other problem we discuss, the Unit Commitment or UC problem, is also related to the optimization of power systems, but it can be applied in other settings as well since its formulation is more general and does not include specific constraints related with the power systems themselves. This problem was introduced in [51] to schedule the working periods of electric generators. Compared to the OPF problem, the aim here is just to meed demand in the system (regardless of the physical properties of the network) and decide which generators to turn on or off to minimize the cost of operation. The formulation of the problem is straightforward, but its combinatorial nature with the use of binary variables to represent the on-off status of the generators makes this problem also hard to solve. Similarly to the OPF problem, the UC problem is also NP-hard [80]. As the UC problem is an MILP problem, the approach to solve it is a branch-and-bound scheme. There has been extensive research on how to adapt the branch and bound for this particular problem, for example by modifying how branching is done, [100], or adding cuts to the relaxation, [101, 32].

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

The structure of the chapter is as follows. In Section 7.2 we introduce the fundamental electricity concepts we use later on; in Section 7.3 we define the OPF problem and discuss some of the approaches to solving it present in the literature; and, finally, in Section 7.4 we discuss the UC problem.

7.2. Basic electricity concepts

Before continuing with the rest of the chapter, we first need to revise some of the basic concepts in electric circuits. This section builds upon [113, 98, 17, 16, 67, 43].

The first concepts we present are for circuits where *direct current (DC)* is used. If current is the movement of electrons in the circuit, this type of current has all of the electrons moving in the same direction. This is for example the kind of power a battery produces. The main, most basic, electricity concepts are *current* (I), *voltage* (V), *power* (P) and *resistance* (R). The first one, the current, is just the *charge* (Q), i.e., electrons, that crosses some area per unit time. The unit used to measure the charge is the *Coulomb* (C), and the unit for the current is the *Ampere* (A), being $1\,\mathrm{A} = 1\,\mathrm{C/s}$, that is, one Coulomb per second. The current is then:

$$I\ (\text{Amperes}) = \frac{Q\ (\text{Coulombs})}{t\ (\text{seconds})}.$$

The second one is the voltage, related to the concept of *work* (W). The work measures the energy needed to move one object from one point to another by an external *force* (F). The unit for the work is the *joule* (J), the one for the force is the *newton* (N) and the one for the distance is the *meter* (meter). The work is then calculated as:

$$W\ (\text{joules}) = F\ (\text{newtons}) \times s\ (\text{meters}).$$

The voltage is just the work in joules required to move 1 C of charge from one point in a circuit to another. The unit is the *volt* (V),

$$V\ (\text{volts}) = \frac{W\ (\text{joules})}{Q\ (\text{coulombs})}.$$

The power represents the rate at which energy is absorbed or produced in a circuit, measured in *watts* (W). If 1 J of work is absorbed or generated in 1 s time, the resulting power is 1 W:

$$P\ (\text{watts}) = \frac{W\ (\text{joules})}{t\ (\text{seconds})}.$$

Finally, the resistance measures the amount of current that passes through a certain material when a specific voltage is applied. It's measured in *ohms* (Ω), being $1\,\Omega$ the resistance through which a current of 1 A flows when applying a voltage of 1 V. In some conductors, there is a relation between resistance, voltage and current, called the *Ohm's law*:

$$I\ (\text{amperes}) = \frac{V\ (\text{volts})}{R\ (\text{ohms})},$$

written also as

$$V = IR. \tag{17}$$

Apart from the Ohm's law, there is also the *Kirchoff's laws* that play a fundamental role in circuit analysis. There are two laws, one for the voltages and one for the currents. The one for the voltages states that the sum of voltages in a

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

closed loop inside a circuit must be zero. The one for the currents forces the sum of currents in each node of the circuit to be zero. Both of these laws are needed for the definition of the problem in the next chapter, but they won't be used explicitly since they are implied by the construction of the model's parameters.

So far, we have only discussed the most basic concepts for DC power. This is only a starting point, since we now need to look at the related concepts for *alternating current (AC)*, which is the one usually present in power networks and the one we use in this chapter. We follow a standard approach in the analysis of these systems, which consists on using a simplification where the voltage is represented with a sine function and with fixed magnitude, frecuency and phase shift.

In AC power the charge does not always go in the same direction, but changes direction periodically. To represent this varying current, sine functions are used, under the simplification we are assuming:

$$c\sin(2\pi ft+\gamma), \tag{18}$$

where c is the magnitude, f is the frequency of the wave, t is the time and γ is the phase shift. To simplify notation, (18) is usually denoted by $ce^{j\gamma}$, and is called the *phasor* representation, where j denotes the imaginary part. This notation removes the time from the representation, and is therefore changing from the "time domain" to the "frecuency domain". It may also be written as $c\angle\gamma$ in polar coordinates or $a+jb$ in rectangular coordinates, where a and b are computed using Euler's formula.

This representation is then used for the voltage and the current, being the voltage usually represented by $\tilde{V} = V\angle\delta$, being V the peak voltage (magnitude) and δ the phase angle, and the current by $\tilde{I} = I\angle\theta$. Compared to the voltage and current in the DC context, these magnitudes are now varying quantities, represented by a sine function and denoted with the phasor representation. One big difference is the fact that we have changed from real numbers to complex ones. The voltage and the current are now represented by their magnitude and by a phase assigned to them. This forces the introduction of new concepts related to the resistance. We won't talk now of resistance in general, but *impedance*. This new concept represents the entirety of the opposition the elements of a circuit present to the current. It is usually represented by $\tilde{Z} = R+jX$, where R is the resistance (as in the DC world) and X is the *reactance*, representing the extra opposition to the current appearing with the periodic changes the current has in AC. With this, the Ohm's law, that continues to hold in this context, can be rewritten as

$$\tilde{V} = \tilde{I}\tilde{Z}. \tag{19}$$

Another important concept is the *admittance*, which is defined as the reciprocal of the impedance: $\tilde{Y} = 1/\tilde{Z}$. It's also represented in rectangular coordinates by $\tilde{Y} = A + jB$, being A the *conductance* and B the *susceptance*. Then, Ohm's law can also be written as

$$\tilde{Y}\tilde{V} = \tilde{I}. \tag{20}$$

Finally, the last remaining concept is the power. Again, since we are now working with complex numbers, we consider the *complex power* and separate it into two new concepts: *real power* and *reactive power*. Real power represents (as in DC) the work generated or absorbed in a circuit, whereas the reactive power represents circulating energy that doesn't entail an energy transfer or real work in

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

the circuit. Real power only occurs when the voltage and the current are in phase, and reactive power occurs when they are 90° out of phase. In terms of notation, complex power is denoted by S, and defined as

$$S = \tilde{V}\tilde{I}^* = P + jQ, \tag{21}$$

where the operator $\cdot^*$ denotes complex conjugation, P is the real power and Q the reactive power. The magnitude of complex power, $|S|$ is called *apparent power*.

7.3. The Optimal Power Flow problem

We now move to the definition of the first of the optimization problems in this chapter, the *Optimal Power Flow (OPF)* problem. The content in this section is based on [43, 78].

Electric power systems are modelled as a network of electrical buses (nodes) connected by branches (arcs or edges). The branches, i.e., the connections between the buses, are the representation of transmission lines, cables, transformers and other equipment. The objective of the system is to transfer electrical energy from the generation buses (supply) to the load buses (demand). Since we are dealing with a network, each bus is an element of the set of nodes, denoted by N, and each branch belongs to the set of arcs of the network, denoted by L, and is therefore a pair $(i, k) \in L$, with $i, k \in N$. Then, we have an undirected graph (N, L) that represents the network.

Each bus i has a voltage associated with it, $\tilde{V}_i$, which induces current in each branch proportionally to the branch admittance. The equation that models this behavior is the same as (20), but extended to the case of the entire network:

$$\tilde{\boldsymbol{Y}}\tilde{\boldsymbol{V}} = \tilde{\boldsymbol{I}}, \tag{22}$$

where $\tilde{\boldsymbol{V}} = \left(\tilde{V}_1, \ldots, \tilde{V}_{|N|}\right)$ are the phasor voltages of each bus, $\tilde{\boldsymbol{I}} = \left(\tilde{I}_1, \ldots, \tilde{I}_{|N|}\right)$ are the currents injected into the network at each bus, and

$$\tilde{\boldsymbol{Y}} = \begin{pmatrix} \tilde{Y}_{11} & \ldots & \tilde{Y}_{1|N|} \\ \vdots & \ddots & \vdots \\ \tilde{Y}_{|N|1} & \ldots & \tilde{Y}_{|N||N|} \end{pmatrix}$$

is the $|N| \times |N|$ complex bus admittance matrix. This matrix is not usually provided in the data, but needs to be computed for the network. The precise calculations can be seen in [43].

In these kind of systems it is common to work with the power instead of the currents. To do this, Eq. (21) is used and, combined with Eq. (22), we obtain:

$$\boldsymbol{S} = \tilde{\boldsymbol{V}}\tilde{\boldsymbol{I}}^* = \tilde{\boldsymbol{V}} \circ (\tilde{\boldsymbol{Y}}\tilde{\boldsymbol{V}})^*, \tag{23}$$

where $\boldsymbol{S} = \boldsymbol{P} + j\boldsymbol{Q}$ and $\circ$ denotes element-wise multiplication. If for each bus we consider the generation at the bus, S_i^G, and the load, S_i^L, the injected power is the difference between both:

$$S_i = S_i^G - S_i^L,$$

and breaking it down to the real and reactive components,

$$P_i + jQ_i = (P_i^G - P_i^L) + j(Q_i^G - Q_i^L).$$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

So far we have shown how a power system can be modelled with power flow equations. We now want to look at the optimization part. In a system of this

kind, given the admittance matrix $\tilde{\boldsymbol{Y}}$, the voltages for each bus $\tilde{\boldsymbol{V}}$ give all the information needed to represent the power flow in the circuit. That's why the main variables in the optimization are the voltages. The other variables present are the ones controlling the power injection in each bus. Using the phasor representation, each voltage can be represented by a voltage and a phase angle. The voltages are denoted by $\boldsymbol{V}$ and the phase angles by $\boldsymbol{\delta}$, obtaining the following equations for the system:

$$\begin{aligned} P_i(\boldsymbol{V},\boldsymbol{\delta}) &= P_i^G - P_i^L \quad i \in N, \text{ and} \\ Q_i(\boldsymbol{V},\boldsymbol{\delta}) &= Q_i^G - Q_i^L \quad i \in N, \end{aligned} \tag{24}$$

where P_i and Q_i are trigonometric functions of the voltage. One possible form for P_i and Q_i could be

$$P_i(\boldsymbol{V},\boldsymbol{\delta}) = V_i \sum_{k=1}^{N} V_k \left(A_{ik}\cos(\delta_i - \delta_k) + B_{ik}\sin(\delta_i - \delta_k)\right), \text{ and} \tag{25}$$

$$Q_i(\boldsymbol{V},\boldsymbol{\delta}) = V_i \sum_{k=1}^{N} V_k \left(A_{ik}\sin(\delta_i - \delta_k) + B_{ik}\cos(\delta_i - \delta_k)\right), \tag{26}$$

where $\tilde{Y}_{ik} = A_{ik} + jB_{ik}$. The power injection in each bus depends as well on the power generated on all of the other buses, as they are all interconnected. There are alternative representations for these functions, for example using rectangular coordinates as we discuss in Section 7.3.1.

We have then four variables for each bus: V_i, δ_i, P_i and Q_i. In the conventional *Power Flow (PF)* problem, the objective is just to compute values for these variables that are feasible in the system, with no objective function. As we have four variables but only two equations, buses have two of those variables fixed in order to have a deterministic solution. Depending on the variables fixed we can talk of three types of buses: *slack bus*, where the voltage and the phase angle are fixed (this guarantees the existence of a feasible solution); *load bus*, where the power injections P_i and Q_i are fixed; and *voltage-controller bus*, where the real power injection P_i is fixed as well as the voltage magnitude, while the phase angle and the reactive power injection Q_i are not. Thus, every power flow problem of this kind has a feasible solution.

The OPF problem builds on the PF problem by adding an objective function to minimize the cost of generating the required power. For the formulation, a subset of nodes with generation capacity, $G \subseteq N$, is chosen, and for each one of them a cost function is defined, $C_i(P_i^G)$. With this, one possible formulation for the basic OPF problem could be the following:

$$\min_{\boldsymbol{P}^G,\boldsymbol{Q}^G,\boldsymbol{V},\boldsymbol{\delta}} \quad \sum_{i\in G} C_i(P_i^G)$$

$$\text{s.t.} \quad P_i(\boldsymbol{V},\boldsymbol{\delta}) = P_i^G - P_i^L \qquad i \in N \tag{27.1}$$

$$Q_i(\boldsymbol{V},\boldsymbol{\delta}) = Q_i^G - Q_i^L \qquad i \in N \tag{27.2}$$

$$P_i^{G,\min} \le P_i^G \le P_i^{G,\max} \qquad i \in G \tag{27.3}$$

$$Q_i^{G,\min} \le Q_i^G \le Q_i^{G,\max} \qquad i \in G \tag{27.4}$$

$$V_i^{\min} \le V_i \le V_i^{\max} \qquad i \in N \tag{27.5}$$

$$\delta_i^{\min} \le \delta_i \le \delta_i^{\max} \qquad i \in N. \tag{27.6}$$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

7.3.1. Solving the OPF problem. Problem (27) is nonlinear and nonconvex, and therefore hard to solve. As mentioned in [80], the OPF problem is NP-hard. There is abundant literature devoted to proposing different approaches to solve this problem: from simplifications of the model using DC power to specific algorithms designed for this particular problem. More details can be found in the survey [44]. In this thesis we focus our attention on the SDP-based method to obtain a lower bound for problem (27) proposed in [78]. In Chapter 8 we employ this approach as part of the methodology developed there. We now describe the approach in [78].

As discussed avobe, we need to select the form of functions P_i and Q_i in constraints (27.1) and (27.2) respectively. In [78] a formulation using rectangular coordinates is introduced. This formulation requires the definition of several auxiliary matrices, all built from the admittance matrix $\tilde{\boldsymbol{Y}}$ and the matrix $\tilde{\boldsymbol{Y}}^{Sh}$, where $\tilde{Y}_{ij}^{Sh}$ is defined as the value of the shunt element at bus i for branch (i,j) (more details in [78]). For $z \in \mathbb{C}$, $\mathrm{Re}\,(z)$ and $\mathrm{Im}\,(z)$ represent the real and imaginary part respectively, and e_i is the canonical vector in $\mathbb{R}^{|N|}$. The matrices are then:

$$\boldsymbol{Y}_i^G = \boldsymbol{e}_i \boldsymbol{e}_i^T \tilde{\boldsymbol{Y}} \tag{28.1}$$

$$\boldsymbol{E}_i^G = \frac{1}{2}\begin{bmatrix} \mathrm{Re}\left(\boldsymbol{Y}_i^G + (\boldsymbol{Y}_i^G)^T\right) & \mathrm{Im}\left((\boldsymbol{Y}_i^G)^T - \boldsymbol{Y}_i^G\right) \\ \mathrm{Im}\left(\boldsymbol{Y}_i^G - (\boldsymbol{Y}_i^G)^T\right) & \mathrm{Re}\left(\boldsymbol{Y}_i^G + (\boldsymbol{Y}_i^G)^T\right) \end{bmatrix} \tag{28.2}$$

$$\bar{\boldsymbol{E}}_i^G = -\frac{1}{2}\begin{bmatrix} \mathrm{Im}\left(\boldsymbol{Y}_i^G + (\boldsymbol{Y}_i^G)^T\right) & \mathrm{Re}\left(\boldsymbol{Y}_i^G - (\boldsymbol{Y}_i^G)^T\right) \\ \mathrm{Re}\left((\boldsymbol{Y}_i^G)^T - \boldsymbol{Y}_i^G\right) & \mathrm{Im}\left(\boldsymbol{Y}_i^G + (\boldsymbol{Y}_i^G)^T\right) \end{bmatrix} \tag{28.3}$$

$$\boldsymbol{Y}_{ij}^B = \left(\tilde{Y}_{ij}^{Sh} + \tilde{Y}_{ij}\right) \boldsymbol{e}_i \boldsymbol{e}_i^T - \left(\tilde{Y}_{ij}\right) \boldsymbol{e}_i \boldsymbol{e}_j^T \tag{28.4}$$

$$\boldsymbol{E}_{ij}^B = \frac{1}{2}\begin{bmatrix} \mathrm{Re}\left(\boldsymbol{Y}_{ij}^B + (\boldsymbol{Y}_{ij}^B)^T\right) & \mathrm{Im}\left((\boldsymbol{Y}_{ij}^B)^T - \boldsymbol{Y}_{ij}^B\right) \\ \mathrm{Im}\left(\boldsymbol{Y}_{ij}^B - (\boldsymbol{Y}_{ij}^B)^T\right) & \mathrm{Re}\left(\boldsymbol{Y}_{ij}^B + (\boldsymbol{Y}_{ij}^B)^T\right) \end{bmatrix} \tag{28.5}$$

$$\bar{\boldsymbol{E}}_{ij}^B = -\frac{1}{2}\begin{bmatrix} \mathrm{Im}\left(\boldsymbol{Y}_{ij}^B + (\boldsymbol{Y}_{ij}^B)^T\right) & \mathrm{Re}\left(\boldsymbol{Y}_{ij}^B - (\boldsymbol{Y}_{ij}^B)^T\right) \\ \mathrm{Re}\left((\boldsymbol{Y}_{ij}^B)^T - \boldsymbol{Y}_{ij}^B\right) & \mathrm{Im}\left(\boldsymbol{Y}_{ij}^B + (\boldsymbol{Y}_{ij}^B)^T\right) \end{bmatrix} \tag{28.6}$$

$$\boldsymbol{M}_i = \begin{bmatrix} e_i e_i^T & 0 \\ 0 & e_i e_i^T \end{bmatrix}. \tag{28.7}$$

With these auxiliary matrices, one can rewrite problem (27) as described in [78]. The resulting formulation is:

$$\min_{\boldsymbol{P}^G, \boldsymbol{Q}^G, \boldsymbol{X}} \quad \sum_{i \in G} C_i(P_i^G)$$

$$\text{s.t.} \quad P_i^G - P_i^L \geq \mathrm{tr}(\boldsymbol{E}_i^G \boldsymbol{X} (\boldsymbol{X})^T) \qquad i \in N \tag{29.1}$$

$$Q_i^G - Q_i^L \geq \mathrm{tr}(\bar{\boldsymbol{E}}_i^G \boldsymbol{X} (\boldsymbol{X})^T) \qquad i \in N \tag{29.2}$$

$$P_i^{G,\min} \leq P_i^G \leq P_i^{G,\max} \qquad i \in G \tag{29.3}$$

$$Q_i^{G,\min} \leq Q_i^G \leq Q_i^{G,\max} \qquad i \in G \tag{29.4}$$

$$(V_i^{\min})^2 \leq \mathrm{tr}(\boldsymbol{M}_i \boldsymbol{X} (\boldsymbol{X})^T) \leq (V_i^{\max})^2 \qquad i \in N, \tag{29.5}$$

where $\boldsymbol{X} \in \mathbb{R}^{2|N|}$ are new variables introduced and the point $X_i + jX_{i+|N|}$ represents the value $V_i \angle \delta_i$.

To compute a lower bound for problem (29), the authors in [78] build a relaxation of the problem. The first thing is substituting the variables $\boldsymbol{X}$ by new variables $\boldsymbol{W} = \boldsymbol{X}(\boldsymbol{X})^T$. As mentioned in [78], this can be used to obtain an equivalent

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

problem by using conic programming concepts[1], as it's proven that $\boldsymbol{W} = \boldsymbol{X}(\boldsymbol{X})^T$ is equivalent to having $\boldsymbol{W} \in \mathcal{S}_+^{2\cdot|N|}$ and $\text{rank}(\boldsymbol{W}) = 1$. If the constraint $\text{rank}(\boldsymbol{W}) = 1$ is removed, we have a relaxation of problem (29) that is an SDP problem that can be solved to obtain a lower bound for (29).

7.3.2. Exploiting the sparsity. As defined, the SDP constraint $\boldsymbol{W} \in \mathcal{S}_+^{2\cdot|N|}$ scales in size with the increase of the number of nodes in the problem. This complicates its use for large problems. There has been a lot of research in the literature on how to reduce the size of this constraint, for example by breaking the matrix into smaller submatrices in a way that it's sufficient for those matrices to be SDP. The authors in [132] describe one possible method to simplify the SDP constraint called CS-TSSOS, that combines two approaches to decompose the matrix, exploiting the correlative sparsity [129, 128] and the term sparsity [131] at the same time.

Matrix $\boldsymbol{W}$ contains all possible monomials of degree 2 that can appear in problem (29). The *correlative sparsity* method builds a graph, the *correlative sparsity patter (csp) graph* similar to the VIG graph defined in Section 1.5.1, that captures which variables appear together in a monomial in the constraints or in the objective function of the problem. In [132] the authors cite a result that proves the condition $\boldsymbol{W} \in \mathcal{S}_+^{2\cdot|N|}$ can be broken into several constraints for each of the maximal cliques of the csp graph.

Term sparsity is similar to this, but instead of exploiting the interaction between the variables it uses the one between the monomials. In the graph, called in this case the *term sparsity pattern (tsp) graph*, the nodes are the standard monomial basis for the objective function and constraints of problem (29), and two nodes are joined if the product of the two basis elements is a monomial in the problem. Then the same argument with the maximal cliques is used to build a decomposition of matrix $\boldsymbol{W}$.

In the end, both these methods deliver smaller matrices (blocks) that can be used to decompose the SDP constraint. The method CS-TSSOS uses both, first obtaining the cliques of the csp graph and then applying the term sparsity decomposition to each of the remaining blocks to further reduce their size. This needs to be done with care, to maintain the theoretical properties of the decompositions, as described in [132].

7.4. The Unit Commitment problem

We now move to the second of the optimization problems we define in this chapter, that is, the *Unit Commitment Problem* (UC). This is a well known problem in integer programming, being in [51] the first time it was formulated. Since the UC problem is well-known in the field of optimization, we do not enter into as much detail as in the description of the OPF problem, and just present the formulation we use in Chapter 8.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[1]Go back to Section 1.6 for a quick introduction to conic programming.

$$\min_{\boldsymbol{v},\boldsymbol{y},\boldsymbol{z},\boldsymbol{P}^G,\boldsymbol{Q}^G} \quad \sum_{i\in G}\left(\sum_{t\in T}\left(C_i^2(P_{it}^G)^2 + C_i^1 P_{it}^G + C_i^0 v_{it}\right) + \sum_{t\in T\setminus\{0\}}\left(C_i^{SU} y_{it}\right)\right)$$

$$\text{s.t.} \quad v_{i,t-1} - v_{it} + y_{it} - z_{it} = 0 \qquad i\in G, t\in T\setminus\{0\} \tag{30.1}$$

$$P_{it}^G - P_{i,t-1}^G \le R_i^{SU} v_{i,t-1} + P_i^{SU} y_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{30.2}$$

$$P_{i,t-1}^G - P_{it}^G \le R_i^{SD} v_{it} + P_i^{SD} z_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{30.3}$$

$$\sum_{k=t-T_i^U+1,\, k\ge 1}^{t} y_{ik} \le v_{it} \qquad i\in G, t\in T \tag{30.4}$$

$$v_{it} + \sum_{k=t-T_i^D+1,\, k\ge 1}^{t} z_{ik} \le 1 \qquad i\in G, t\in T \tag{30.5}$$

$$P_i^{G,\min} v_{it} \le P_{it}^G \le P_i^{G,\max} v_{it} \qquad i\in G, t\in T \tag{30.6}$$

$$Q_i^{G,\min} v_{it} \le Q_{it}^G \le Q_i^{G,\max} v_{it} \qquad i\in G, t\in T \tag{30.7}$$

$$\sum_{i\in G} P_{it}^G = D_t \qquad t\in T \tag{30.8}$$

$$v_{it}\in\{0,1\},\ y_{it}\in\{0,1\},\ z_{it}\in\{0,1\} \qquad i\in G, t\in T. \tag{30.9}$$

The structure of the problem is the following. Constraint (30.1) represents the core of the problem, since it establishes the relation between variables v (indicates if the generator is turned on), y (indicates wether the generator was turned on in that time period) and z (indicates wether the generator was shut down in that time period). Constraints (30.2) and (30.3) limit the increase and decrease of the power generated in a particular generator from one time period to another, using the startup and shutdown rates, R^{SU} and R^{SD}, and the power generated after being turned on, P^{SU}, and the maximum power that the generator can be generating before being shut down, P^{SD}. Constraints (30.4) and (30.5) put a minimum duration of the on or off status, according to the parameters T^U and T^D. Finally, constraints (30.6) and (30.7) limit the power generated in each generator, as in constraints (27.3) and (27.4), and constraint (30.8) ensures the demand is met.

7.4.1. Solving the UC problem. As already mentioned, this problem is well known in the optimization community, as it's a particularly relevant mixed-integer programming problem. Similarly to the OPF problem, the UC problem is also an NP-hard problem [80]. A lot of research has been done on how to solve this problem efficiently, as one can see for example in the survey [92]. In addition, there is an active effort to develop valid cuts for UC problems to reduce the search space (particularly when using a branch-and-bound scheme to solve it), and also to develop specialized algorithms to solve it, as for example in [100, 101, 88, 32, 73].

In Chapter 8 we don't employ any specific algorithm for solving the UC problem. We just develop a basic branch-and-bound scheme that is used to handle the integer variables in the UC. The only adaptation we make, is the introduction of constraints $y_{it} + z_{it} \le v_{it} + v_{i,t-1}$ and $y_{it} + z_{it} \le 2 - (v_{it} + v_{i,t-1})$. Both of these constraints are cuts that remove unfeasible solutions from the model that can help reduce the search space of the branch and bound scheme.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

CHAPTER 8

Security-Constrained Unit Commitment problem

8.1. Introduction

In Chapter 7 we introduced both the OPF and the UC optimization problems. Both of these problems are widely studied in the literature, and are used to take decisions regarding the operation of power networks. UC for example has been used successfully to obtain significant savings in the operation of a particular power network as can be seen in [23]. However, the UC problem might deliver solutions that can't be used in practice, as they might compromise the stability of the power network. That is the reason behind the interest in adding the OPF constraints to the UC problem, and define the Security-Constrained Unit Commitment problem or SCUC (also called the UC-ACOPF problem). This new problem has the scheduling capability of the UC but at the same time, because it has the power flow equations included, the physical properties in the network are guaranteed for the solution obtained.

The resulting optimization problem combines the constraints from the UC and the OPF problem, being former a combinatorial problem with binary variables and the latter a nonconvex nonlinear problem. Additionally, both problems are NP-hard, as already mentioned in Chapter 7. Therefore, the resulting SCUC optimization problem is a challenging problem to solve, as it combines the complexity from both problems. There has been substancial effort in developing different approaches to obtain feasible solutions for the SCUC problem. See for instance the surveys [102, 11]. Most of the effort has focused on using decomposition methods (such as Benders decomposition [47, 121, 96] or Outer Approximation), Lagrangian relaxations [95, 87] and branch-and-bound methods designed for general MINLP problems. But in general, these proposed solution methods don't guarantee the optimality of the solution obtained, not even local optimality in some cases.

Building upon the SDP relaxation for the OPF problem, discussed in Section 7.3.1, we introduce in this chapter a new framework to solve the SCUC problem with optimality guarantees. We employ the SDP relaxation, adapting it to the SCUC case, and then embed the resulting mixed-integer SDP problem in a branch-and-bound scheme to find an optimal solution to the relaxation. This optimal solution is then used to certify the optimality of a solution to the SCUC problem obtained by the use of a local solver for MINLP problems. We apply this framework to three test cases from the IEEE test system [122] modified as described in [25] to introduce the UC constraints and variables.

The structure of the chapter is as follows. In Section 8.2 we introduce the SCUC problem and present the formulation we use in the rest of the chapter. In Section 8.3 we describe the general framework we propose to solve the SCUC optimization problem, describing the details of the implementation in Section 8.4. Finally, in Section 8.5 we present and analyze the numerical results obtained from

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

solving the aforementioned test cases. This chapter is a joint work with Bissan Ghaddar, Pietro Belotti and Julio González Díaz.

8.2. Security-Constrained Unit Commitment problem

In Chapter 7 we defined the OPF and the UC optimization problems. In this chapter we define the so called *Security-Constrained Unit Commitment (SCUC)* problem, that aims at combining both problems. The idea behind it is the following. We have one problem that can give us the power we need to generate in each node to meet demand and maintain the stability of the network. The other one gives the amount of power and which generators to turn on along some time periods to meet demand. The combination of the two would then give us the amount of power to generate in each time period so we meet demand and at the same time we guarantee the physical stability of the power network. The first time the combination of these two problems was proposed was in [34]. There are multiple formulations for this combination of the problems, but here we present the one we work with:

$$\min_{\boldsymbol{v},\boldsymbol{y},\boldsymbol{z},\boldsymbol{P}^G,\boldsymbol{Q}^G,\boldsymbol{V},\boldsymbol{\delta}} \quad \sum_{i\in G}\left(\sum_{t\in T}\left(C_i^2(P_{it}^G)^2 + C_i^1 P_{it}^G + C_i^0 v_{it}\right) + \sum_{t\in T\setminus\{0\}}\left(C_i^{SU} y_{it}\right)\right)$$

$$\text{s.t.} \quad v_{i,t-1} - v_{it} + y_{it} - z_{it} = 0 \qquad i\in G, t\in T\setminus\{0\} \tag{31.1}$$

$$P_{it}^G - P_{i,t-1}^G \le R_i^{SU} v_{i,t-1} + P_i^{SU} y_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{31.2}$$

$$P_{i,t-1}^G - P_{it}^G \le R_i^{SD} v_{it} + P_i^{SD} z_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{31.3}$$

$$\sum_{k=t-T_i^U+1,\, k\ge 1}^{t} y_{ik} \le v_{it} \qquad i\in G, t\in T \tag{31.4}$$

$$v_{it} + \sum_{k=t-T_i^D+1,\, k\ge 1}^{t} z_{ik} \le 1 \qquad i\in G, t\in T \tag{31.5}$$

$$P_{it}(\boldsymbol{V}_t,\boldsymbol{\delta}_t) = P_{it}^G - P_{it}^L \qquad i\in N, t\in T \tag{31.6}$$

$$Q_{it}(\boldsymbol{V}_t,\boldsymbol{\delta}_t) = Q_{it}^G - Q_{it}^L \qquad i\in N \tag{31.7}$$

$$P_i^{G,\min} v_{it} \le P_{it}^G \le P_i^{G,\max} v_{it} \qquad i\in G, t\in T \tag{31.8}$$

$$Q_i^{G,\min} v_{it} \le P_{it}^G \le Q_i^{G,\max} v_{it} \qquad i\in G, t\in T \tag{31.9}$$

$$V_i^{\min} \le v_{it} \le V_i^{\max} \qquad i\in N, t\in T \tag{31.10}$$

$$\delta_i^{\min} \le \delta_{it} \le \delta_i^{\max} \qquad i\in N, t\in T \tag{31.11}$$

$$I_{ikt}(\boldsymbol{V}_t,\boldsymbol{\delta}_t) \le I_{ik}^{\max} \qquad (i,k)\in L, t\in T \tag{31.12}$$

$$v_{it}\in\{0,1\},\ y_{it}\in\{0,\,1\},\ z_{it}\in\{0,1\} \qquad i\in G, t\in T. \tag{31.13}$$

The structure of the problem is straightforward: we combine the variables and constraints from both (27) and (30). The main difference here is that, for every time period in the UC problem, we now have an entire OPF problem associated with it. This makes tackling this problem a daunting task. Notice that constraint (30.8) has been removed, as constraint (31.6) forces the generator to produce enough power to meet the demand for each time period. Also, constraint (31.12) was added to limit the current on each transmission line, where L represents the set of links between the nodes in N as defined in Section 7.3.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Following the discussion in Section 7.3.1 and Section 7.4.1, we reformulate problem (31). The constraints associated with the UC problem stay the same (that is, constraints (31.1), (31.2), (31.3), (31.4), (31.5), (31.8) and (31.9)), and constraints (32.16) and (32.17) are added. The ones associated with the OPF problem change as follows. New variables are introduced, $\boldsymbol{X}_t \in \mathbb{R}^{2|N|}$, where the point $X_{it} + jX_{i+|N|t}$ represents the value $V_{it}\angle\delta_{it}$. In constraints (31.6) and (31.7), the functions $P_{it}(\boldsymbol{V}_t, \boldsymbol{\delta}_t)$ and $Q_{it}(\boldsymbol{V}_t, \boldsymbol{\delta}_t)$ are substituted by $\text{tr}(\boldsymbol{E}_i^G \boldsymbol{X}_t(\boldsymbol{X}_t)^T)$ and $\text{tr}(\bar{\boldsymbol{E}}_i^G \boldsymbol{X}_t(\boldsymbol{X}_t)^T)$ respectively. Constraints (31.10) and (31.11) are substituted by one constraint, $(V_i^{\min})^2 \leq \text{tr}(\boldsymbol{M}_i \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \leq (V_i^{\max})^2$. And constraint (31.12) is now written as $(\tilde{P}_{ijt})^2 + (\tilde{Q}_{ijt})^2 \leq (S_{ij}^{\max})^2$, where $\tilde{P}_{ijt}$ and $\tilde{Q}_{ijt}$ are defined as $\text{tr}(\boldsymbol{E}_{ij}^B \boldsymbol{X}_t(\boldsymbol{X}_t)^T)$ and $\text{tr}(\bar{\boldsymbol{E}}_{ij}^B \boldsymbol{X}_t(\boldsymbol{X}_t)^T)$, respectively. This is not the same constraint, as it's now written in terms of power instead of current. But, as mentioned in [43], the simplification is sufficiently accurate in most cases. The resulting model is the following:

$$\min_{\boldsymbol{v},\boldsymbol{y},\boldsymbol{z},\boldsymbol{P}^G,\boldsymbol{Q}^G,\boldsymbol{X}} \quad \sum_{i\in G}\left(\sum_{t\in T}\left(C_i^2(P_{it}^G)^2 + C_i^1 P_{it}^G + C_i^0 v_{it}\right) + \sum_{t\in T\setminus\{0\}}\left(C_i^{SU} y_{it}\right)\right)$$

$$\text{s.t.} \quad v_{i,t-1} - v_{it} + y_{it} - z_{it} = 0 \qquad i \in G, t \in T\setminus\{0\} \tag{32.1}$$

$$P_{it}^G - P_{i,t-1}^G \leq R_i^{SU} v_{i,t-1} + P_i^{SU} y_{it} \qquad i \in G, t \in T\setminus\{0\} \tag{32.2}$$

$$P_{i,t-1}^G - P_{it}^G \leq R_i^{SD} v_{it} + P_i^{SD} z_{it} \qquad i \in G, t \in T\setminus\{0\} \tag{32.3}$$

$$\sum_{k=t-T_i^U+1,\, k\geq 1}^{t} y_{ik} \leq v_{it} \qquad i \in G, t \in T \tag{32.4}$$

$$v_{it} + \sum_{k=t-T_i^D+1,\, k\geq 1}^{t} z_{ik} \leq 1 \qquad i \in G, t \in T \tag{32.5}$$

$$P_{it} - P_{it}^L \geq \text{tr}(\boldsymbol{E}_i^G \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \qquad i \in N, t \in T \tag{32.6}$$

$$Q_{it} - Q_{it}^L \geq \text{tr}(\bar{\boldsymbol{E}}_i^G \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \qquad i \in N, t \in T \tag{32.7}$$

$$P_i^{\min} v_{it} \leq P_{it}^G \leq P_i^{\max} v_{it} \qquad i \in G, t \in T \tag{32.8}$$

$$Q_i^{\min} v_{it} \leq Q_{it}^G \leq Q_i^{\max} v_{it} \qquad i \in G, t \in T \tag{32.9}$$

$$(V_i^{\min})^2 \leq \text{tr}(\boldsymbol{M}_i \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \leq (V_i^{\max})^2 \qquad i \in N, t \in T \tag{32.10}$$

$$\tilde{P}_{ijt} = \text{tr}(\boldsymbol{E}_{ij}^B \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \qquad (i,j) \in L, t \in T \tag{32.11}$$

$$\tilde{Q}_{ijt} = \text{tr}(\bar{\boldsymbol{E}}_{ij}^B \boldsymbol{X}_t(\boldsymbol{X}_t)^T) \qquad (i,j) \in L, t \in T \tag{32.12}$$

$$(\tilde{P}_{ijt})^2 + (\tilde{Q}_{ijt})^2 \leq (S_{ij}^{\max})^2 \qquad (i,j) \in L, t \in T \tag{32.13}$$

$$P_{it} = 0 + \sum_{k\in H_i} P_{kt}^G \qquad i \in N, t \in T \tag{32.14}$$

$$Q_{it} = 0 + \sum_{k\in H_i} Q_{kt}^G \qquad i \in N, t \in T \tag{32.15}$$

$$y_{it} + z_{it} \leq v_{it} + v_{i,t-1} \qquad i \in G, t \in T\setminus\{0\} \tag{32.16}$$

$$y_{it} + z_{it} \leq 2 - (v_{it} + v_{i,t-1}) \qquad i \in G, t \in T\setminus\{0\} \tag{32.17}$$

$$v_{it} \in \{0,1\},\ y_{it} \in \{0,\ 1\}, z_{it} \in \{0,1\} \qquad i \in G, t \in T. \tag{32.18}$$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

8.3. Solving the SCUC

After defining the formulation we use, the rest of the chapter is aimed at finding a solution for the problem. We are dealing with a mixed-integer nonlinear nonconvex optimization problem, as we have binary variables from the unit commitment part, a quadratic objective function and nonlinear constraints. In Section 7.3.1 we mentioned how the authors in [78] build an SDP relaxation for the OPF problem. This relaxation is an SDP problem with integer variables, and has been studied before for the OPF problem, where it's shown that it delivers good lower bounds [78]. Applying the same procedure to problem (32), we obtain an SDP constraint for each time period, and therefore also a relaxation. The resulting formulation is as follows:

$$\min_{\boldsymbol{v},\boldsymbol{y},\boldsymbol{z},\boldsymbol{P}^G,\boldsymbol{Q}^G,\boldsymbol{W}} \quad \sum_{i\in G}\left(\sum_{t\in T}\left(C_i^2(P_{it}^G)^2 + C_i^1 P_{it}^G + C_i^0 v_{it}\right) + \sum_{t\in T\setminus\{0\}}\left(C_i^{SU} y_{it}\right)\right)$$

$$\text{s.t.} \quad v_{i,t-1} - v_{it} + y_{it} - z_{it} = 0 \qquad i\in G, t\in T\setminus\{0\} \tag{33.1}$$

$$P_{it}^G - P_{i,t-1}^G \le R_i^{SU} v_{i,t-1} + P_i^{SU} y_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{33.2}$$

$$P_{i,t-1}^G - P_{it}^G \le R_i^{SD} v_{it} + P_i^{SD} z_{it} \qquad i\in G, t\in T\setminus\{0\} \tag{33.3}$$

$$\sum_{k=t-T_i^U+1,\, k\ge 1}^{t} y_{ik} \le v_{it} \qquad i\in G, t\in T \tag{33.4}$$

$$v_{it} + \sum_{k=t-T_i^D+1,\, k\ge 1}^{t} z_{ik} \le 1 \qquad i\in G, t\in T \tag{33.5}$$

$$P_{it} - P_{it}^L \ge \operatorname{tr}(\boldsymbol{E}_i^G \boldsymbol{W}_t) \qquad i\in N, t\in T \tag{33.6}$$

$$Q_{it} - Q_{it}^L \ge \operatorname{tr}(\bar{\boldsymbol{E}}_i^G \boldsymbol{W}_t) \qquad i\in N, t\in T \tag{33.7}$$

$$P_i^{\min} v_{it} \le P_{it}^G \le P_i^{\max} v_{it} \qquad i\in G, t\in T \tag{33.8}$$

$$Q_i^{\min} v_{it} \le Q_{it}^G \le Q_i^{\max} v_{it} \qquad i\in G, t\in T \tag{33.9}$$

$$(V_i^{\min})^2 \le \operatorname{tr}(\boldsymbol{M}_i \boldsymbol{W}_t) \le (V_i^{\max})^2 \qquad i\in N, t\in T \tag{33.10}$$

$$\tilde{P}_{ijt} = \operatorname{tr}(\boldsymbol{E}_{ij}^B \boldsymbol{W}_t) \qquad (i,j)\in L, t\in T \tag{33.11}$$

$$\tilde{Q}_{ijt} = \operatorname{tr}(\bar{\boldsymbol{E}}_{ij}^B \boldsymbol{W}_t) \qquad (i,j)\in L, t\in T \tag{33.12}$$

$$(\tilde{P}_{ijt})^2 + (\tilde{Q}_{ijt})^2 \le (S_{ij}^{\max})^2 \qquad (i,j)\in L, t\in T \tag{33.13}$$

$$P_{it} = 0 + \sum_{k\in H_i} P_{kt}^G \qquad i\in N, t\in T \tag{33.14}$$

$$Q_{it} = 0 + \sum_{k\in H_i} Q_{kt}^G \qquad i\in N, t\in T \tag{33.15}$$

$$y_{it} + z_{it} \le v_{it} + v_{i,t-1} \qquad i\in G, t\in T\setminus\{0\} \tag{33.16}$$

$$y_{it} + z_{it} \le 2 - (v_{it} + v_{i,t-1}) \qquad i\in G, t\in T\setminus\{0\} \tag{33.17}$$

$$\boldsymbol{W}_t \in \mathcal{S}_+^{2\cdot|N|} \qquad t\in T \tag{33.18}$$

$$v_{it}\in\{0,1\},\ y_{it}\in\{0,1\},\ z_{it}\in\{0,1\} \qquad i\in G, t\in T. \tag{33.19}$$

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

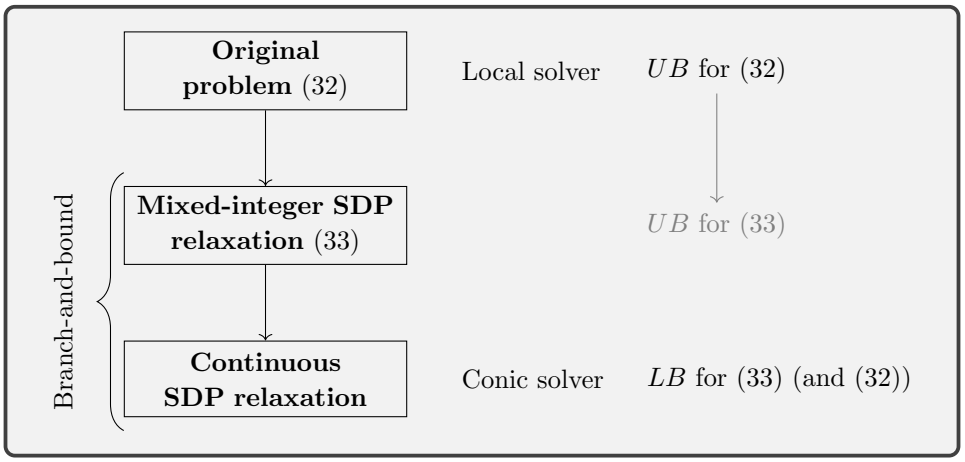


FIGURE 8.1. Diagram of the framework to solve (32).

The next step would be to find a solver that could solve mixed-integer SDP problems, but there is none available commercially. For that reason and as already mentioned in Section 7.4.1, we use a basic branch-and-bound scheme to find an integer solution to this SDP relaxation, solving in each node an SDP problem. More details on the implementation can be found in Section 8.4. The resulting framework to solve problem (32) is described in Figure 8.1.

As described in Figure 8.1, we have the original problem (32) which we can solve using a local solver in order to find a feasible solution (upper bound) for the problem. From that solution, we can find an upper bound for the SDP relaxation (33), just by fixing the integer variables to the solution of problem (32) and solving the SDP problem using the conic solver. This might not be feasible, but in case it is we would have an upper bound for the branch-and-bound scheme. Finally, relaxing the integrality condition on the SDP relaxation (33) we have the "relaxation" used in the branch-and-bound. This branch-and-bound gives us also a lower bound for problem (33), and since it's also a relaxation of problem (32), we have a lower bound for it as well. Summarizing, using these three problems we are able to find an upper bound for problem (32), through the use of the local solver, and employing the branch-and-bound certify its optimality by finding a lower bound.

There is a crucial point in this framework. The quality of the relaxation (33) is relevant as it puts a limit on how much closer the upper and lower bound can get. We will never find a lower bound better than the optimal solution of problem (33), which means that if that optimal is "far" from the optimum for problem (32), the gap between the lower and the upper bound would not close enough.

8.4. Computational setup

After the description of the elements we need in order to solve the problem, we are now in situation to start describing the computational setup we use. We begin with the three instances we solve. We take the instances `case6ww`, `case24_ieee_rst` (which we call `case24` to abbreviate) and `case118` from MATPOWER [134] on top of which we add the unit commitment data from [25]. Because the underlying OPF data from [25] is not the same as the MATPOWER one, we

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

have modified it to obtain feasible and sufficiently interesting cases for our numerical analysis.

Using the data, we define all of the parameters of the optimization problem. Additionally, we also fix the on-off status for some of the generators, for the first time period (time period 0). For those generators, we fix their associated v variable to 1, and we also fix the power they generate in that particular time period.

Every one of the three cases has information for 24 time periods. To obtain more cases on which to test our solving environment, we have restricted the cases to 4 and 12 time periods as well, just to get easier problems to compare with and check how our approach scales not only with the number of nodes but also with the increase in time periods. Using the data for the 24 time period case but with fewer time periods makes some of the decisions in the problem irrelevant. For example, if one has that one generator needs to be on during the first time period, and it cannot shut down for 12 time periods (constraint (31.4)), this implies that when only considering 6 time periods, the variable v must be 1 for all of them. This is something to have in mind when comparing the performance of the algorithm on the different number of time periods considered.

Another important thing to have, as described in Section 7.3.2, is the decomposition of the matrix in the SDP relaxation. As mentioned in that section, we use the Julia package from [132] to compute the corresponding decomposition. To do that, we first need to write our problem in Julia using the JuMP language [86] then we can pass the problem to the function that computes the decomposition. Once that is done, we generate several files that contain the decomposition, which we later read when running the algorithm.

Once we have the data that defines the problems, we can start looking at the solution methods. We need to describe the three things that conform our implementation: the local solver, the SDP solver and the branch-and-bound scheme to deal with the binary variables. We start with the local solver. In this work we are using Knitro [22] as the local solver. To interface with it, we use Pyomo [21, 65] (the entire algorithm is implemented in Python [42]). We are using the default options for Knitro, except for the options `mip_multistart` which we fix to 1; `convex`, which we fix to 0; and `mip_terminate` which we fix to 1. This last option is interesting because each time we call Knitro in a node, we just want for it to deliver a feasible solution and not to spend more time trying to improve it. Just in case it's not able to find one in a reasonable amount of time we also limit the time with option `mip_maxtime_real`. The problem we pass to Knitro is problem (32), as we are utilizing the capabilities Knitro has to obtain solutions for general MINLP problems. The second element we use is the SDP solver. In our case, we use Mosek [93] as a conic solver. To interface with it we use their API for Python called Fusion. It arms the developer with tools to formulate an optimization problem to solve directly with Mosek and include SOCP and SDP constraints.

The employment of SDP constraints in the relaxation and our selection of solvers has several drawbacks. The first one is that we need to reformulate constraint (32.13) and the objective function and write them as SOCP constraints because it is the only way Mosek accepts the use of quadratic constraints. Using what we discussed in Section 1.6.2, and introducing auxiliary variables x and y,

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

constraint (33.13) is now written as

$$(1/2, (S_{ij}^{max})^2, \tilde{P}_{ijt}, \tilde{Q}_{ijt})^T \in \mathcal{Q}_r^4,$$

the objective function as

$$\min \quad \sum_{i\in G} \Big(\sum_{t\in T} \big(y + C_i^1 P_{it}^G + C_i^0 v_{it}\big) + \sum_{t\in T\setminus\{0\}} \big(C_i^{SU} y_{it}\big) \Big),$$

and new constraints $\boldsymbol{x} = (\boldsymbol{p}^G)^T$ and $(1, y, \boldsymbol{F}\boldsymbol{x}) \in \mathcal{Q}_r^{|G|\cdot|T|+2}$ are added to obtain an equivalent problem.

The second one is that we need to code the problem twice in Python, once for the local solver Knitro using Pyomo, and another for the SDP solver Mosek using the Fusion API. This is because Pyomo still has a relatively incomplete support of conic constraints. This "double coding" of the problem is a source of inconsistencies with the algorithm that has slowed the development of the project noticeably.

Finally, because Mosek does not support integer variables when SDP constraints are included in the problem, and there are no other solvers available that can deal both with SDP and integer variables, we made our own implementation of such a "solver". Using a naive implementation of branch-and-bound in Python, we adapted it to solve SDP problems with integer variables, and in each node we solve the resulting continuous SDP problem with Mosek.

With these three ingredients, we have our complete environment on which to test our approach to solving the SUCU problem. Just to summarize all of the elements and their interaction:

- We obtain the data from MATPOWER [134] using the unit commitment data from [25]. We modify it and then we use the CS-TSSOS package from [132] to compute the decomposition for the SDP constraint.
- We start with problem (32), which we code into Pyomo so Knitro can solve it. At the same time, we code the relaxation (33) using Mosek's Fusion API. Here we read the decomposition of the SDP matrix and add the resulting constraints. We start the branch-and-bound process.
- A value LS is fixed, and during the branch-and-bound, in each node, we run the local solver if the iteration i satisfies that $i \geq LS^l$, being l a counter for the calls to the local solver. This means that the local solver is run more often during the first iterations (when it's urgent to find an upper bound since probably there is not one available yet) than in later iterations. The selection of LS is crucial, as we will see in Section 8.5, specially for the larger cases.
- Each time the local solver is called, the bounds for the binary variables are sent to it (i.e. the branching that has already been done to define the corresponding node). This way, we explore different parts of the feasible region each time we call the solver and therefore it starts from different points increasing the probability of obtaining different feasible solutions and hopefully one closer to the global optimum of the problem.
- Once the local solver gives a solution back, we fix the binary variables to the solution obtained and run the SDP solver to get an upper bound for the SDP problem. Notice here that the local solver does give an upper bound, but for the original problem, and running the SDP solver can

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

improve upon the value obtained for the original problem to obtain a more tight upper bound for the relaxation.

- As it's usually done in branch-and-bound schemes, we stop the process when the time limit is reached or when the optimality gap (defined as $(UB - LB)/(LB + 10^{-9})$) is less than a predefined tolerance of 0.02.

We have finished the description of our implementation, but before continuing to analyze the numerical results in Section 8.5, just comment on something relevant. Solving an SDP problem is less robust than solving a linear problem. The solver is really affected by differences in scale between the constraints and the objective function (we rescale the objective function in our case to avoid numerical issues) and this means that sometimes the lower bound in the branch-and-bound does not increase monotonically and decreases from one node to the next, as we have observed in our computational study.

8.5. Numerical results

In this section we analyze the numerical results obtained for the three cases mentioned: `case6ww`, `case24`, `case118`. As described in Section 8.4 we run each one for time periods 4, 12 and 24, which allows to check the changes in performance as the size of the problem increases. Also, we run the cases for different values of LS. In Table 8.1 we report the results for the best value of LS in each case. The values represented are the lower bound, the upper bound (of the SDP relaxation), the "real upper bound" (of the original problem), the corresponding gaps and the total running time in seconds. The time limit for the executions was one hour. Everything was run on CESGA on nodes power with 2 twenty-four-core Intel Xeon Gold 6240R CPUs with 180GB of RAM.

TABLE 8.1. Performance of the algorithm on different test cases.

Case	T	LS	LB	UB	Gap	Real UB	Real gap	Time
`case6ww`	4	1.1	11 295	11 441	0.013	11 441	0.013	14
`case6ww`	12	1.1	34 664	35 343	0.020	35 343	0.020	86
`case6ww`	24	1.1	75 701	77 192	0.020	77 192	0.020	210
`case24`	4	1.1	67 455	69 796	0.035	69 796	0.035	3600
`case24`	12	1.1	340 634	351 501	0.032	351 501	0.032	3600
`case24`	24	**1.3**	879 671	904 750	0.029	904 750	0.029	3600
`case118`	4	1.1	331 636	339 982	0.025	339 982	0.025	3600
`case118`	12	1.1	1 001 967	1 086 367	0.084	1 086 366	0.084	3600
`case118`	24	**1.5**	2 223 577	2 349 116	0.057	2 349 118	0.057	3600

Overall, the performance is good, as a gap less than 4% was obtained for cases `case6ww` and `case24` with the three selections of time periods. In the most difficult case, `case118`, the performance is not as good, as the 12 and 24 time periods cases have a bigger gap of 8% and 6% respectively. Nonetheless, for `case6ww` the solving time is really small as it reached a relative gap of 0.02 which is the tolerance set as the stopping criteria. In `case24` and `case118`, the algorithm ran until the time limit was reached.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Another thing to notice is that the UB and the Real UB are the same in all cases (there are differences in the decimal places which do not show up on the

table). This implies that the SDP relaxation build by using the approach described in Section 8.3, that employs a relaxation build for OPF problems and applies it at every time period, works well and delivers tight bounds for the problem. Also, the impact of increasing from 4 to 12 and then to 24 time periods doesn't seem to have a big impact in the performance.

Finally, the value of LS taken for almost all cases is 1.1. Except for the two highlighted ones, in the other cases the performance was really similar between different values of LS. But in the two highlighted cases, there was a big change between different values as we can see in Table 8.2 and Table 8.3. The values represented there are the lower and upper bound, the relative gap of the SDP, the percentage of time over the total time employed that the local solver used (% t LS), the percentage of nodes on which the local solver run (% n LS), the iteration where the last "jump" (improvement of the upper bound) was made (LJ), the total number of "jumps" that happened in the algorithm (TJ) and the total number of iterations done by the algorithm (i).

TABLE 8.2. Impact of LS in case `case24`.

Case	T	LS	LB	UB	Gap	% t LS	% n LS	LJ	TJ	i
case24	24	2	881 115	1 186 654	0.347	47%	1%	256	4	1024
case24	24	1.75	881 235	1 239 901	0.407	37%	1%	7	3	1230
case24	24	1.5	880 467	1 198 543	0.361	79%	4%	58	4	417
case24	24	1.3	879 671	904 750	0.029	94%	17%	40	6	115
case24	24	1.1	879 342	907 054	0.032	98%	51%	41	6	63

TABLE 8.3. Impact of LS in case `case118`.

Case	T	LS	LB	UB	Gap	% t LS	% n LS	LJ	TJ	i
case118	24	2	2 223 812	2 708 206	0.218	67%	7%	2	2	98
case118	24	1.75	2 223 638	2 708 206	0.218	76%	11%	2	2	70
case118	24	1.5	2 223 577	2 349 116	0.057	79%	19%	12	3	57
case118	24	1.3	2 221 798	2 708 206	0.219	93%	67%	2	2	17
case118	24	1.1	2 221 616	2 708 206	0.219	94%	100%	2	2	11

The impact of the selection of LS is really noticeably in these cases. For example, we see that between the 5 selections of LS value in Table 8.2, only two of them manage to get the gap below 30% and the gap obtained is noticeable smaller than in the other options. And in Table 8.3 only one gets it below 20%. The explanation for this is that the nodes on which the local solver is run are different for different values of LS. That way, if there is a subregion of the feasible region that contains the global optimum and in that particular region other feasible points are not present (or are not found by the local solver because of the starting point for instance), then the local solver would find that point but in other regions would not. This means that a bit of "luck" is present in the performance of the approach, as choosing the value of LS correctly can mean the difference between reaching a reasonable relative gap or not.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Another thing to notice is that when the local solver is called less, as the local solver employs much of the total time of the algorithm, the number of iterations that can be done increases noticeably, and therefore the lower bound reached in the same amount of time is better. If this improvement on the lower bound could outperform the improvement "by luck" of the upper bound, the impact of LS would be less. But that is not the case in our results.

8.6. Conclusions

The numerical results in Section 8.5 are promising, as we are able to find a gap below 10% in one hour running time. Nonetheless, there is still room for improvement in the algorithm. For instance, proposals from the literature such as specific cuts for the UC problem can be added. Also, in order to strengthen the results, a richer pool of instances should be considered. As future work, we want to generate more instances by modifying the data used to build the three used in this chapter, but also experiment with larger instances from the IEEE test set adding the necessary UC information to them.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Conclusions

In Chapter 1 we have introduced the basic concepts of the RLT technique. In Chapter 2 we have studied the impact of several approaches to enhance the efficacy of domain reduction functionalities within the specific context of the RLT technique for polynomial optimization problems. The presented results showed that, although all these enhancements lead to performance improvements, the overall impact may sometimes be relatively small. Interestingly, the largest impact comes from the approaches to select the branching point which are, by far, the simplest and easiest to implement among all the enhancements under consideration. We believe these kind of insights on the trade-offs between performance improvement and implementation costs are very relevant for the design of more efficient optimization solvers, since they can guide the implementation efforts of the developing teams of state-of-the-art solvers towards the most promising directions. When a new technique or algorithmic enhancement is first studied, it is natural to analyze is impact in isolation, in order to better quantify its impact. Yet, because of this isolated analyses, it is often hard to assess which of a series of potential enhancements should be prioritized when designing a new solver or when evolving an already existing one, since the impacts individually reported for each of them might no be easy to compare to one another.

In Chapter 3 we have not been able to find any compelling rationale behind the results in Table 3.1 and Table 3.2. Yet, given the size of the variability in the different measures of performance, it definitely seems important to develop additional studies to gain some understanding on the observed behavior. These additional studies might even go beyond the RLT technique and polynomial optimization problems, and study whether similar effects might arise for other classes of problems and branch-and-bound schemes. In Section 5.5 we try to shed some light on this by applying ML to try and anticipate which of the approaches works best for a particular problem. Analyzing the resulting models could allow the extraction of information to shed some light on what makes the performance of either of the approaches so different for different solvers and problems.

In Chapter 4 we have introduced an extension of `plainRLT` to deal with variables with infinte upper bounds. We also proved the result guaranteeing the optimality of the result obtained by the algorithm, in case of convergence, and discussed the impact other enhancements could have on this convergence. Lastly, we presented some preliminary numerical results that showcase the implementation works but there is still work to be done to obtain a competitive implementation.

As future work, the refinement of the implementation, for example re-adding the "warm start" generation of the bound-factor constraints or refining the selection of the branching point is a natural next step. Also running the algorithm on more test problems from different test sets such as QPLIB [49]. But, more importantly,

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

an effort must be done to implement some process to bound the linear relaxation for those problems where it's unbounded. Doing this is crucial to be able to solve more problems where the base linear relaxation may not be bounded, but adding some appropriate cuts, for example some conic programming-based ones, or using probing to discard the parts of the feasible region that lead to the unboundedness because they are infeasible in the original problem, leads to a bounded relaxation.

In Chapter 5 we have described the learning framework from [54, 58], including the KPI used, the explanatory variables considered and the learning technique chosen. This learning technique defines a general framework that can be used for example to select between different configurations of an optimization algorithm (as the RLT technique), but also between different solvers for the same type problems, as long as an adequate KPI is defined and appropriate explanatory variables selected. In this chapter we have shown the robustness of the technique, applying it to the results from the different chapters in Part 1. As general as this framework is, adapting it to the different contexts on which it's applied could deliver even better results.

Chapter 6 reveals interesting insights into the effectiveness of different branching rules and variable selection strategies for general nonlinear programming problems when using a spatial branch-and-bound algorithm. The analysis was done on RLT for polynomial programming problems however everything can be extended to general nonlinear programming. In particular, we analyze three different experts that learning approaches can try to imitate and show the limitations of two of them ($\texttt{BVar}^{\texttt{D}}$ and $\texttt{BRule}^{\texttt{D}}$), which rely on dynamic features compared to the one that is based on static features ($\texttt{ORule}^{\texttt{S}}$). These limitations primarily stem from the shortsighted decisions made using local information from the branch-and-bound tree. The dynamic use of features, while seemingly adaptive, often fails to provide long-term benefits, rendering it less effective against the $\texttt{ORule}^{\texttt{S}}$ expert. This underscores the challenge of relying too heavily on information that only pertains to the current state of the tree, without considering broader implications.

Among the three experts, $\texttt{ORule}^{\texttt{S}}$ outperforms the others while $\texttt{BRule}^{\texttt{D}}$, which relies on dynamic branching rule selection, performs less effectively because of the myopic nature of the KPI it uses which sets a limit on the attainable outcomes within the dynamic rule framework. Meanwhile, $\texttt{BVar}^{\texttt{D}}$ is the least effective of the three. Although $\texttt{BRule}^{\texttt{D}}$ might not always choose the best immediate options, it benefits from incorporating insights into the branching process' progression. This allows $\texttt{BRule}^{\texttt{D}}$ to exclude certain variables that $\texttt{BVar}^{\texttt{D}}$ might select based on their immediate, myopic, KPI focused solely on bound improvement.

The underperformance of $\texttt{BVar}^{\texttt{D}}$, which attempts to mimic strong branching, suggests that future efforts in developing branching rules for spatial branching in the context of nonlinear optimization should perhaps not focus solely on approximating strong branching. Instead, these efforts might aim to explore new directions such as extending reinforcement learning to spatial branching which can potentially lead to more effective branching strategies in complex optimization environments. Other directions worth exploring are finding a better KPI to learn in the dynamic setting and enriching the portfolio of branching rules in the static learning setting.

In hapter 7 we have described the fundamental elements of the Optimal Power Flow and the Unit Commitment problems, that allow to define in Chapter 8 the Security-Constrained Unit Commitment problem that we deal with. The numerical

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

results in Section 8.5 are promising, as we are able to find a gap below 10% in one hour running time. Nonetheless, there is still room for improvement in the algorithm. For instance, proposals from the literature such as specific cuts for the UC problem can be added. Also, in order to strengthen the results, a richer pool of instances should be considered. As future work, we want to generate more instances by modifying the data used to build the three used in this chapter, but also experiment with larger instances from the IEEE test set adding the necessary UC information to them.

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

References

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. "Branching rules revisited". In: *Operations Research Letters* 33.1 (2005), pp. 42–54. DOI: 10.1016/j.orl.2004.04.002.

[2] Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. "A Machine Learning-Based Approximation of Strong Branching". In: *INFORMS Journal on Computing* 29.1 (2017), pp. 185–195. DOI: 10.1287/ijoc.2016.0723.

[3] Maria-Florina Balcan et al. "Learning to Branch". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 344–353.

[4] Radu Baltean-Lugojan et al. "Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks". https://optimization-online.org/?p=17362. 2019.

[5] Pietro Belotti. "Bound reduction using pairs of linear inequalities". In: *Journal of Global Optimization* 56.3 (2013), pp. 787–819. DOI: 10.1007/s10898-012-9848-9.

[6] Pietro Belotti et al. *Branching and bounds tightening techniques for non-convex MINLP*. Tech. rep. Optimization-online 2059. IBM Research Report RC24620, 2008.

[7] Pietro Belotti et al. "Branching and bounds tightening techniques for non-convex MINLP". In: *Optimization Methods and Software* 24.4-5 (2009), pp. 597–634. DOI: 10.1080/10556780903087124.

[8] Pietro Belotti et al. "On feasibility based bounds tightening". https://enac.hal.science/hal-00935464. 2012.

[9] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[10] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. 2021. arXiv: 2112.08872 [math.OC]. URL: https://arxiv.org/abs/2112.08872.

[11] Amit Bhardwaj et al. "Unit commitment in electrical power system-a literature review". In: *Unit commitment in electrical power system-a literature review*. IEEE, 2012, p. 30.

[12] Daniel Bienstock and Gonzalo Muñoz. "LP Formulations for Polynomial Optimization Problems". In: *SIAM Journal on Optimization* 28.2 (2018), pp. 1121–1150. DOI: 10.1137/15M1054079.

[13] Hans L. Bodlaender and Arie M.C.A. Koster. "Treewidth computations I. Upper bounds". In: *Information and Computation* 208.3 (2010), pp. 259–275.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[14] Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. "A classifier to decide on the linearization of mixed-integer quadratic problems in CPLEX". In: *Operations Research* (2022).

[15] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[16] Britannica Academic. "Electrical impedance". In: *Encyclopædia Britannica*. https://academic.eb.com/levels/collegiate/article/electrical-impedance/32292. Accessed 1 Apr. 2024. 1998.

[17] Britannica Academic. "Reactance". In: *Encyclopædia Britannica*. https://academic.eb.com/levels/collegiate/article/reactance/62852. Accessed 1 Apr. 2024. 1998.

[18] Christoph Buchheim and Angelika Wiegele. "Semidefinite relaxations for non-convex quadratic mixed-integer programming". In: *Mathematical Programming* 141.1-2 (2012), pp. 435–452. DOI: 10.1007/s10107-012-0534-y.

[19] Samuel Burer and Dieter Vandenbussche. "A finite branch-and-bound algorithm for nonconvex quadratic programming via semidefinite relaxations". In: *Mathematical Programming* 113.2 (2006), pp. 259–282. DOI: 10.1007/s10107-006-0080-6.

[20] Michael R. Bussieck, Arne Stolbjerg Drud, and Alexander Meeraus. "MINLPLib-A Collection of Test Models for Mixed-Integer Nonlinear Programming". In: *INFORMS Journal on Computing* 15 (2003), pp. 114–119.

[21] Michael L. Bynum et al. *Pyomo–optimization modeling in python*. Third. Vol. 67. Springer Science & Business Media, 2021.

[22] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. "Knitro: An Integrated Package for Nonlinear Optimization". In: *Large-Scale Nonlinear Optimization*. Ed. by G. Di Pillo and M. Roma. Boston, MA: Springer US, 2006, pp. 35–59. DOI: 10.1007/0-387-30065-1_4.

[23] Brian Carlson et al. "MISO Unlocks Billions in Savings Through the Application of Operations Research for Energy and Ancillary Services Markets". In: *Interfaces* 42.1 (2012), pp. 58–73.

[24] J. Carpentier. "Contribution á l'étude du dispatching économique". In: *Bulletin de la Société Francaise des électriciens* 8.3 (1962), pp. 431–447.

[25] Anya Castillo et al. "The Unit Commitment Problem With AC Optimal Power Flow Constraints". In: *IEEE Transactions on Power Systems* 31.6 (2016), pp. 4853–4866.

[26] Chen Chen, Alper Atamturk, and Shmuel S. Oren. "Bound Tightening for the Alternating Current Optimal Power Flow Problem". In: *IEEE Transactions on Power Systems* 31.5 (2016), pp. 3729–3736. DOI: 10.1109/tpwrs.2015.2497160.

[27] Aaron Clauset. "Finding community structure in very large networks". In: *Physical Review E* 70.6 (2004). DOI: 10.1103/PhysRevE.70.066111.

[28] Carleton Coffrin, Hassan L Hijazi, and Pascal Van Hentenryck. "Strengthening convex relaxations with bound tightening for power network optimization". In: *International conference on principles and practice of constraint programming*. Springer. 2015, pp. 39–57.

[29] J. Czyzyk, M.P. Mesnier, and J.J. More. "The NEOS Server". In: *IEEE Computational Science and Engineering* 5.3 (1998), pp. 68–75.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[30] Evrim Dalkiran and Hanif D. Sherali. "RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems". In: *Mathematical Programming Computation* 8 (2016), pp. 337–375.

[31] Evrim Dalkiran and Hanif D. Sherali. "Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality". In: *Journal of Global Optimization* 4 (2013), pp. 1147–1172.

[32] Pelin Damcı-Kurt et al. "A polyhedral study of production ramping". In: *Mathematical Programming* 158.1-2 (2016), pp. 175–205.

[33] Giovanni Di Liberto et al. "DASH: Dynamic Approach for Switching Heuristics". In: *European Journal of Operational Research* 248.3 (2016), pp. 943–953. DOI: 10.1016/j.ejor.2015.08.018.

[34] J.C. Dodu et al. "An optimal formulation and solution of short-range operating problems for a power system with flow constraints". In: *Proceedings of the IEEE* 60.1 (1972), pp. 54–63.

[35] Elizabeth D. Dolan and Jorge J. Moré. "Benchmarking optimization software with performance profiles". In: *Mathematical Programming* 91 (2002), pp. 201–213. DOI: 10.1007/s101070100263.

[36] Marc Etheve et al. "Reinforcement learning for variable selection in a branch and bound algorithm". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2020, pp. 176–185.

[37] Yuri Faenza, Gonzalo Muñoz, and Sebastian Pokutta. "New limits of treewidth-based tractability in optimization". In: *Mathematical Programming* 191 (2022), pp. 559–594.

[38] Jacques Faraut and Adam Korányi. *Analysis on Symmetric Cones*. Oxford University Press, Dec. 1994. DOI: 10.1093/oso/9780198534778.001.0001.

[39] Matteo Fasiolo et al. "Fast Calibrated Additive Quantile Regression". In: *Journal of the American Statistical Association* 116.535 (2021), pp. 1402–1412. DOI: 10.1080/01621459.2020.1725521.

[40] FICO. *FICO Xpress Optimization*. Available at: https://community.fico.com/s/optimization. 2023.

[41] John Forrest and Julian Hall. *COIN-OR Linear Programming (CLP)*. Computational Infrastructure for Operations Research. Available at: https://www.coin-or.org/Clp/. 2012.

[42] Python Software Foundation. *Python Language Reference, version 3.11*. Available at: http://www.python.org. 2024.

[43] Stephen Frank and Steffen Rebennack. "An introduction to optimal power flow: Theory, formulation, and examples". In: *IIE Transactions* 48.12 (2016), pp. 1172–1197.

[44] Stephen Frank, Ingrida Steponavice, and Steffen Rebennack. "Optimal power flow: a bibliographic survey I. Formulations and deterministic methods". In: *Energy Systems* 3.3 (2012), pp. 221–258.

[45] Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[46] Jerome H. Friedman. "Stochastic gradient boosting". In: *Computational Statistics & Data Analysis* 38.4 (2002), pp. 367–378. DOI: 10.1016/S0167-9473(01)00065-2.

[47] Y. Fu, M. Shahidehpour, and Z. Li. "AC Contingency Dispatch Based on Security-Constrained Unit Commitment". In: *IEEE Transactions on Power Systems* 21.2 (2006), pp. 897–908.

[48] Delbert Fulkerson and Oliver Gross. "Incidence matrices and interval graphs". In: *Pacific Journal of Mathematics* 15.3 (Sept. 1965), pp. 835–855. DOI: 10.2140/pjm.1965.15.835.

[49] Fabio Furini et al. "QPLIB: a library of quadratic programming instances". In: *Mathematical Programming Computation* 1 (2018), pp. 237–265.

[50] Gerald Gamrath and Christoph Schubert. "Measuring the impact of branching rules for mixed-integer programming". In: *Operations Research Proceedings 2017: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR), Freie Universiät Berlin, Germany, September 6-8, 2017*. Springer. 2018, pp. 165–170.

[51] L. L. Garver. "Power Generation Scheduling by Integer Programming-Development of Theory". In: *Transactions of the American Institute of Electrical Engineers. Part III: Power Apparatus and Systems* 81.3 (1962), pp. 730–734.

[52] Maxime Gasse et al. *Exact Combinatorial Optimization with Graph Convolutional Neural Networks*. 2019. arXiv: 1906.01629 [cs.LG]. URL: https://arxiv.org/abs/1906.01629.

[53] David M. Gay. *Writing .nl Files*. Tech. rep. Sandia National Laboratories, 2005.

[54] Bissan Ghaddar et al. "Learning for Spatial Branching: An Algorithm Selection Approach". In: *INFORMS Journal on Computing* 35.5 (2023), pp. 1024–1043. DOI: 10.1287/ijoc.2022.0090.

[55] Ambros M. Gleixner et al. "Three enhancements for optimization-based bound tightening". In: *Journal of Global Optimization* 67.4 (2017), pp. 731–757. DOI: 10.1007/s10898-016-0450-4.

[56] Carla P. Gomes and Bart Selman. "Algorithm portfolios". In: *Artificial Intelligence* 126.1 (2001), pp. 43–62. DOI: 10.1016/S0004-3702(00)00081-3.

[57] Ignacio Gómez-Casares et al. *Impact of domain reduction techniques in polynomial optimization: A computational study*. 2024. arXiv: 2403.02823 [math.OC]. URL: https://arxiv.org/abs/2403.02823.

[58] Brais González Rodríguez. "Advances in polynomial optimization". PhD thesis. 2022.

[59] Brais González-Rodríguez et al. "Computational advances in polynomial optimization: RAPOSa, a freely available global solver". In: *Journal of Global Optimization* 85 (2023), pp. 541–568.

[60] Brais González-Rodríguez et al. *Learning in Spatial Branching: Limitations of Strong Branching Imitation*. 2024. arXiv: 2406.03626 [math.OC]. URL: https://arxiv.org/abs/2406.03626.

[61] Brais González-Rodríguez et al. *Polynomial Optimization: Enhancing RLT relaxations with Conic Constraints*. 2022. arXiv: 2208.05608 [math.OC]. URL: https://arxiv.org/abs/2208.05608.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[62] Devon R. Graham, Kevin Leyton-Brown, and Tim Roughgarden. *Formalizing Preferences Over Runtime Distributions*. 2023. arXiv: 2205.13028 [cs.AI]. URL: https://arxiv.org/abs/2205.13028.

[63] Prateek Gupta et al. "Hybrid Models for Learning to Branch". In: *NeurIPS*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/d1e946f4e67db4b362ad23818a6fb78a-Abstract.html.

[64] Gurobi Optimization. *Gurobi Optimizer Reference Manual*. Available at: http://www.gurobi.com. 2023.

[65] William E Hart, Jean-Paul Watson, and David L Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260.

[66] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. 2nd ed. Springer, New York, NY, 2009. DOI: 10.1007/978-0-387-84858-7.

[67] Clyde N. Herrick. *Basic Electronics Math*. Newnes, 1996. DOI: 10.1016/B978-075069727-9/50001-3.

[68] R. Horst. "A general class of branch-and-bound methods in global optimization with some new approaches for concave minimization". In: *Journal of Optimization Theory and Applications* 51.2 (1986), pp. 271–291. DOI: 10.1007/BF00939825.

[69] Reiner Horst and Hoang Tuy. *Global Optimization*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, p. 535.

[70] IBM Corp. *IBM ILOG CPLEX Optimization Studio. CPLEX User's Manual*. Available at: https://www.ibm.com/es-es/products/ilog-cplex-optimization-studio. 2022.

[71] Rohit Kannan, Harsha Nagarajan, and Deepjyoti Deka. *Learning to Accelerate the Global Optimization of Quadratically-Constrained Quadratic Programs*. 2023. arXiv: 2301.00306 [math.OC]. URL: https://arxiv.org/abs/2301.00306.

[72] Elias B. Khalil et al. "Learning to Branch in Mixed Integer Programming". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2016. DOI: 10.1609/aaai.v30i1.10080.

[73] Ben Knueven, Jim Ostrowski, and Jianhui Wang. "The Ramping Polytope and Cut Generation for the Unit Commitment Problem". In: *INFORMS Journal on Computing* 30.4 (2018), pp. 739–749.

[74] Roger Koenker. *Quantile Regression*. Econometric Society Monographs. Cambridge University Press, 2005.

[75] Roger Koenker et al. *Handbook of Quantile Regression*. CRC Press, 2017. DOI: 10.1201/9781315120256.

[76] Brian Kriegler and Richard Berk. "Small area estimation of the homeless in Los Angeles: An application of cost-sensitive stochastic gradient boosting". In: *The Annals of Applied Statistics* 4.3 (2010), pp. 1234–1255. DOI: 10.1214/10-AOAS328.

[77] Vito Latora, Vincenzo Nicosia, and Giovanni Russo. *Complex networks: principles, methods and applications*. Cambridge University Press, 2017.

[78] Javad Lavaei and Steven H. Low. "Zero Duality Gap in Optimal Power Flow Problem". In: *IEEE Transactions on Power Systems* 27.1 (2012), pp. 92–107.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[79] E. L. Lawler and D. E. Wood. "Branch-And-Bound Methods: A Survey". In: *Operations Research* 14.4 (1966), pp. 699–719.

[80] Karsten Lehmann, Alban Grastien, and Pascal Van Hentenryck. "AC-Feasibility on Tree Networks is NP-Hard". In: *IEEE Transactions on Power Systems* 31.1 (2016), pp. 798–801.

[81] Adam N. Letchford and Andrew J. Parkes. "A guide to conic optimisation and its applications". In: *RAIRO - Operations Research* 52.4-5 (2018), pp. 1087–1106.

[82] Kevin Leyton-Brown et al. "A Portfolio Approach to Algorithm Selection". In: *Proceedings of the 18th International Joint Conference on Artificial Intelligence*. IJCAI'03. Acapulco, Mexico: Morgan Kaufmann Publishers Inc., 2003, pp. 1542–1543.

[83] Youdong Lin and Linus Schrage. "The global solver in the LINDO API". In: *Optimization Methods & Software* 24.4-5 (2009), pp. 657–668.

[84] J. T. Linderoth and M. W. P. Savelsbergh. "A Computational Study of Search Strategies for Mixed Integer Programming". In: *INFORMS Journal on Computing* 11.2 (1999), pp. 173–187. DOI: 10.1287/ijoc.11.2.173.

[85] Andrea Lodi and Giulia Zarpellon. "On learning and branching: a survey". In: *Top* 25.2 (2017), pp. 207–236.

[86] Miles Lubin et al. "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization". In: *Mathematical Programming Computation* (2023). DOI: 10.1007/s12532-023-00239-3.

[87] H. Ma and S.M. Shahidehpour. "Unit commitment with transmission security and voltage constraints". In: *IEEE Transactions on Power Systems* 14.2 (1999), pp. 757–764.

[88] Marian G. Marcovecchio, Augusto Q. Novais, and Ignacio E. Grossmann. "Deterministic optimization of the thermal Unit Commitment problem: A Branch and Cut search". In: *Computers & Chemical Engineering* 67 (2014), pp. 53–68.

[89] Garth P McCormick. "Computability of global solutions to factorable nonconvex programs: Part I – Convex underestimating problems". In: *Mathematical programming* 10.1 (1976), pp. 147–175.

[90] Nicolai Meinshausen. "Quantile Regression Forests". In: *Journal of Machine Learning Research* 7 (2006), pp. 983–999.

[91] Ruth Misener and Christodoulos A Floudas. "ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations". In: *Journal of Global Optimization* 59.2 (2014), pp. 503–526.

[92] Luis Montero, Antonio Bello, and Javier Reneses. "A Review on the Unit Commitment Problem: Approaches, Techniques, and Resolution Methods". In: *Energies* 15.4 (2022), p. 1296.

[93] MOSEK ApS. *Introducing the MOSEK Optimization Suite 9.3.20*. 2022. URL: https://docs.mosek.com/latest/intro/index.html.

[94] MOSEK ApS. *MOSEK Modeling Cookbook. Release 3.3.0*. 2024. URL: https://docs.mosek.com/modeling-cookbook/index.html.

[95] C. Murillo-Sanchez and R.J. Thomas. "Thermal unit commitment with nonlinear power flow constraints". In: *Thermal unit commitment with nonlinear power flow constraints*. IEEE, 1999, p. 10.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[96] Amin Nasri et al. “Network-Constrained AC Unit Commitment Under Uncertainty: A Benders’ Decomposition Approach”. In: *IEEE Transactions on Power Systems* 31.1 (2016), pp. 412–422.

[97] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994. DOI: 10.1137/1.9781611970791.

[98] John O’Malley. *Schaum’s Outline of Basic Circuit Analysis, Second Edition*. McGraw-Hill Education, 2011.

[99] Octeract. *Octeract Eengine Documentation*. Available at: https://octeract.gg/docs/. 2024.

[100] James Ostrowski, Miguel F. Anjos, and Anthony Vannelli. “Modified orbital branching for structured symmetry with an application to unit commitment”. In: *Mathematical Programming* 150.1 (2015), pp. 99–129.

[101] James Ostrowski, Miguel F. Anjos, and Anthony Vannelli. “Tight Mixed Integer Linear Programming Formulations for the Unit Commitment Problem”. In: *IEEE Transactions on Power Systems* 27.1 (2012), pp. 39–46.

[102] N.P. Padhy. “Unit Commitment—A Bibliographical Survey”. In: *IEEE Transactions on Power Systems* 19.2 (2004), pp. 1196–1205.

[103] Christopher W. F. Parsonson, Alexandre Laterre, and Thomas D. Barrett. *Reinforcement Learning for Branch-and-Bound Optimisation using Retrospective Trajectories*. 2022. arXiv: 2205.14345 [cs.LG]. URL: https://arxiv.org/abs/2205.14345.

[104] Laurent Perron and Vincent Furnon. *OR-Tools*. Version v9.9. Google, Mar. 7, 2024. URL: https://developers.google.com/optimization/.

[105] Yash Puranik and Nikolaos V. Sahinidis. “Domain reduction techniques for global NLP and MINLP optimization”. In: *Constraints* 22.3 (2017), pp. 338–376.

[106] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2023. URL: https://www.R-project.org.

[107] Neil Robertson and Paul D Seymour. “Graph minors. III. Planar tree-width”. In: *Journal of Combinatorial Theory, Series B* 36.1 (1984), pp. 49–64.

[108] Hong S. Ryoo and Nikolaos V. Sahinidis. “A branch-and-reduce approach to global optimization”. In: *Journal of Global Optimization* 8.2 (1996), pp. 107–138.

[109] Nikolaos V Sahinidis. “BARON: A general purpose global optimization software package”. In: *Journal of global optimization* 8 (1996), pp. 201–205.

[110] Nikolaos V. Sahinidis. *BARON 24.1.30: Global Optimization of Mixed-Integer Nonlinear Programs, User’s Manual*. https://www.minlp.com/downloads/docs/baron%20manual.pdf. 2024.

[111] Lara Scavuzzo et al. “Learning to branch with tree mdps”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 18514–18526.

[112] Lara Scavuzzo et al. *Machine Learning Augmented Branch and Bound for Mixed Integer Linear Programming*. 2024. arXiv: 2402.05501 [math.OC].

[113] Paul Scherz and Simon Monk. *Practical Electronics for Inventors, Fourth Edition*. McGraw-Hill Education TAB, 2016, p. 1056.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[114] Dmitry Shchetinin, Tomas Tinoco De Rubira, and Gabriela Hug. "Efficient Bound Tightening Techniques for Convex Relaxations of AC Optimal Power Flow". In: *IEEE Transactions on Power Systems* 34.5 (2019), pp. 3848–3857. DOI: 10.1109/tpwrs.2019.2905232.

[115] Hanif D. Sherali. "RLT: A unified approach for discrete and continuous nonconvex optimization". In: *Annals of Operations Research* 149.1 (2007), pp. 185–193.

[116] Hanif D. Sherali and Warren P. Adams. "A hierarchy of relaxations and convex hull characterizations for mixed-integer zero—one programming problems". In: *Discrete Applied Mathematics* 52.1 (1994), pp. 83–106.

[117] Hanif D. Sherali and Warren P. Adams. "A Hierarchy of Relaxations between the Continuous and Convex Hull Representations for Zero-One Programming Problems". In: *SIAM Journal on Discrete Mathematics* 3.3 (1990), pp. 411–430. DOI: 10.1137/0403036.

[118] Hanif D. Sherali, Evrim Dalkiran, and Jitamitra Desai. "Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts". In: *Computational Optimization and Applications* 52.2 (2012), pp. 483–506. DOI: 10.1007/s10589-011-9425-z.

[119] Hanif D. Sherali, Evrim Dalkiran, and Leo Liberti. "Reduced RLT representations for nonconvex polynomial programming problems". In: *Journal of Global Optimization* 52 (2012), pp. 447–469. DOI: 10.1007/s10898-011-9757-3.

[120] Hanif D. Sherali and Cihan H. Tuncbilek. "A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique". In: *Journal of Global Optimization* 1 (1992), pp. 101–112.

[121] Wilfredo S. Sifuentes and Alberto Vargas. "Hydrothermal Scheduling Using Benders Decomposition: Accelerating Techniques". In: *IEEE Transactions on Power Systems* 22.3 (2007), pp. 1351–1359.

[122] Probability Subcommittee. "IEEE Reliability Test System". In: *IEEE Transactions on Power Apparatus and Systems* PAS-98.6 (1979), pp. 2047–2054.

[123] Kaarthik Sundar et al. *Optimization-Based Bound Tightening using a Strengthened QC-Relaxation of the Optimal Power Flow Problem*. 2019. arXiv: 1809.04565 [math.OC]. URL: https://arxiv.org/abs/1809.04565.

[124] Mohit Tawarmalani and Nikolaos V Sahinidis. "A polyhedral branch-and-cut approach to global optimization". In: *Mathematical programming* 103.2 (2005), pp. 225–249.

[125] Mohit Tawarmalani and Nikolaos V Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*. Vol. 65. Springer Science & Business Media, 2013.

[126] Mohit Tawarmalani and Nikolaos V Sahinidis. "Global optimization of mixed-integer nonlinear programs: A theoretical and computational study". In: *Mathematical Programming* 99.3 (2004), pp. 563–591.

[127] Alexander Tornede et al. "Run2Survive: a decision-theoretic approach to algorithm selection based on survival analysis". In: *Asian Conference on Machine Learning*. PMLR. 2020, pp. 737–752.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

[128] Hayato Waki et al. “Algorithm 883. SparsePOP—A Sparse Semidefinite Programming Relaxation of Polynomial Optimization Problems”. In: *ACM Transactions on Mathematical Software* 35.2 (2008), pp. 1–13.

[129] Hayato Waki et al. “Sums of Squares and Semidefinite Program Relaxations for Polynomial Optimization Problems with Structured Sparsity”. In: *SIAM Journal on Optimization* 17.1 (2006), pp. 218–242. DOI: 10.1137/050623802.

[130] Huixia Judy Wang and Deyuan Li. “Estimation of Extreme Conditional Quantiles Through Power Transformation”. In: *Journal of the American Statistical Association* 108.503 (2013), pp. 1062–1074. DOI: 10.1080/01621459.2013.820134.

[131] Jie Wang, Haokun Li, and Bican Xia. “A New Sparse SOS Decomposition Algorithm Based on Term Sparsity”. In: *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation*. ISSAC ’19. Beijing, China: Association for Computing Machinery, 2019, pp. 347–354. DOI: 10.1145/3326229.3326254.

[132] Jie Wang et al. “CS-TSSOS: Correlative and Term Sparsity for Large-Scale Polynomial Optimization”. In: *ACM Transactions on Mathematical Software* 48.4 (2022), pp. 1–26.

[133] Marvin N. Wright and Andreas Ziegler. “ranger: A Fast Implementation of Random Forests for High Dimensional Data in C++ and R”. In: *Journal of Statistical Software* 77.1 (2017), pp. 1–17. DOI: 10.18637/jss.v077.i01.

[134] Ray Daniel Zimmerman, Carlos Edmundo Murillo-Sánchez, and Robert John Thomas. “MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education”. In: *IEEE Transactions on Power Systems* 26.1 (2011), pp. 12–19. DOI: 10.1109/TPWRS.2010.2051168.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Resumo en galego

No ámbito da investigación operativa, en más en concreto, da optimización matemática, os esforzos de desenvolvemento de algoritmos céntranse fundamentalmente nos problemas lineares ou nos problemas non lineares xerais. Pero existe unha clase "intermedia" de problemas, os problemas polinómicos, ós que non se lles prestou moita atención pero que constitúen un conxunto de problemas importantes no ámbito da optimización matemática. Isto é debido a dous aspectos. En primeiro lugar, estes problemas son máis xerais que os problemas lineais, e aínda que no cobren todas as casuísticas que un problema non linear xeral (ou incluso un problema non linear convexo, para os que existen algoritmos eficientes para a súa resolución), si que permiten o modelado de moitos problemas presentes na práctica, como poden ser os problemas cuadráticos ou problemas con variables binarias (estes últimos non son polinómicos na súa formulación básica, pero si que se poden reformular para obter un problema polinómico). En segundo lugar, na práctica aparecen non só estes problemas cuadráticos ou binarios, senón que outros problemas polinómicos son utilizados habitualmente para modelar situacións reais, coma por exemplo a optimización de redes eléctricas. Nesta tese, estudiamos este campo dos problemas polinómicos, dende a súa formulación até a solución dos problemas, a utilización de técnicas de aprendizaxe estatístico para mellorar o rendemento da resolución ou a aplicación destes problemas polinómicos ó ámbito da optimización de redes eléctricas.

Na primeira parte da tese, na Parte 1, centrámonos no estudo dunha técnica deseñada para resolver problemas de optimización polinómicos xerais, onde tanto a función obxectivo do problema, coma as restricións poden estar definidas por polinomios. Esta técnica chámase a Reformulation-Linearization Technique, que pasamos a denominar a partires de agora como a técnica RLT. Esta foi utilizada por investigadores da Universidade de Santiago de Compostela, e do Centro de Investigación e Tecnoloxía Matemática de Galicia ou CITMAga, entre eles o meu director de tese Julio González Díaz, para desenvolver un software de resolución de problemas de optimización baseado nesta técnica RLT. Este optimizador, nomeado polos seus creadores RAPOSa, Reformulation Algorithm for Polynomial Optimization Santiago, pódese descargar gratuitamente aínda que non é un software de código aberto. RAPOSa permite a resolución de problemas polinómicos xerais, e pódese utilizar tamén na plataforma online NEOS, onde o ano pasado foi utilizado arredor de 1000 veces por usuarios de todo o mundo. Que RAPOSa sexa o software utilizado para a obtención de resultados nesta parte da tese, non implica que estes non se poidan estender ó ámbito da optimización polinómica en xeral, ou incluso máis aló da optimización polinómica, a todo optimizador deseñado sobre a mesma base que a técnica RLT, é dicir, todo optimización que constrúa unha relaxación do problema e despois utilice una técnica de ramificación e acotación para obter

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

solucións para o problema polinómico orixinal. A pesares de todo, como nesta tese necesitabamos elixir un optimizador para obter os resultados, e ademais temos un control completo sobre o código de RAPOSa, e consecuentemente podemos probar todas aquelas modificacións do algoritmo que queiramos, utilizamos RAPOSa no que se refire ós resultados computacionáis da tese. Utilizando RAPOSa para obter os resultados, compararemos nos distintos capítulos como diferentes configuracións do optimizador, nas que modificamos pasos na técnica RLT para intentar introducir melloras nos mesmos, lévannos a distinto rendemento de RAPOSa en xeral, o que tamén se podería estender a outros algoritmos para resolver problemas non lineares. No Capítulo 1 introducimos formalmente os problemas polinómicos, a técnica RLT e RAPOSa. Ademais, tamén describimos e explicamos diferentes melloras que se poden incorporar á técnica RLT e que xa están implementadas en RAPOSa, como por exemplo técnicas de redución de dominio, o uso de J-sets para reducir o tamaño da relaxación que se constrúe ou o uso de técnicas de optimización cónica para obter unha mellor relaxación linear do problema polinómico. No Capítulo 2 estudiamos o rendemento de distintas melloras que se poden introducir nas técnicas de bound tightening ou axuste de cotas. Estas melloras xa foron propostas no ámbito da optimización non linear en xeral, e aquí as implementamos nun optimizador pensado para problemas polinómicos e estudamos o seu rendemento. Máis aínda, propoñemos un cambio sinxelo que se pode facer no algoritmo, que é cambiar o punto de ramificación, que leva a unha diferencia notable de rendemento para un cambio tan sinxelo como é este. O Capítulo 3 analiza como outra decisión sinxela, como é se mandarlle ou non ó optimizador linear auxiliar as cotas das variables RLT que se crean ó construír a relaxación do problema, tamén pode ter un impacto grande no rendemento dun algoritmo baseado nunha técnica de ramificación e acotación. Ademais, vese tamén nos resultados como, dependendo da elección do optimizador auxiliar, isto ten un impacto moi distinto. Tamén inflúe a natureza dos problemas resoltos, xa que se observa unha diferencia no impacto segundo o conxunto de problemas dos que proveña cada un. Finalmente, no Capítulo 4 presentamos unha extensión da técnica RLT para poder resolver problemas con variables non acotadas, é dicir, variables onde a cota superior das mesmas non é finita. Aínda que na maior parte dos problemas reais si que se coñecen, ou son doadas de obter utilizando o coñecemento que se ten da situación real que se modela, as cotas das variables, moitos dos problemas non as inclúen na súa formulación. Isto ocorre con moitos dos problemas das baterías que usamos nesta tese para os resultados numéricos. No só isto, senón que moitas veces interesa saber se o límite imposto nunha variable está impedindo o algoritmo mellorar o óptimo do problema. Nese caso poderíase estudar a razón pola que dita variable ten esa cota e facer as modificacións pertinentes para incrementar ou diminuír o seu valor. Con esta modificación da técnica, introducimos dous modificacións no algoritmo, e posteriormente discutimos a súa interacción con outras partes do algoritmo así como presentamos uns resultados preliminares do rendemento da súa implementación en RAPOSa. Además, demostramos un resultado de converxencia estendendo o resultado orixinal da técnica RLT.

A Parte 2 da tese está adicada ó estudo da utilización de técnicas de aprendizaxe estatístico para a mellora do rendemento de algoritmos de optimización. Esta interacción entre estes dous campos das matemáticas está collendo moita relevancia nos últimos anos, especialmente no ámbito académico da optimización matemática.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

A optimización sempre tivo unha estreita relación có aprendizaxe automático ou aprendizaxe estatístico, xa que non existirían moitas das técnicas de aprendizaxe sen un algoritmo de optimización detrás, que permite calcular os parámetros adoitados para logo facer predicións para novos valores que se introduzan no modelo. Nos últimos anos, estes modelos de aprendizaxe estatístico, estanse a utilizar en diversos ámbitos polo seu bo rendemento á hora de detectar patróns en datos de diversa índole. Os investigadores en optimización tamén comezaron a probar estes modelos de aprendizaxe, coa intención en primeiro lugar de imitar tarefas custosas computacionalmente dentro dos algoritmos, pero que funcionan ben, utilizando estas técnicas de aprendizaxe. Con isto, o que conseguen e utilizar o histórico destas tarefas custosas e mediante métodos estatísticos de aprendizaxe analizar ese histórico para desenvolver un modelo que poida facer o mesmo pero sen ese coste computacional engadido. Por outra banda, tamén se están a utilizar estes métodos para elixir entre diferentes posibilidades á hora de resolver un problema de optimización. Estas posibilidades poden xurdir polas distintas configuracións posibles dun mesmo optimizador, ou polo uso de distintos optimizadores que poidan resolver o mesmo tipo de problemas. Cada un dos optimizadores ou das configuracións, desenvolverase mellor ou peor dependendo da natureza e da estrutura do problema que se quere resolver, e utilizando técnicas de aprendizaxe é posible detectar estes patróns e anticiparse a eles.

Para poder aplicar estas técnicas de aprendizaxe no ámbito da optimización, é necesario ter un sistema de aprendizaxe pensado para este tipo de situacións. No Capítulo 5 presentamos xustamente iso: un sistema completo de aprendizaxe pensado para o ámbito da optimización. Este sistema inclúe unha proposta de medida de rendemento, adoitada para despois ser utilizada como variable resposta para o posterior adestramento do modelo; unha proposta de variables explicativas que se poden calcular dado un problema de optimización polinómica, incluíndo no conxunto de variables explicativas algunhas calculadas a partires de grafos que se constrúen como resumos da estrutura do modelo de optimización polinómica; e una selección de técnicas de aprendizaxe que son adoitadas para estas variables explicativas e esta medida de rendemento. Posteriormente, no mesmo capítulo, utilízase este sistema introducido en varias situacións. En primeiro lugar, inténtase aprender cal é o mellor criterio de ramificación para un problema determinado, de entre os definidos no optimizador RAPOSa. En segundo lugar, aplícase a técnica para decidir cando é interesante utilizar as melloras propostas nos capítulos anteriores, en cando non. E por último, compróbase a versatilidade da técnica aplicándoa a outro contexto distinto, cunha medida de rendemento diferente e outras variables explicativas, para seleccionar cal é o mellor optimizador para resolver un problema non linear xeral, escollendo entre catro optimizadores comerciais coñecidos no mundo da optimización non linear. Este capítulo céntrase no que se chama o aprendizaxe estático con variables explicativas estáticas, isto é, un modelo que se adestra unha vez cos datos históricos, e non se actualiza durante a execución do algoritmo; e unhas variables explicativas que unicamente recollen características do problema de optimización mais non introducen ningunha información referente á execución do optimizador. Con todo, non é a única posibilidade, xa que neste último caso, é doado introducir variables referentes á árbore de ramificación e acotación por exemplo, como poderían ser a profundidade dun nodo, ou o número de veces cunha variable foi seleccionada para a ramificación. Isto provocaría que o aprendizaxe xa

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

non se puidese facer a nivel de problema, mais deberíase facer a nivel de nodo da árbore de ramificación de acotación. Aínda que isto pódese facer, con máis ou menos traballo, a pregunta que intentamos responder no Capítulo 6 é se merece a pena facer o esforzo de adaptar a técnica a dito contexto. Ademais, non só buscamos responder a esta pregunta, mais tamén imos máis alá e analizamos se o estándar na optimización linear e enteira referente á selección da variable de ramificación, o strong branching, segue sendo a mellor opción, ou incluso una opción boa no caso da optimización polinómica. Esta pregunta non xurde casualmente, xa que a natureza da ramificación nun problema no que o obxectivo e obter una solución enteira é moi distinta á ramificación cando o que se quere é cumprir unhas restricións que forzan a unhas variables a representar adecuadamente ós correspondentes monomios do problema orixinal.

Finalmente, a Parte 3 da tese está adicada a estudiar un problema particular dentro do ámbito da optimización polinómica. Este problema xurde como problema utilizado para xestionar a operación dunha rede eléctrica, na que se teñen distintos "nodos" (os usuarios nas casas, as empresas e os edificios, o alumeado público, etc.) que demandan enerxía, distintos "nodos" nos que se xera enerxía (centrais nucleares ou térmicas, instalacións fotovoltaicas ou de aeroxeradores) e todas as liñas que conectan estes "nodos". Estas redes adóitanse representar por un grafo, que é sobre o que se define o problema de optimización. Para poder definir o devandito problema, no Capítulo 7 introdúcese ó lector nos conceptos básicos da electricidade por unha banda e os problemas de xestión de redes eléctricas por outra banda. No primeiro, introdúcese o concepto e a forma de denotar ás variables asociadas coa corrente alterna, que é a que se adoita utilizar nas liñas de transmisión de media e alta tensión por exemplo. A corrente alterna recibe ese nome porque os electróns non sempre se moven na mesma dirección, mais cambiar regularmente de dirección. Isto pódese modelar utilizando funcións periódicas coma as trigonométricas. Ademais, estes cambios de dirección xeran un novo tipo de resistencia diferente á propia resistencia do material ó paso dos electróns. Para modelar isto, utilízanse adecuadamente os números complexos, o que complica o tratamento matemático deste tipo de corrente. Para o segundo, isto é, para a introdución ó lector nos problemas de xestión de redes eléctricas, defínense formalmente dous problemas utilizados na xestión real de redes eléctricas. O primeiro deles, o problema de Optimal Power Flow ou OPF como o chamaremos a partires de agora, é un problema moi utilizado que permite calcular canta enerxía debe producir cada xerador dunha rede eléctrica, de forma que a demanda total da rede satisfágase y ó mesmo tempo a estabilidade da rede eléctrica (as propiedades físicas que ten que cumprir a corrente que circula por ela, para o correcto funcionamento dos aparatos que consumen dita corrente) estea garantida. Este problema utiliza estas propiedades da corrente alterna que nomeábamos antes para modelar a voltaxe que cada nodo da rede inxecta na mesma. Esta voltaxe depende á súa vez da voltaxe de todos os demais nodos da rede, o que fai deste problema de optimización un problema difícil de resolver, dado que o problema resultante é un problema non linear, non convexo e demostrouse que é un problema NP-duro, é dicir, non existe ningún algoritmo que poda resolvelo en tempo polinomial respecto ó número de variables do problema. Por outro lado, tamén se define o problema de Unit Commitment ou UC, como o chamaremos a partires de agora. Este problema é moito máis coñecido entre os investigadores da optimización matemática, xa que constitúe un importante exemplo de problema de

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

optimización linear e enteira. O obxectivo do problema é o mesmo co do problema de OPF: decidir canta enerxía xerar en cada xerador da rede. Con todo, neste caso hai dúas diferencias co problema anterior. A primeira delas é que non se traballa unicamente cun período de tempo, senón que trabállase con varios períodos de tempo (que poden representar a horas, días, semanas, etc.). Isto da lugar a novas restricións, que limitan cousas como os períodos de arrefriamento dos xeradores unha vez son apagados, a máxima enerxía que poden producir nada máis son acendidos, e canto poden aumentar dende un período de tempo ó seguinte, etc. A segunda diferencia é que, neste caso, non hai restricións que garantan a estabilidade da rede eléctrica, e unicamente búscase satisfacer a demanda dos diferentes usuarios da rede. Este problema, de forma similar ó problema de OPF, tamén é un problema difícil. Neste caso a dificultade ven dada pola combinatoria presente no problema, debida a utilización de variables enteiras, pero constitúe tamén unha dificultade grande e demostrouse que este problema de UC tamén é un problema NP-duro. Tamén neste capítulo discutimos distintas maneiras propostas na literatura para resolver de forma eficiente o problema de OPF, técnicas que utilizaremos posteriormente no Capítulo 8. Estas técnicas baséanse na utilización da optimización cónica, mediante a adición de restricións baseadas en matrices semidefinidas positivas. Ademais, tamén explicamos que a utilización directa destas restricións non é moi eficiente, debido ó tamaño que toman estas matrices, e que existen diversas técnicas para reducir o tamaño destas restricións, aproveitando a estrutura de "sparsity" destas matrices.

A pregunta natural que xurde despois de definir os dous problemas anteriores, é se hai posibilidade de xuntar ámbolos dous e conseguir un problema de optimización que permita decidir canta enerxía teño que xerar en cada xerador por cada período de tempo, de forma que garanta a estabilidade da rede en cada período de tempo, satisfaga a demanda de todos os usuarios e ó mesmo tempo tamén as limitacións en canto á modificación da enerxía xerada dun período ó seguinte así coma o mínimo tempo que necesitan os xeradores para arrefriarse unha vez son apagados. A resposta a esta pregunta é afirmativa. Os investigadores neste ámbito propuxeron un posible modelado deste problema, que se chama o problema Security-Constrained Unit Commitment ou problema SCUC como nos referiremos a el desde este momento. O problema SCUC defínese como a combinación natural dos problemas de OPF e UC, combinando as súas restricións de forma axeitada. Coma estamos xuntando dous problemas con dificultades para a súa resolución (parte combinatoria polas variables enteiras nun caso e a presenza de non linearidades no outro), dificultades que non desaparecen no proceso de combinación dos problemas, estamos ante un problema difícil de resolver, se cabe máis difícil aínda cos problemas de OPF e UC por separado, que tamén é un problema NP-duro, non linear, non convexo e con variables enteiras. No Capítulo 8 definimos formalmente este problema, combinando as formulacións propostas no Capítulo 7. Ademais, analizamos como podemos estender as técnicas nomeadas anteriormente para resolver eficientemente o problema de OPF para aplicalas neste novo problema. Todo isto necesita unha implementación que presentamos como parte final do capítulo, implementación na que combinamos as técnicas de resolución de problemas cónicos, os optimizadores locais para problemas non lineares, e as técnicas para explotar a "sparsity" das matrices. Con esta implementación, aplicándoa a tres problemas distintos, presentamos uns resultados computacionáis que buscan mostrar a capacidade da técnica

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

para obter solucións factibles para o problema SCUC con garantías de optimalidade para as mesmas.

Finalmente, presentamos nos apéndices o entorno deseñado de desenvolvido durante estes anos de realización da tese para levar a cabo as execucións coas que se obtiveron os distintos resultados numéricos (isto no Apéndice A), e formalizamos as probas dos distintos lemas utilizados no Capítulo 1 e tamén no Capítulo 4. A ferramenta presentada no Apéndice A, constitúe a principal ferramenta utilizada no desenvolvemento da tese. Isto é debido a que nunha tese tan centrada nos resultados computacionáis, é necesario facer numerosos experimentos con distintas configuracións do optimizador. Facer isto a man, tendo en conta que as execucións débense facer nun entorno controlado, sempre nas mesmas condicións, e para iso utilizamos o Centro de Supercomputación de Galicia ou CESGA, leva moito tempo e esforzo. No só iso, facelo a man pode levar a que haxa confusións na xestión dos resultados e pode levar a que se ocasionen mesturas dos resultados entre distintas execucións. Deseñar a ferramenta levou unha considerable cantidade de tempo, pero esta viuse compensada notablemente con unha redución no tempo adicado á execución das distintas configuracións e posterior xestión dos resultados. Non só iso, tamén fóronse incorporando outras funcionalidades na ferramenta, como pode ser a aprendizaxe estatística presentada no Capítulo 5, ou a construción de "cotas superiores" do rendemento de RAPOSa en función do estudo de distintas medidas de rendemento.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Acknowledgments

The author acknowledges the funding received from the Spanish Ministry of Education through FPU grant 20/01555 and from the R&D project PID2021-124030NB-C32 granted by MICIU/AEI/10.13039/501100011033. This research was also funded by Grupos de Referencia Competitiva ED431C-2021/24 from the Consellería de Cultura, Educación e Universidades, Xunta de Galicia. The author also acknowledges the Galician Supercomputing Center, CESGA, for providing the computational resources for the executions.

USC
UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

Publications included

We include here a list of the publications included in the thesis.

Bissan Ghaddar et al. "Learning for Spatial Branching: An Algorithm Selection Approach". In: *INFORMS Journal on Computing* 35.5 (2023), pp. 1024–1043. DOI: 10.1287/ijoc.2022.0090

Authors: Bissan Ghaddar from the Ivey Business School at Western University, Ignacio Gómez-Casares, Julio González-Díaz, Brais González-Rodríguez and Beatriz Pateiro-López from the Galician Center for Mathematical Research and Technology and the University of Santiago de Compostela and Sofía Rodrígez-Ballesteros from the Galician Center for Mathematical Research and Technology.

Impact factor: Q2 on OPERATIONS RESEARCH & MANAGEMENT SCIENCE category (42/106, 60.8) in 2023.

Copyright and use: Information about permissions for reuse can be found at https://pubsonline.informs.org/authorportal/rights-permissions. There, it's explicitly mentioned: "Permission is required but is granted at no charge, and includes dissertations published online, e.g., UMI Dissertation Publishing." The author has requested the permission, granted in license 1503052-1 to "Republish in a thesis/dissertation".

Contents of the thesis based on this work: Chapter 5.

Contribution of the author of this thesis: The author of this thesis contributed to the theoretical development of the machine learning framework presented in this work, as well as in its implementation and in the writing of the final document published. He also participated in the development of some of the functionality of the RLT-based solver RAPOSa, some of which are employed in this work.

Ignacio Gómez-Casares et al. *Impact of domain reduction techniques in polynomial optimization: A computational study.* 2024. arXiv: 2403.02823 [math.OC]. URL: https://arxiv.org/abs/2403.02823

Authors: Ignacio Gómez-Casares from the Galician Center for Mathematical Research and Technology and the University of Santiago de Compostela, Brais González-Rodríguez from the University of Santiago de Compostela, Julio González-Díaz from the Galician Center for Mathematical Research and Technology and the University of Santiago de Compostela and Pablo Rodríguez-Fernández from the Galician Center for Mathematical Research and Technology.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Copyright and use: Information about permissions can be found in the website https://info.arxiv.org/help/license/reuse.html. 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 the thesis based on this work: Chapter 2.

Contribution of the author of this thesis: The author of this thesis contributed to the bibliography review, the theoretical adaptation of the techniques to the RLT algorithm, the implementation in RAPOSa and the computation and analysis of the results. He also participated in the writing of the final document sent for publication.

Brais González-Rodríguez et al. *Learning in Spatial Branching: Limitations of Strong Branching Imitation*. 2024. arXiv: 2406.03626 [math.OC]. URL: https://arxiv.org/abs/2406.03626

Authors: Brais González-Rodríguez from the University of Santiago de Compostela, Ignacio Gómez-Casares from the Galician Center for Mathematical Research and Technology and the University of Santiago de Compostela, Bissan Ghaddar from the Ivey Business School at Western University, Julio González-Díaz and Beatriz Pateiro-López from the Galician Center for Mathematical Research and Technology and the University of Santiago de Compostela.

Copyright and use: Information about permissions can be found in the website https://info.arxiv.org/help/license/reuse.html. 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 the thesis based on this work: Chapter 6.

Contribution of the author of this thesis: The author of this thesis contributed to the theoretical discussions that led up to the work, as well as in the implementation, execution and analysis of the results obtained.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

APPENDIX A

Computational environment

Running the executions for the thesis required to solve all of the problems from different test sets with the different configurations of RAPOSa that are the subject of study. Since we are using CESGA for the executions, and therefore need to use its queue system, running the executions by hand has many drawbacks. One is that the waiting time in the queue depends on the time limit imposed on the task sent. In order to get the tasks running faster, less instances can be run on the same task (so the overall time limit of the task, the sum of the time limits of the instances solved in each task), but this poses a problem, as managing the obtained results manually is harder when the instances are split into more tasks. Other drawback is that there is no way to know when a certain CESGA task will be executed, so the user must be available at any time to send more tasks to the queue when the older ones finish (there is a hard limit on the number of tasks one user can put on the queue).

This complexity makes running these executions by hand a daunting task that leads to errors in the executions. To solve this, the author of this thesis, together with Brais González Rodríguez, developed an automatized system to run the executions on CESGA which they called *RAPOSa Cloud*. There are several elements that conform this system:

- A database to store the executions and the results.
- A bot running on CESGA that checks wether there are new executions to send to the queue, and also detects when the executions are finished and uploads the results to the database.
- A web interface to create new executions, send the tasks to CESGA and manage and visualize the results.
- An integrated R [106] flow to use the machine learning framework from Chapter 5.
- A git repository for RAPOSa's code.

All this system is running on a computer at CITMAga. The database is using PostgreSQL (`https://www.postgresql.org`), the git repository uses a self-hosted GitLab instance (`https://about.gitlab.com`), and the interface runs on the platform Ruby on Rails (`https://rubyonrails.org`), using Python and R for the plots and the machine learning parts. For the design of the interface, the CSS code from Bootstrap was used (`https://getbootstrap.com`).

As we can see in Figure A.1, each execution in the system has some properties, such as running time, test sets to run on, RAPOSa's code commit in the git repository and the licenses it depends on (to avoid running more instances of an auxiliary solver than we have licenses for). Once the execution is created, the user sets the RAPOSa's options of interest, as we can see in Figure A.2.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

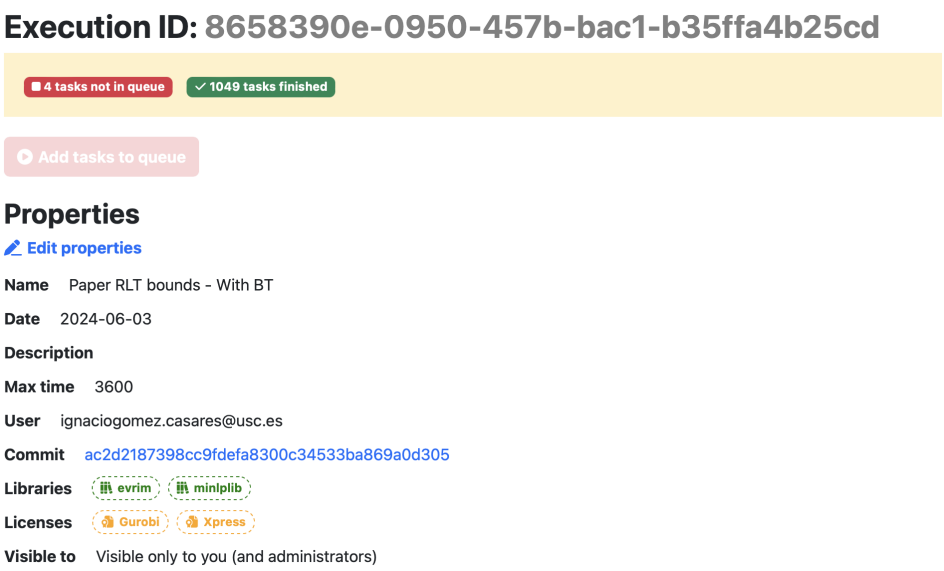
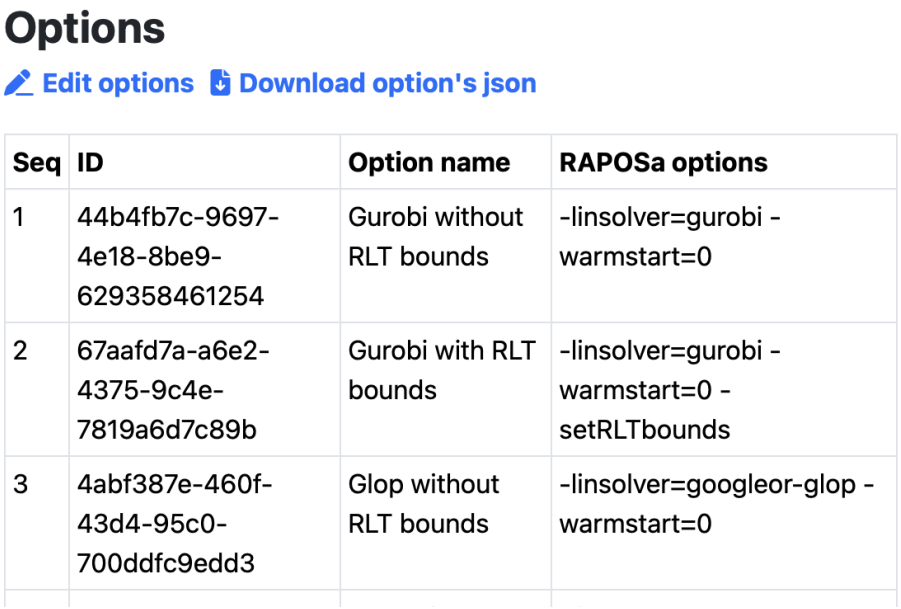


FIGURE A.1. Execution properties in RAPOSa Cloud



Seq	ID	Option name	RAPOSa options
1	44b4fb7c-9697-4e18-8be9-629358461254	Gurobi without RLT bounds	-linsolver=gurobi -warmstart=0
2	67aafd7a-a6e2-4375-9c4e-7819a6d7c89b	Gurobi with RLT bounds	-linsolver=gurobi -warmstart=0 -setRLTbounds
3	4abf387e-460f-43d4-95c0-700ddfc9edd3	Glop without RLT bounds	-linsolver=googleor-glop -warmstart=0

FIGURE A.2. List of options from an execution in RAPOSa Cloud

Once the execution is set up, the tasks are added to the queue. This queue is an internal queue in the database that is later access by a bot running on CESGA to act on the pending tasks. The combinations of options and problems (each representing a call to RAPOSa) are splited into tasks that can be run on CESGA. Depending on the maximum time set up for the execution, the number of combinations in each task is selected, in a way that CESGA's tasks always have the same time limit of 6 hours, which makes the waiting time in CESGA's queue relatively small. This is one of the main advantages of this system, because even executions with a maximum

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

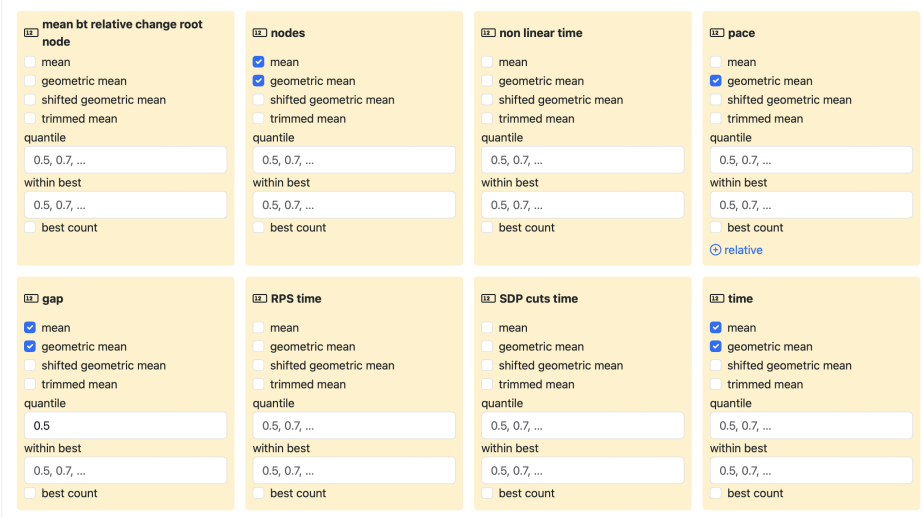


FIGURE A.3. Configuring performance measures for the tables in RAPOSa Cloud

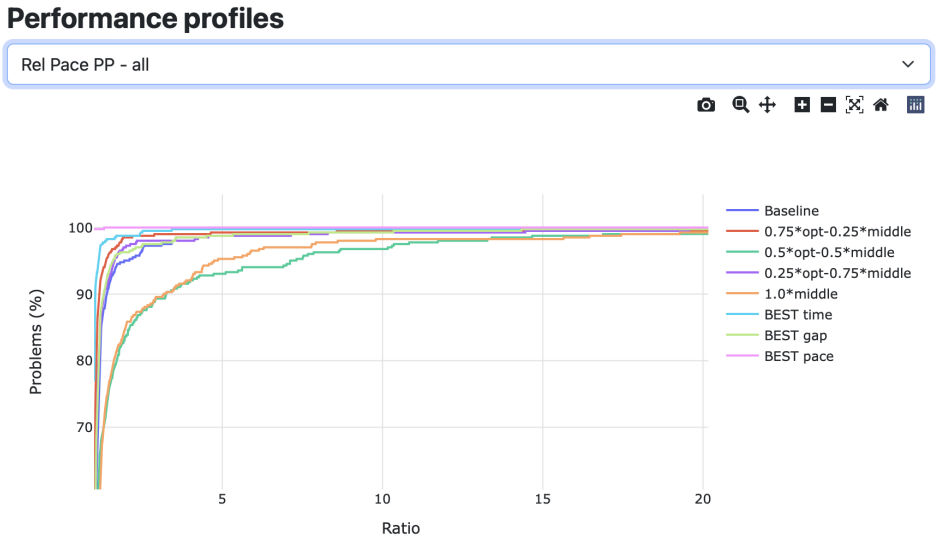


FIGURE A.4. Performance profile in RAPOSa Cloud

time of 1 hour can be run fast at CESGA, as they are splited into small tasks that don't have a big maximum time and therefore don't get postponed excessively by the queue system at CESGA.

Once the tasks are run, and CESGA's bot has uploaded the results to the database, we can generate some plots and tables to analyze the results obtained. We have a Python code that generates the tables we use in this thesis (the ones representing the running time, the gap, the pace, the number of nodes, etc.) and that can be extensively configured with different performance metrics, different aggregate measures (mean, quantiles, geometric means, shifted geometric means) and also with a calculation of the best that could be done if the optimal option

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

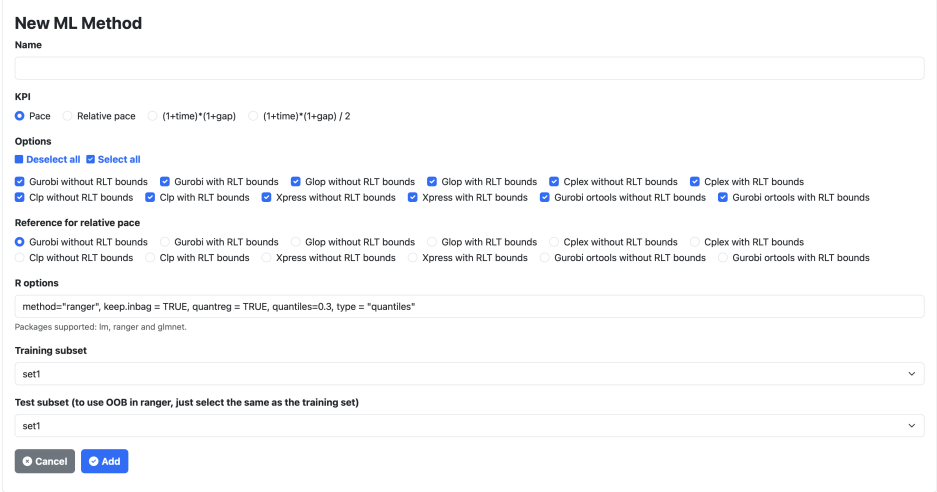


FIGURE A.5. Configuring the learning in RAPOSa Cloud

according to time, gap or pace is selected for each problem (this we use in Part 2 of the thesis to display the `Optimal` option in the tables). An example of the interface to select the performance measures to include in the tables can be seen on Figure A.3. This Python code is also capable of displaying performance profiles comparing the different options, as well as one-by-one comparisons for each of the performance metrics selected. An example of a performance profile generated in RAPOSa Cloud can be seen on Figure A.4.

Finally, the machine learning framework from Chapter 5 is incorporated into the interface, allowing the user to learn with respect to a chosen KPI, selecting the appropiate R options, and building an "option" for RAPOSa that can be displayed alongside the real options and compare its performance. This new "option" is just an option that copies the results of the configuration selected by the machine learning for each of the problems. The interface for configuring the learning can be seen in Figure A.5.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

APPENDIX B

Proofs of the convergence results of the RLT technique

LEMMA (Lemma 1.1). *Let $B \subseteq N$ be a subset of bounded variables in Ω. Let δ' be such that $1 \leq \delta' < \delta$ and let J_1, J_2 be two multisets such that $J_1 + J_2 \in N^{\delta'}$, $\text{supp}(J_2) \subseteq B$. Then, the bound-factor constraint $[F_{\delta'}(J_1, J_2)]_L \geq 0$ is implied by the bound-factor constraints $[F_\delta(J_3, J_4)]_L \geq 0$ with $J_3 + J_4 \in N^\delta$, $\text{supp}(J_4) \subseteq B$.*

PROOF. Given δ' satisfying $1 \leq \delta' < \delta$ and J_1, J_2 such that $J_1 + J_2 \in N^{\delta'}$, $\text{supp}(J_2) \subseteq B$, take $j \in B$ and define J_* as the multiset (B, μ_*) such that $\mu_*(j) = 1$ and $\mu_*(i) = 0$ for all $i \neq j$. Consider the constraints

$$[F_{\delta'+1}(J_1 + J_*, J_2)]_L \geq 0 \quad \text{and} \quad [F_{\delta'+1}(J_1, J_2 + J_*)]_L \geq 0.$$

It holds that

$$\begin{gathered}[F_{\delta'+1}(J_1 + J_*, J_2)]_L + [F_{\delta'+1}(J_1, J_2 + J_*)]_L = \\ [(x_j - l_j)F_{\delta'}(J_1, J_2)]_L + [(u_j - x_j)F_{\delta'}(J_1, J_2))]_L = \\ [x_j F_{\delta'}(J_1, J_2)]_L - l_j F_{\delta'}(J_1, J_2)]_L + u_j[F_{\delta'}(J_1, J_2)]_L - [x_j F_{\delta'}(J_1, J_2)]_L = \\ (u_j - l_j)[F_{\delta'}(J_1, J_2)]_L.\end{gathered}$$

Since $u_j - l_j \geq 0$, we have that the constraint $[F_{\delta'}(J_1, J_2)]_L \geq 0$ is implied by the constraints $[F_{\delta'+1}(J_1 + J_*, J_2)]_L \geq 0$ and $[F_{\delta'+1}(J_1, J_2 + J_*)]_L \geq 0$. If $\delta' + 1$ is not δ, for the two bound-factors involving J_* we would repeat the same process until δ is reached. □

COROLLARY (Corollary 1.2). *In the feasible region of $LP(\Omega)$, all X_J variables with $J \in B^\delta_-$ are bounded.*

PROOF. Let X_J be an RLT variable with $J \in B^\delta_-$. We prove that X_J has finite lower and upper bounds by induction on the cardinality of J.

Consider a multiset $J \in B^2$. Denote the variables in $\text{supp}(J)$ as j_1 and j_2, not necessarily different. By Lemma 1.1, the bound-factor constraint $[F_2(J, \emptyset)]_L \geq 0$ is implied by some bound-factor constraint in $LP(\Omega)$. Therefore, we have

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

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. A similar calculation for bound-factor constraint $[F_2(\{j_1\}, \{j_2\})]_L$ gives an upper bound for X_J.

Given $3 \leq \delta' \leq \delta$, suppose the statement holds for $J \in B^{\delta'}_-$, and let us prove if for $J \in B^{\delta'}$. Consider then $J \in B^{\delta'}$, and denote $\text{supp}(J) = \{j_1, \ldots, j_{\delta'}\}$, with the variables not necessarily different. The bound-factor constraint $[F_{\delta'}(J, \emptyset)]_L \geq 0$ is again implied by those in $LP(\Omega)$, and therefore we have

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

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

where $f(\boldsymbol{x}, \boldsymbol{X})$ is a linear function where all RLT variables involved are associated with monomials with degree smaller than δ', and therefore bounded by the induction hypothesis. Hence X_J has a finite lower bound. To obtain an upper bound, a similar argument can be applied using bound-factor constraint $[F_{\delta'}(J \setminus \{j_{\delta'}\}, \{j_{\delta'}\})]_L$. □

COROLLARY (Corollary 1.3). *Let $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ be a feasible solution of $LP(\Omega)$. If $\bar{x}_p = l_p$ for $p \in B$, then, for all $J \in B_-^{\delta-1}$, $\bar{X}_{J\cup\{p\}} = l_p \bar{X}_J$. Similarly, if $\bar{x}_p = u_p$ for $p \in B$, then, for all $J \in B_-^{\delta-1}$, $\bar{X}_{J\cup\{p\}} = u_p \bar{X}_J$.*

PROOF. We prove the result by induction on the cardinality of J. Take $J \in B^1$, being $\text{supp}(J) = \{q\}$. By Lemma 1.1, the following bound-factor constraints are implied by $LP(\Omega)$:

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

Consequently, $l_q(x_p - l_q) + l_p x_q \leq X_{\{q,p\}} \leq l_p x_q + u_q(x_p - l_p)$. Since $\bar{x}_p = l_p$, evaluating the inequality in $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$ we have that $\bar{\boldsymbol{X}}_{q,p} = l_p x_q$, i.e., $\bar{\boldsymbol{X}}_{J\cup\{p\}} = l_p \bar{\boldsymbol{X}}_J$.

Now, take $2 \leq t \leq \delta - 1$, assume that the result holds for $J \in B_-^{t-1}$, and consider $J \in B^t$. By Lemma 1.1, for some $q \in J$ we have

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

We have that $\prod_{j\in J\setminus\{q\}}(x_j - l_j) = \prod_{j\in J\setminus\{q\}} x_j + \phi(\boldsymbol{x})$, where $\phi(\boldsymbol{x})$ is a polynomial of degree less than $t-2$. Applying the linearization, we get

$$\begin{aligned} X_{J\cup\{p\}} - l_p X_J &\geq l_q \left(X_{(J\setminus\{q\})\cup\{p\}} - l_p X_{J\setminus\{q\}}\right) \\ &\quad + [l_p x_q \phi(\boldsymbol{x}) - x_p x_q \phi(\boldsymbol{x})]_L + [l_q x_p \phi(\boldsymbol{x}) - l_q l_p \phi(\boldsymbol{x})]_L \text{ and} \\ X_{J\cup\{p\}} - l_p X_J &\leq u_q \left(X_{(J\setminus\{q\})\cup\{p\}} - l_p X_{J\setminus\{q\}}\right) \\ &\quad + [l_p x_q \phi(\boldsymbol{x}) - x_p x_q \phi(\boldsymbol{x})]_L + [u_q x_p \phi(\boldsymbol{x}) - u_q l_p \phi(\boldsymbol{x})]_L. \end{aligned}$$

By the induction hypothesis, $\bar{\boldsymbol{X}}_{(J\setminus\{q\})\cup\{p\}} = l_p \bar{\boldsymbol{X}}_{J\setminus\{q\}}$. Therefore, since $\bar{x}_p = l_p$, evaluating the expressions in $(\bar{\boldsymbol{x}}, \bar{\boldsymbol{X}})$, both right hand sides become zero, and consequently $\bar{\boldsymbol{X}}_{J\setminus\{p\}} = l_p \bar{\boldsymbol{X}}_J$.

A similar argument can be used to prove the case with $\bar{x}_p = u_p$. □

LEMMA (Lemma 1.4). *Suppose `plainRLT` is applied to solve problem $PP(\Omega)$. Let k correspond to a relaxation $LP(\Omega^k)$ solved during the course of the algorithm. Then, $LB^k \leq \nu[PP(\Omega)]$.*

PROOF. Consider the iteration of `plainRLT` at which problem k is solved, and let UB be the upper bound of the algorithm at that iteration. We distinguish two cases.

Case 1. $UB = \nu[PP(\Omega)]$. Then, since subproblem k was not pruned by bounding, we have $LB^k < UB = \nu[PP(\Omega)]$.

Case 2. $UB > \nu[PP(\Omega)]$. Then, the optimal solution belongs to the feasible region of some of the problems in the queue Q. Thus, $\nu[PP(\Omega)] > LB$, the best that might be achieved by solving the problems in Q. Finally, since problem k is solved at the current iteration, we have that $LB^k = LB$ and, thus, $LB^k = LB < \nu[PP(\Omega)]$. □

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

USC
UNIVERSIDADE
DE SANTIAGO
DE COMPOSTELA

In the effort of developing algorithms for solving nonlinear optimization problems, most of the research is aimed at solving general nonlinear problems. However, there is a subclass of problems, polynomial optimization problems, that it's relevant as it provides an additional structure to the problems while still covering classes of problems often studied, such as continuous convex and nonconvex problems with quadratic costs and constraints or binary linear problems. In this thesis, we study this field of polynomial optimization and focus on three relevant aspects: solving the problem, using learning techniques to improve the performance of a solver and applying polynomial optimization techniques to a real-world problem in power network optimization.