Computational advances in polynomial optimization: RAPOSa, a freely available global solver

Brais González-Rodríguez[1,3] (ORCID) · Joaquín Ossorio-Castillo[3] · Julio González-Díaz[1,3] · Ángel M. González-Rueda[2] · David R. Penas[1,3] · Diego Rodríguez-Martínez[3]

Abstract

In this paper we introduce RAPOSa, a global optimization solver specifically designed for (continuous) polynomial programming problems with box-constrained variables. Written entirely in C++, RAPOSa is based on the Reformulation-Linearization (Sherali and Tuncbilek in J Glob Optim 103:225–249, 1992). We present a description of the main characteristics of RAPOSa along with a thorough analysis of the impact on its performance of various enhancements discussed in the literature, such as bound tightening and SDP cuts. We also present a comparative study with three of the main state-of-the-art global optimization solvers: BARON, Couenne and SCIP.

Keywords Global optimization · Polynomial programming · Reformulation-Linearization Technique (RLT)

1 Introduction

In this paper we introduce RAPOSa (**R**eformulation **A**lgorithm for **P**olynomial **O**ptimization —**Sa**ntiago), a new global optimization solver specifically designed for polynomial programming problems with box-constrained variables. It is based on the Reformulation-Linearization Technique [33], hereafter RLT, and has been implemented in C++. Although it is not open source, RAPOSa is freely distributed and available for Linux, Windows and MacOS. It can also be run from AMPL [18] and from NEOS Server [12]. The RLT-based scheme in RAPOSa solves polynomial programming problems by successive linearizations embedded into a branch-and-bound scheme. At each iteration, a linear solver must

✉ Brais González-Rodríguez
braisgonzalez.rodriguez@usc.es

1   Department of Statistics, Mathematical Analysis and Optimization, MODESTYA Research Group, University of Santiago de Compostela, Santiago, Spain

2   Department of Mathematics, MODES Research Group and CITIC, University of A Coruña, A Coruña, Spain

3   CITMAga (Galician Centre for Mathematical Research and Technology), Santiago de Compostela, Spain

be called, and RAPOSa has been integrated with a wide variety of linear optimization solvers, both open source and commercial, including those available via Google OR-Tools [28]. Further, auxiliary calls to nonlinear solvers are also performed along the branch-and-bound tree to improve the performance of the algorithm, and again both open source and commercial solvers are supported as long as they can be called from .nl files [19, 20]. More information about RAPOSa can be found at https://raposa.usc.es.

In conjunction with the introduction of RAPOSa, the other major contribution of this paper is to study the impact of different enhancements on the performance of the RLT. We discuss not only the individual impact of a series of enhancements, but also the impact of combining them. To this end, Sects. 4 and 5 contain computational analyses on the use of J-sets [13], warm starting of the linear relaxations, changes in the branching criterion, the introduction of bound tightening techniques [5, 6, 29] and the addition of SDP cuts [31], among others. Interestingly, RAPOSa incorporates a fine-grained distributed parallelization of the branch-and-bound core algorithm, which delivers promising speedups as the number of available cores increases.

The most competitive configurations of RAPOSa according to the preceding extensive analysis are then compared to three popular state-of-the-art global optimization solvers: BARON [30], Couenne [6] and SCIP [8]. The computational analysis is performed on two different test sets. The first one, DS-TS, is a set of randomly generated polynomial programming problems of different degree introduced in Dalkiran and Sherali [14] when studying their own RLT implementation: RLT-POS.[1] The second test set, MINLPLib-TS, contains the polynomial programming problems with box-constrained and continuous variables available in MINLPLib [9]. The main results can be summarized as follows: (i) In DS-TS, all configurations of RAPOSa clearly outperform BARON, Couenne and SCIP, with the latter performing significantly worse than all the other solvers and (ii) In MINLPLib-TS, differences in performance are smaller across solvers, with SCIP exhibiting a slightly superior performance. Importantly, the enhanced versions of RAPOSa are clearly superior in this test set to the baseline configuration.

The outline of the paper is as follows. In Sect. 2 we present a brief overview of the classic RLT scheme and different enhancements that have been introduced in recent years. In Sect. 3 we discuss some specifics of the implementation of RAPOSa and of the testing environment. In Sect. 4 we present some preliminary computational results, in order to define a configuration of RAPOSa that can be used as the baseline to assess the impact of the main enhancements, discussed in Sect. 5. In Sect. 6 we present the comparative study with BARON, Couenne and SCIP. Finally, we conclude in Sect. 7.

2 RLT: state of the art and main contributions

2.1 Brief overview of the technique

The Reformulation-Linearization Technique was originally developed in Sherali and Tuncbilek [33]. It was designed to find global optima in polynomial optimization problems of the

---

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

following form:

$$
\begin{aligned}
\text{minimize } & \phi_0(\boldsymbol{x}) \\
\text{subject to } & \phi_r(\boldsymbol{x}) \geq \beta_r, \quad r = 1, \dots, R_1 \\
& \phi_r(\boldsymbol{x}) = \beta_r, \quad r = R_1 + 1, \dots, R \\
& \boldsymbol{x} \in \Omega \subset \mathbb{R}^n.
\end{aligned}
\tag{1}
$$

where $N = \{1, \dots, n\}$ denotes the set of variables, each $\phi_r(\boldsymbol{x})$ is a polynomial of degree $\delta_r \in \mathbb{N}$, $\Omega = \{\boldsymbol{x} \in \mathbb{R}^n : 0 \leq l_j \leq x_j \leq u_j < \infty, \ \forall j \in N\} \subset \mathbb{R}^n$ is a hyperrectangle containing the feasible region, and the degree of the problem is defined as $\delta = \max_{r \in \{0, \dots, R\}} \delta_r$.

A multiset is a pair (S, p), in which S is a set and $p : S \to \mathbb{N}$ is a map that indicates the multiplicity of each element of S. We slightly abuse notation and use (N, δ) to denote the multiset of variables (N, p) in which $p(i) = \delta$ for each $i \in N$. For each multiset (N, p), its cardinality is defined by $|(N, p)| = \sum_{i \in N} p(i)$.

The RLT algorithm involves two main ingredients. First, the bound-factor constraints, given, for each pair of multisets J_1 and J_2 such that $J_1 \cup J_2 \subset (N, \delta)$ and $|J_1 \cup J_2| = \delta$, by

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

Note that any point in Ω satisfies all the bound-factor constraints. Second, the RLT variables, given, for each multiset $J \subset (N, \delta)$ such that $2 \leq |J| \leq \delta$, by

$$
X_J = \prod_{j \in J} x_j.
\tag{3}
$$

Each multiset J can be identified with a monomial. For instance, the multiset $J = \{1, 1, 2, 3, 4, 4\}$ refers to the monomial $x_1^2 x_2 x_3 x_4^2$. Therefore, to each monomial J one can associate different bound factors of the form $J = J_1 \cup J_2$, depending on which variables are used for the lower-bound factors and which ones for the upper-bound factors. Further, monomial $J = \{1, 1, 2, 3, 4, 4\}$ also defines the RLT variable X_{112344}.

The first step of the RLT algorithm is to build a linear relaxation of the polynomial problem (1). To do this, the polynomials of the original problem are linearized by replacing all the monomials with degree greater than 1 by their corresponding RLT variable (3). This linearization is denoted by $[\cdot]_L$. Furthermore, the linearized bound-factor constraints in (2) are added to get tighter linear relaxations:

$$
\begin{aligned}
\text{minimize } & [\phi_0(\boldsymbol{x})]_L \\
\text{subject to } & [\phi_r(\boldsymbol{x})]_L \geq \beta_r, \quad r = 1, \dots, R_1 \\
& [\phi_r(\boldsymbol{x})]_L = \beta_r, \quad r = R_1 + 1, \dots, R \\
& [F_\delta(J_1, J_2)]_L \geq 0, \ J_1 \cup J_2 \subset (N, \delta), |J_1 \cup J_2| = \delta \\
& \boldsymbol{x} \in \Omega \subset \mathbb{R}^n.
\end{aligned}
\tag{4}
$$

Note that if constraints in (3) are added to the linear relaxation (4), the resulting problem is equivalent to problem (1). The next step of the RLT algorithm is to solve the linear relaxation in (4) to obtain a lower bound of the original polynomial problem. Next, a branch-and-bound scheme is used to find the global optimum of the polynomial problem. Since a solution of the linear relaxation that satisfies constraints in (3) is feasible to problem (1), the branching rule is usually based on violations of these constraints, which are referred to as RLT-defining identities. The convergence of this scheme to a global optimum is proven in Sherali and Tuncbilek [33].

Along the branch-and-bound tree, the RLT algorithm obtains, and gradually increases, lower bounds for the optimal solution of the minimization problem by solving the linear relaxations. It obtains upper bounds when the solution of a linear relaxation is feasible in the original problem. The absolute and relative differences between the lower and upper bounds lead to the so called absolute and relative optimality gaps. The goal of the algorithm is to close these gaps to find the global optimum of the polynomial problem. Thus, stopping criteria often revolve around thresholds on the optimality gaps.

2.2 Enhancements of the original RLT algorithm

In this section we briefly discuss six enhancements of the basic RLT algorithm. All of them are part of the current implementation of RAPOSa and its impact on the performance of the algorithm is thoroughly analyzed in Sects. 4 and 5.

2.2.1 J-sets

This enhancement was introduced in Dalkiran and Sherali [13], where the authors prove that it is not necessary to consider all the bound-factor constraints in the linear relaxation (4). Specifically, they prove two main results. The first one is that convergence to a global optimum is ensured even if only the bound-factor constraints associated with the monomials that appear in the original problem are included in the linear relaxation. The second result is that convergence to a global optimum is also ensured without adding the bound-factor constraints associated with monomials $J \subset J'$, provided the bound-factor constraints associated with J' are already incorporated. Equipped with these two results, the authors identify a collection of monomials, which they call J-sets, such that convergence to a global optimum is still guaranteed if only the bound-factor constraints associated with these monomials are considered.

The use of J-sets notably reduces the number of bound-factor constraints in the linear relaxation (4) (although it still grows exponentially fast as the size of the problem increases). The main benefit of this reduction is that it leads to smaller LP relaxations and, hence, the RLT algorithm requires less time in each iteration. The drawback is that the linear relaxations become less tight because they have less constraints and more iterations may be required for convergence. Nevertheless, practice has shown that the approach with J-sets is clearly superior. We corroborate this fact in the numerical analysis Sect. 4.1.

2.2.2 Use of an auxiliary local NLP solver

Most branch-and-bound algorithms for global optimization rely on auxiliary local solvers and the RLT scheme can also profit from them, as already discussed in Dalkiran and Sherali [14]. They proposed to call the nonlinear local solver at certain nodes of the branch-and-bound tree. In each call, the nonlinear local solver is provided with an initial solution, which is the one associated to the lower bound of the RLT algorithm at that moment.

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

2.2.3 Products of constraint factors and bound factors

This enhancement was already mentioned in Sherali and Tuncbilek [33] when first introducing the RLT technique for polynomial programming problems. It consists of defining tighter linear programming relaxations by strengthening *constraint factors* of the form $\phi_r(\boldsymbol{x}) - \beta_r \geq 0$ of degree less than δ. More precisely, one should take products of these constraint factors and/or products of bound factors in such a way that the resulting degree is no more than δ. Similar strengthenings can also be associated to equality constraints $\phi_r(\boldsymbol{x}) = \beta_r$ of degree less than δ, by multiplying them by variables of the original problem.

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

2.2.4 Branching criterion

The design of a branch-and-bound algorithm requires to properly define different components of the algorithm. The most widely studied ones are the search strategy in the resulting tree, pruning rules and branching criterion; refer, for instance, to Achterberg et al. [1] and Morrison et al. [26]. In Sect. 5.4 we focus on the latter of these components. More precisely, we study the impact of the criterion for the selection of the branching variable on the performance of the RLT technique.

2.2.5 Bound tightening

Bound tightening techniques are at the core of most global optimization algorithms for nonlinear problems [5, 6, 29]. These techniques allow to reduce the search space of the algorithm by adjusting the bounds of the variables of the problem. Two main approaches have been discussed in the literature: (i) Optimality-based bound tightening, OBBT, in which bounds are tightened by solving a series of relaxations of minor variations of the original problem and (ii) Feasibility-based bound tightening, FBBT, in which tighter bounds are deduced directly by exploring the problem constraints. Since OBBT is computationally demanding while FBBT is not, combined schemes are often used, under which OBBT is only performed at the root node and FBBT is performed at every node of the branch-and-bound tree.

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

2.2.6 SDP cuts

This enhancement is introduced in Sherali et al. [31] and consists of adding specific constraints, called positive semidefinite cuts, SDP cuts, to the linear relaxations generated along the branch-and-bound tree. These constraints are built as follows. First, a matrix of the form $\boldsymbol{M} = [\boldsymbol{y}\boldsymbol{y}^T]$ is defined, where $\boldsymbol{y}$ can be any vector defined using variables and products

of variables of the original problem. It holds that matrix M is positive semidefinite in any feasible solution. Therefore, given a solution of the current linear relaxation, we can evaluate M at this solution, obtaining matrix $\bar{M}$. If this matrix is not positive semidefinite, then we can identify a valid linear cut to be added to the linear relaxation. More precisely, if there is a vector $\boldsymbol{\alpha}$ such that $\boldsymbol{\alpha}^T \bar{M} \boldsymbol{\alpha} < 0$, then constraint $\boldsymbol{\alpha}^T M \boldsymbol{\alpha} \geq 0$ is added to the linear relaxation.

In Sherali et al. [31] this process is thoroughly explained and several strategies are discussed, such as different approaches to take vector y, different methods to find $\boldsymbol{\alpha}$ and the number of cuts to add in each iteration.

3 Implementation and testing environment

3.1 Implementation

Each execution of `RAPOSa` leans on two solvers: one for solving the linear problems generated in the branch-and-bound process and one to compute upper bounds as mentioned in Sect. 2.2.2. The default (free) configuration of `RAPOSa` runs with `Glop` [2] and `Ipopt` [35], although the best results are obtained using commercial solver `Gurobi` [23] instead of `Glop`. Currently, `RAPOSa` supports the following solvers:

- Linear solvers: `Gurobi` [23], `Glop` [2], `Clp` [17] and `lp_solve` [7].[2]
- Nonlinear local solvers: `Ipopt` [35], `Knitro` [11], `MINOS` [27] and `CONOPT` [16].

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

Importantly, the user does not need to explicitly generate `.nl` files to execute `RAPOSa`, since it can also be run from an `AMPL` interface [18].[3] Moreover, `RAPOSa` can also be executed on NEOS Server [12].

3.2 The testing environment

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

Regarding the test sets, we use two different sets of problems. The first one is taken from Dalkiran and Sherali [14] and consists of 180 instances of randomly generated polynomial programming problems of different degrees, number of variables and density.[4] The second test set comes from the well known benchmark MINLPLib [9], a library of Mixed-Integer

---

[2] The connection with Glop and Clp was implemented using Google OR-Tools [28].

[3] The web site https://raposa.usc.es contains a short tutorial that explains how to interact with `RAPOSa` either directly with `.nl` files or through `AMPL`.

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

Nonlinear Programming problems. We have selected from MINLPLib those instances that are polynomial programming problems with box-constrained and continuous variables, resulting in a total of 168 instances. Hereafter we refer to the first test set as DS-TS and to the second one as MINLPLib-TS.[5]

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

4 Preliminary results and RAPOSa's baseline configuration

The main goal of this section is to add to RAPOSa's RLT basic implementation a minimal set of enhancements that make it robust enough to define a baseline version on which to assess, in Sect. 5, the impact of the rest of the enhancements. First, in Sect. 4.1 we show that the inclusion of both the J-sets enhancement and the auxiliary nonlinear solver are crucial in order to be able to get a competitive solver when tackling the problems in DS-TS and MINLPLib-TS. Then, in Sect. 4.2 we present a comparison of the performance of different LP solvers and thereafter all new enhancements are validated and tested on the best performing one. Last, but not least, in Sect. 4.3 we present the results of a parallel version of RAPOSa's RLT implementation, to illustrate the potential of improvement of this type of branch-and-bound algorithms when run on multi-core processors. Yet, in order to provide fair comparisons in the rest of the paper, particularly in Sect. 6, these parallelization capabilities won't be used beyond Sect. 4.3.

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

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

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

Time   Geometric mean time,[6] but disregarding those instances solved by all configurations in less than 5 s and also those not solved by any configuration within the time limit. In brackets we show the remaining number of instances.

Gap   Geometric mean gap, but with the following considerations: (i) instances solved by all configurations under study are discarded, (ii) instances for which no configuration could return a gap after the time limit are also discarded and (iii) when a configuration is not able to return a lower or upper bound after the time limit, we assign to it a relative optimality gap of 10^5. In brackets we show the remaining number of instances.

Table 1 Impact of J-Sets and auxiliary nonlinear solver

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

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

Moreover, for each table discussed in the text there are two associated performance profiles [15] for each test set although, for the sake of brevity, most of them have been relegated to "Appendix A". The first performance profile is for the running times and the second one for the relative optimality gaps. They contain, respectively, the instances involved in the computations of the geometric mean times and the geometric mean gaps as described above. In the x-axis we represent ratios of running times or relative optimality gaps, while in the y-axis we represent the percentage of instances in which the corresponding configuration has a ratio lower than the value on the x-axis. For each instance, the ratios are computed dividing running times or relative optimality gaps of each configuration by the best configuration in that instance.[7]

4.1 J-sets and nonlinear solver

We start by jointly evaluating the impact of the introduction of J-sets, discussed in Sect. 2.2.1, and the use of an auxiliary local NLP solver (NLS). Regarding the latter, `Ipopt` is run at the root node and whenever the total number of solved nodes in the branch-and-bound tree is a power of two. We tried other strategies, but we did not observe a significant impact on the resulting performance. In Table 1 we present the results of four different configurations: (i) `RAPOSa`'s RLT basic implementation (No J-sets, No NLS), (ii) the inclusion of the auxiliary nonlinear solver (No J-sets), (iii) the inclusion of J-sets (No NLS) and (iv) the inclusion of both the auxiliary nonlinear solver and J-sets (With J-sets and NLS).

The results in Table 1 show that J-sets have a huge impact on the performance of the RLT technique in both tests sets. It does not matter whether we assess their impact with respect to `RAPOSa`'s RLT basic implementation or with respect to the version that already incorporates the nonlinear solver: there are dramatic gains in all dimensions. What is a bit surprising is that the configurations with J-sets do even increase the average number of nodes explored in problems solved by all configurations. Since the version without J-sets leads to tighter

---

[7] It is worth noting that the reliability of performance profiles is sometimes limited when comparing more than two configurations/solvers [22]. Yet, we think that, when complemented with the numeric results in the tables, they help to obtain a clearer image of the numerical analysis developed in this paper.

relaxations, one would expect to observe a faster increase in the lower bounds and a reduction of the total number of explored nodes (at the cost of higher solve time at each node). A careful look at the individual instances reveals that the number of explored nodes turns to be quite close for most instances. Yet, there are a few "outliers" that required many more nodes to be solved for the version without J-sets, producing a large impact on the average and also in the geometric average. It might be worth studying further whether these outliers appeared in the configuration without J-sets by coincidence or if there is some structural reason that leads to this effect.

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

We move now to the impact of the inclusion of a local solver. Again, Table 1 shows that there is a huge impact on the performance of the RLT technique in both test sets. Performance improves again in all dimensions, and specially so at closing the gap, since the number of instances for which some bound is missing goes down, with respect to `RAPOSa`'s RLT basic implementation, from 144 to 0 in DS-TS and from 55 to 4 in MINLPLib-TS. Similarly, with respect to the version that already uses J-sets, the number goes down from 74 to 0 in DS-TS and from 38 to 2 in MINLPLib-TS. We have run similar computational tests with different local NLP solvers and the results are quite robust. Therefore, the specific choice of local solver does not seem to have a significant impact on the final performance of the RLT technique.

4.2 Different LP solvers

The results of Table 1 in the preceding section where obtained using the commercial solver `Gurobi` [23] for the linear relaxations. We now check to what extent the chosen LP solver can make a difference in the performance of the RLT technique. To this end, we rerun the executions of the version of `RAPOSa` with J-sets and the auxiliary nonlinear solver but with two open source linear solvers: `Clp` [17] and `Glop` [2]. The results in Table 2 show that `Gurobi`'s performance is superior to both `Clp` and `Glop`. On the other hand, the two open source solvers are close to one another. These is confirmed by the performance profiles in Fig. 6 in "Appendix A". Thus, for the remainder of this paper, all the configurations of `RAPOSa` are run using `Gurobi` as the solver for the linear relaxations.

Table 2 Impact of different linear solvers

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

4.3 Parallelized RLT

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

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

Figure 1 shows the evolution of the speedup with the number of cores for DS-TS.[8] In the x-axis we represent the number of cores (master+slaves) used by `RAPOSa`, while the y-axis shows the box plot of the speedup on the 180 DS-TS instances. The red line describes the ideal speedup: the number of workers. As can be seen, the scalability of the speedup obtained by the parallel version of `RAPOSa` is generally good, close to the ideal one when the number of cores is small (3 and 5) or with a reasonable performance with a larger number of cores (9 and 17). Furthermore, the speedup does not seem to depend on the linear solver (`Gurobi` or `Glop`) used by `RAPOSa`.

## 5 Computational analysis of different enhancements

In this section we present a series of additional enhancements of the basic implementation of the RLT technique and try to assess not only the individual impact of each of them, but also the aggregate impact when different enhancements are combined. In order to do so, we build upon a baseline version of `RAPOSa` that is used as the reference for the analysis, with the different enhancements being added to this baseline. Given the results in the preceding section, this configuration is set to use J-sets, `Ipopt` as the auxiliary local NLP solver,[9] and `Gurobi` as the linear one. No parallelization is used.

5.1 Warm generation of J-sets

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

---

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

[9] As we have already mentioned, we have seen in our numerical experiments that the performance of `RAPOSa` is not very sensitive to the chosen local NLP solver.
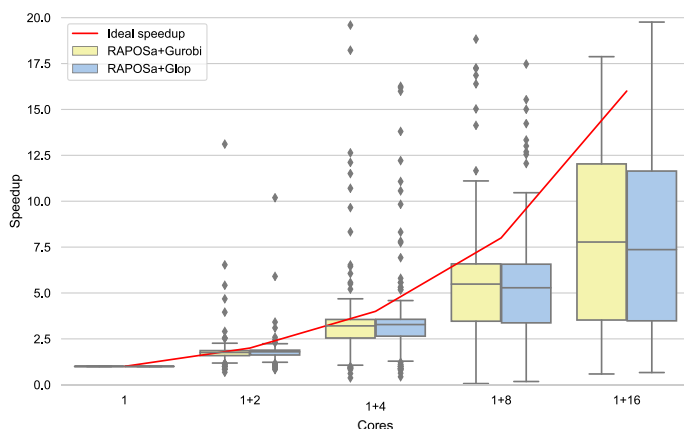
Fig. 1 Speedup of the parallel version of `RAPOSa` in DS-TS

Table 3 Warm generation of J-sets and warm start on LP Solver

Test Set		Baseline	Warm J-sets			Baseline	Warm start
DS	solved (136/180)	127	136	solved (131/180)	127	129
	gap $= \infty$ (0/180)	0	0	gap $= \infty$ (0/180)	0	0
	geom. time (75)	87.78	-19.66%	geom. time (69)	79.53	-26.82%
	geom. gap (53)	0.01	-45.38%	geom. gap (55)	0.01	$+56.17\%$
	geom. nodes (127)	137.02	0.00%	geom. nodes (125)	136.70	$+1.05\%$
MINLPLib	solved (100/168)	99	100	solved (100/168)	99	100
	gap $= \infty$ (5/168)	6	5	gap $= \infty$ (6/168)	6	7
	geom. time (20)	51.81	-28.31%	geom. time (20)	51.66	-27.91%
	geom. gap (64)	0.65	-18.03%	geom. gap (63)	0.55	$+10.40\%$
	geom. nodes (99)	110.06	0.00%	geom. nodes (99)	110.06	$+1.10\%$

(a) Warm generation of J-Sets.　　　　　　　　(b) Warm start on LP solver.

It is important to highlight that there are two running times involved in the process. One is the running time used to generate the bound-factor constraints, which is higher in the case that `RAPOSa` regenerates all the bound-factor constraints. The other running time is the one used to identify the bound-factor constraints that change between the parent and the child node. This time is non-existent in the case that `RAPOSa` regenerates all the bound-factor constraints. As one could expect, the 0.00% percentage of improvement in the last row of Table 3a reflects the fact that the warm generation of J-sets has no impact on the resulting tree.

We can see in Table 3a that the warm generation of J-sets notably improves the performance of `RAPOSa`, which is also convincingly illustrated by the performance profiles in Fig. 7 in "Appendix A".

5.2 Warm start on LP solver

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

Table 3b shows the impact of warm start with respect to the baseline configuration.[10] Differently from the preceding enhancements, the results are somewhat divided now. Warm start reduces the running time in solved instances but, at the same time, the gap in the unsolved ones seems to deteriorate. This suggests that warm start may perform better in relatively easy instances, but not so well in instances that were not solved within the time limit.

5.3 Products of constraint factors and bound factors

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

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

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

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

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

Table 4 shows that the "More in common" approach performs slightly better in DS-TS and slightly worse in MINLPLib-TS. Interestingly, the "Less in common" approach seems to lead to slight improvements in both test sets, so it may be worth to study its impact when combined with other enhancements.

5.4 Branching criterion

In the baseline configuration, we follow the approach in Sherali and Tuncbilek [33] and choose the branching variable involved in a maximal violation of RLT-defining identities. More precisely, given a solution $(\bar{X}, \bar{x})$ of the linear relaxation, we branch on a variable $i \in \operatorname{argmax}_{j \in N} \theta_j$, where θ_j is defined as:

$$\theta_j = \max_{J \subset (N, \delta) : |J| < \delta} \{ |\bar{X}_{J \cup \{j\}} - \bar{x}_j \bar{X}_J| \}. \tag{5}$$

---

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

Table 4 Products of constraint factors and bound factors

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

In a more recent paper, Dalkiran and Sherali [13] apply a slightly more sophisticated criterion for variable selection, in which the maximum in the above equation is replaced by a sum and also the violations associated to each variable are weighted by the minimum distance of the current value of the variable to its lower and upper bound. Here we follow a similar approach and study different criteria, where θ_j is of the form:

$$\theta_j = \sum_{J \subset (N,\delta):\, |J|<\delta} w(j,J) \left| \bar{X}_{J\cup\{j\}} - \bar{x}_j \bar{X}_J \right|, \tag{6}$$

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

Constant weights $\quad w(j, J) = 1$ for all j and J. This corresponds with the baseline configuration when maximums are taken in Eq. (6), thus recovering Eq. (5). Otherwise, if sums are considered we get criterion named "Sum" in Table 5.

Coefficients $\quad w(j, J)$ corresponds with the sum of the absolute values of the coefficients of monomial $J \cup \{j\}$ in the problem constraints.

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

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

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

Table 5 contains the results for the different criteria we have just described. Except for the baseline, which uses the maximum as in Eq. (5), all other criteria use sums as in Eq. (6); the reason being that criteria based on sums have shown to be remarkably superior to their "maximum" counterparts. In particular, we can see in the first two columns of the table that, just by replacing the maximum with the sum in the original branching criterion, the geometric

Table 5 Branching criterion

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

means of the computing time and gap get divided by two in MINLPLib-TS. In general, all criteria based on sums perform notably better than the original one. Arguably "Sum" and "Var. range" are the two most competitive ones and, by looking at the performance profiles in Fig. 10 in "Appendix A", it seems that "Var. range" is slightly superior, which goes along the lines of the approach taken in Sherali et al. [31].[11]

5.5 Bound tightening

We study the effect of different bound tightening strategies. More precisely, Table 6 represents the following approaches, along with some natural combinations of them:

OBBT root node   OBBT is run on the linear relaxation at the root node. OBBT is quite time consuming, so we limit its available time so that it does not use more than 20% of the total time available to RAPOSa. Since this sometimes implies that bound tightening is not applied to all variables, we prioritize tightening the upper bounds and also prioritize variables with larger ranges.

Linear FBBT   FBBT is run at all nodes on the linearized constraints of the original problem.[12]

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

The results in Table 6 show a mild improvement on DS-TS and a very large impact on MINLPLib-TS. The fact that bound tightening has a relatively small impact on DS-TS was expected, since the generation procedure for the random instances in this test set already leads to relatively tight bounds. We can see that the nonlinear FBBT is superior to both the linear FBBT and the OBBT. Overall, the best configuration is the one combining OBBT at the root node with nonlinear FBBT at all nodes, which is a standard approach in well established global solvers. We also run some tests combining OBBT with the execution of both FBBT schemes at all nodes but, while this lead to a reduction on the number of explored nodes, this reduction did not compensate for the additional computational overhead of running two

---

[11] In Sherali et al. [31] their weights were of the form $w(j, J) = \min\{\bar{u}_j - \bar{x}_j, \bar{x}_j - l_j\}$, which we tried as well, but they were outperformed by the ones shown in Table 5. Further, Sherali et al. [31] also included addends of the form $\left| \prod_{k \in J \cup \{j\}} \bar{x}_k - \bar{x}_j \bar{X}_J \right|$. In our experiments we observed no benefit from the inclusion of these terms.

[12] Our C++ FBBT implementation builds upon the following Python implementation https://github.com/Pyomo/pyomo/tree/main/pyomo/contrib/fbbt, which is part of Pyomo's environment [10]. We also run some computational experiments using the bound tightening procedures included in Couenne [6], but the results were slightly worse.

Table 6 Bound tightening

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

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

5.6 SDP cuts

The last enhancement we study is the introduction of SDP cuts to tighten the linear relaxations. As discussed in Sect. 2.2.6, the main choice of this approach is the vector y that is then used to define matrix $M = [y y^T]$. Importantly, the resulting cuts will involve products of the different components of y and, since we are using J-sets, we should carefully define vector y so that the resulting cuts do not lead to the inclusion of new RLT variables (which in turn would require to include additional bound factor constraints) and increase the solving time of the linear relaxations.

To minimize the impact of the above issue we proceed as follows. Recall that J-sets correspond with maximal monomials with respect to set inclusion. Then, given a maximal monomial J, we define a vector y_J composed of the variables included in monomial J. Using these y_J vectors ensures that the set of maximal monomials will not increase significantly after introducing SDP cuts.[13] We are now ready to fully describe the different approaches we have studied regarding SDP cuts, which are partially inspired in the comprehensive study developed in Sherali et al. [31], to which the interested reader is referred for a deeper discussion and motivation.

First, for each y_J vector we consider three possibilities to define matrix M: (i) Taking y_J itself, (ii) expanding it to vector $(1, y_J)$ and (iii) expanding it to a vector of the form $(1, y_J, \ldots)$ in which, if possible, products of the variables in J are added while keeping under control the new RLT variables required by these additional products and without getting any element in M with a degree larger than δ.

For each of the above three possible definitions of the y vector, we proceed as follows: (i) By default, SDP cuts are applied in all nodes and they are inherited "forever", (ii) in order to save computational time in the computation of the α vectors, the corresponding M

---

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

Table 7 SDP cuts

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

matrix is divided in 10×10 overlapping submatrices (each matrix shares its first 5 rows with the preceding one), (iii) for each eigenvector with a negative eigenvalue, we add the corresponding cut and (iv) the procedure is repeated for each maximal monomial.

Table 7 shows the results of approaches $\boldsymbol{y}_J$, $(1, \boldsymbol{y}_J)$ and $(1, \boldsymbol{y}_J, \ldots)$. All of them seem to improve the performance of the algorithm, except for the running times in DS-TS, with $(1, \boldsymbol{y}_J)$ being the best of the three. On the other hand, vectors $(1, \boldsymbol{y}_J)$ and $(1, \boldsymbol{y}_J, \ldots)$ perform very similarly in MINLPLib-TS, the reason being that most problems in this test set are quadratic and these two vectors, by construction, coincide for quadratic (and for cubic) problems.

Given the above results, we carried out some additional experimets with vector $(1, \boldsymbol{y}_J)$. For instance, the last two columns in Table 7 represent the results when considering that cuts are only inherited to child nodes (Inh-1) and that they are also inherited to grandchildren (Inh-2). The performance of the latter is comparable with the one with full inheritance, but no significant gain is observed. Additionally, we also studied the impact of running cycles of the form "solve → add cuts → solve → add cuts…" at each node before continuing with the branching, but in our experiments they had a detrimental effect. Similarly, we also tested configurations in which cuts were added only in nodes at prespecified depths of the branch and bound tree, such as nodes whose depth is a multiple of 10, but performance also worsened.

5.7 Combining different enhancements: RAPOSa's best configuration

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

Products of constraint factors and bound factors   We take the "less in common" approach.
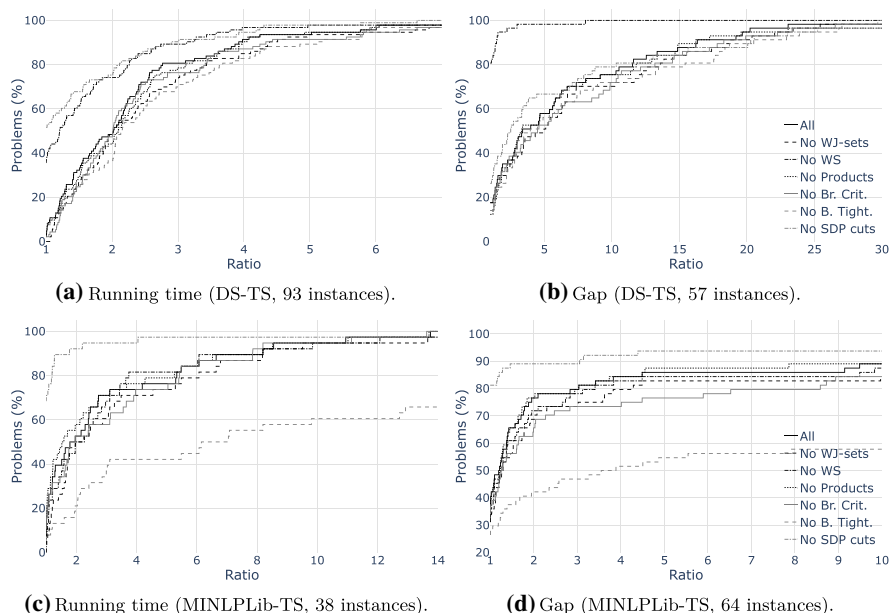
Branching criterion	We consider the criterion based on variable ranges.
Bound tightening	We consider OBBT at the root node and nonlinear FBBT at all nodes.
SDP cuts	We consider the approach with vector $(1, \mathbf{y}_J)$, with cuts being generated in all nodes and inherited forever.

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

The impact of the remaining three enhancements is somewhat mixed. Warm starting the linear solver seems to be beneficial in MINLPLib-TS, but not in DS-TS. The situation gets reversed for the products of constraint factors and bound factors, which slightly improve performance in DS-TS but slightly reduce it in MINLPLib-TS. Finally, we have that SDP cuts, when added on top of the other enhancements, seem to deteriorate performance. This is somewhat surprising and is definitely a direction for further research. Given the promising behavior of SDP cuts in Table 7, it would be important to understand why they have a substantial negative impact when combined with the other enhancements.

Table 8 Individual impact of the different enhancements

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

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

(a) Running time (DS-TS, 93 instances).

(b) Gap (DS-TS, 57 instances).

(c) Running time (MINLPLib-TS, 38 instances).

(d) Gap (MINLPLib-TS, 64 instances).

Fig. 2 Performance profiles for the impact of the different enhancements

In Fig. 2 we represent the performance profiles associated to all the configurations we have just discussed, with the exception of the configurations without J-sets and without nonlinear solver since, given their particularly bad performance, would distort the resulting plots. The performance profiles confirm what we have already seen in Table 8. The version without warm start of the LP solver is the best one in terms of optimality gap of the difficult instances in DS-TS, whereas the version without SDP cuts is the best one in the other three performance profiles. The superior performance of this version is specially good in MINLP-TS. The performance profiles also show that, setting aside the enhancements involving J-sets and the auxiliary nonlinear solver, the highest impact comes from the bound tightening, specially in MINLP-TS, where the configuration without this enhancement falls clearly behind the rest.

6 Overall performance and comparison with other solvers

In this section we present a comparison between `RAPOSa`, `BARON`, `Couenne` and `SCIP` on the instances of DS-TS and MINLPLib-TS. Ideally, we would have liked to include `RLT-POS` [14] in the analysis, since its underlying RLT implementation has been one of the main sources of inspiration for `RAPOSa`. Unfortunately, `RLT-POS` is not publicly available. Instead, we present in Table 9 a comparison of the enhancements included in both `C++` implementations of the RLT scheme, which shows that none of them dominates the other in terms of enhancements.

The "Reduced RLT" enhancement consists of a series of RLT-based reformulations introduced in [32] for polynomial optimization problems containing linear equality constraints. The idea is to rely on the basis of the associated linear systems to reduce the size of the linear relaxations. Different variants of this approach are implemented in `RLT-POS`. One

Table 9 Comparison of the functionalities of `RLT-POS` and `RAPOSa`

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

additional difference between `RLT-POS` and `RAPOSa` is that the former contains a heuristic which, depending on some underlying features of the problem at hand such as the degree, the density and the number of equality constraints, chooses a specific configuration of the different features. The computational study in [14] compares `RLT-POS` with `BARON`, `Couenne` and also with the SDP based solver SparsePOP [34]. The latter turned out to be the least competitive, whereas the performance of `RLT-POS` was notably superior to `Couenne` and slightly superior to `BARON` in DS-TS. These results are comparable to the ones we report below for `RAPOSa`. As far as the computational study is concerned, one of the additions of the current paper is that the analysis is also developed for MINLPLib-TS, which contains a wide variety of instances coming from real applications, and not just randomly generated instances as DS-TS.[14]

We move now to the comparison, for the instances in both DS-TS and MINLPLib-TS, of `RAPOSa` with three of the most popular solvers for finding global optima of nonlinear programming problems: `BARON`, `Couenne` and `SCIP`. All solvers have been run taking as stopping criterion that the relative or absolute gap is below the threshold 0.001 and with a time limit of 1 h in each instance. `RAPOSa` was run with two different configurations: (i) the baseline version in Sect. 5 (with J-sets and nonlinear solver) and (ii) the version that looked superior from the analysis in Sect. 5.7 (all the enhancements except the use of SDP cuts). It is worth mentioning the auxiliary solvers used by each of the global solvers involved in the comparison. All solvers use `Ipopt` as the auxiliary nonlinear solver. Regarding the linear solver, `RAPOSa` uses `Gurobi`, whereas `BARON`, `Couenne` and `SCIP` use `Clp`. Given the superior performance of `Gurobi` with respect to `Clp` reported in Sect. 4.2, it may be that the linear solver is giving a slight edge to `RAPOSa`. Yet, this is not straightforward to assess, since the solution of linear subproblems is not equally critical for all solvers, since `BARON` and `SCIP`, for instance, heavily rely on nonlinear (convex) relaxations whereas all relaxations solved by `RAPOSa` and `Couenne` are indeed linear.

Table 10 contains two summaries of results, one for each test set. First, we can see that, in DS-TS, both configurations of `RAPOSa` solved more problems than the other solvers. For this test set, the version of `RAPOSa` with no SDP cuts clearly dominates all others, not only in the number of solved problems, but also in running times and optimality gaps. At the other end, we see that `SCIP` falls clearly behind `BARON` and `Couenne` in DS-TS. Regarding MINLPLib-TS, `SCIP` is the solver that performs best. The behavior of the best version of `RAPOSa` is comparable to that of `BARON` and `Couenne`, falling behind in running times but being superior in optimality gaps. This suggests that `RAPOSa` may be particularly effective for difficult instances, a hypothesis that the explore more deeply in the performance profiles below. It is worth noting that `BARON`, `Couenne` and `SCIP` have been tested for years on MINLPLib-TS instances and, thus, some of these solvers' enhancements may have been designed to address weaknesses identified on them. Last, we can also compare the

---

[14] With respect to [14], also the thorough analysis enhancement by enhancement in Sects. 4 and 5 represents a novel contribution.

Table 10 Comparative between different configurations of `RAPOSa` and other solvers

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

(a) Running time (DS-TS, 117 instances).

(b) Gap (DS-TS, 82 instances).

(c) Running time (MINLPLib-TS, 50 instances).

(d) Gap (MINLPLib-TS, 85 instances).
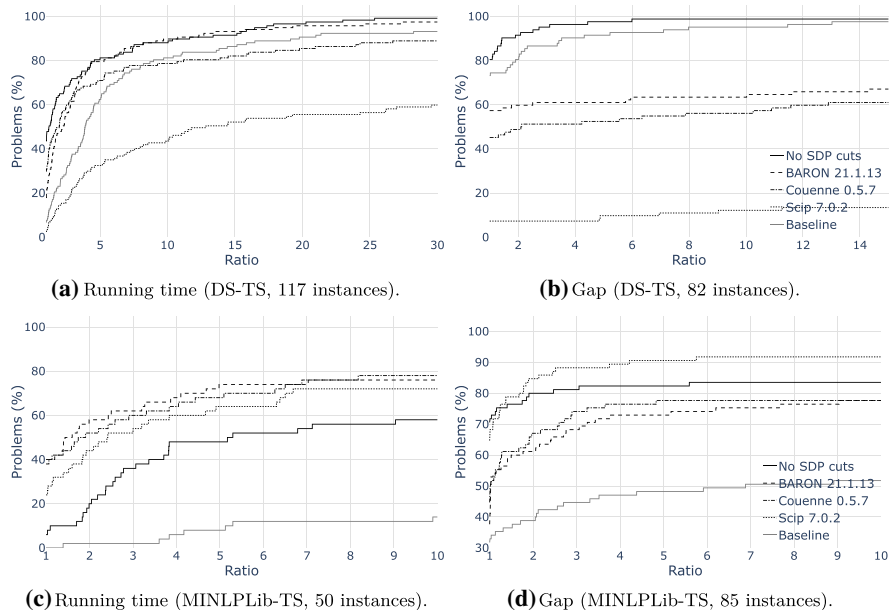
Fig. 3 Performance profiles to compare `RAPOSa` with other solvers

performance of the version of `RAPOSa` with all enhancements except SDP cuts with the version with just J-sets and the auxiliary nonlinear solver. The aggregate improvement in performance is remarkable, specially in MINLPLib-TS.

Figure 3 contains the performance profiles associated to the results reported in Table 10, which confirm the previous conclusions. It is worth emphasizing once again the significant gain in performance exhibited by `RAPOSa` after the inclusion of the enhancements discussed in this paper. This is specially relevant in MINLPLib-TS since, with these changes, `RAPOSa` goes from being the worst solver to being highly competitive with the state-of-the-art solvers.

The results in Table 10 and Fig. 3 both suggest that the performance of `RAPOSa` may be particularly good in the most difficult instances of both test sets, since it is more competitive on the optimality gaps in instances not solved by all instances than on the running times solved by all instances. Indeed, for DS-TS, even the baseline version of `RAPOSa` dominates `BARON`, `Couenne` and `SCIP` in terms of optimality gap. To further explore this insight, in Fig. 4 we represent the performance profiles associated to the instances that no solver managed to solve within the time limit of 1 h. They seem to confirm that `RAPOSa` becomes

(a) Gap (DS-TS, 14 instances).   **(b)** Gap (MINLPLib-TS, 38 instances).
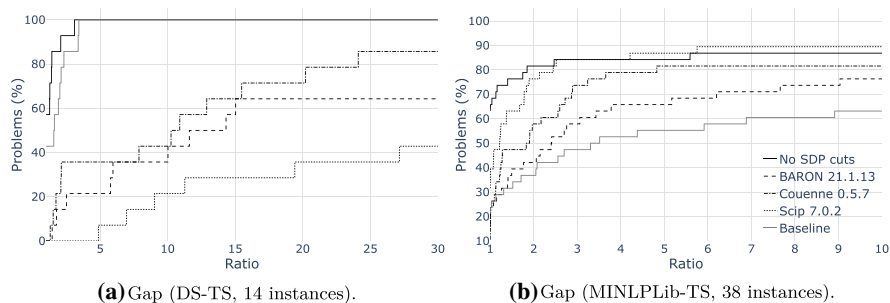
Fig. 4 Performance profiles to compare RAPOSa with other solvers in the most difficult instances (no solver solved them in 1 h)

more and more competitive as the difficulty of the instances to be solved increases (at least in the two test sets under study).

7 Conclusions and future research

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

In Sect. 6 we compared the performance of RAPOSa with three state-of-the-art solvers and the results are already very promising, since RAPOSa has proven to be competitive with all of them. Yet, given that RAPOSa is still a newborn, it has a great potential for improvement and the analyses in Sects. 4 and 5 already highlight some promising directions. We conclude with a brief outline of a few of them:

Bound tightening	Given the impact on performance of bound-tightening techniques, it may be interesting to equip RAPOSa with more sophisticated versions of both FBBT and OBBT routines, such as the ones described in [4] and [21].
Branching criterion	Similarly, given the impact of the branching criterion on the performance of the RLT technique, it is natural to study whether branching criteria adapting the integer programming approaches of pseudo-cost and/or reliability branching [1] can further improve the performance of the algorithm.
SDP cuts and general cut management	As we already mentioned in Sect. 5.7, a clear direction for future research is to get deeper into the analysis of SDP cuts. Given their promising behavior in Table 7, it would be important to understand why it has a negative impact when combined with the other enhancements. Further, implementing a general and flexible cut management scheme might

	help to improve the overall performance of RLT and might also be helpful to correct the undesirable behavior of the SDP cuts.
Learning to branch	There is an intensive research on the use of machine learning techniques to improve the performance of the branch-and-bound algorithm, especially in integer programming [3, 24, 25]. The integration of the insights from this research into RAPOSa, along with some ideas that may be specific to polynomial optimization, is definitely worth pursuing.
Integer problems	A natural avenue for RAPOSa is to extend its branch-and-bound scheme to allow for integer programming problems.

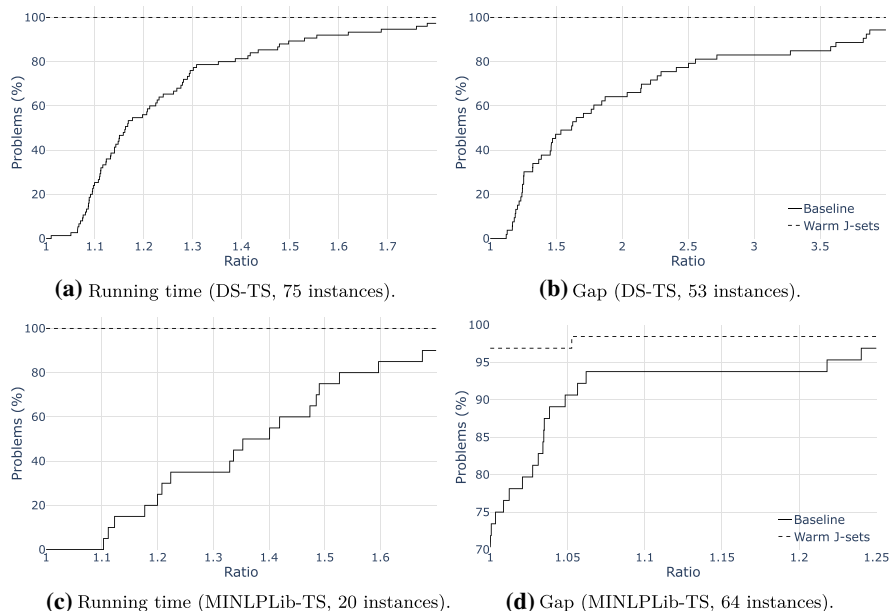# A Performance profiles for the computational experiments in Sects. 4 and 5

In this Appendix we present all performance profiles associated with the computational analyisis of the different enhancements of the basic RLT discussed in Sects. 4 and 5 (Figs. 5, 6, 7, 8, 9, 10, 11, 12).
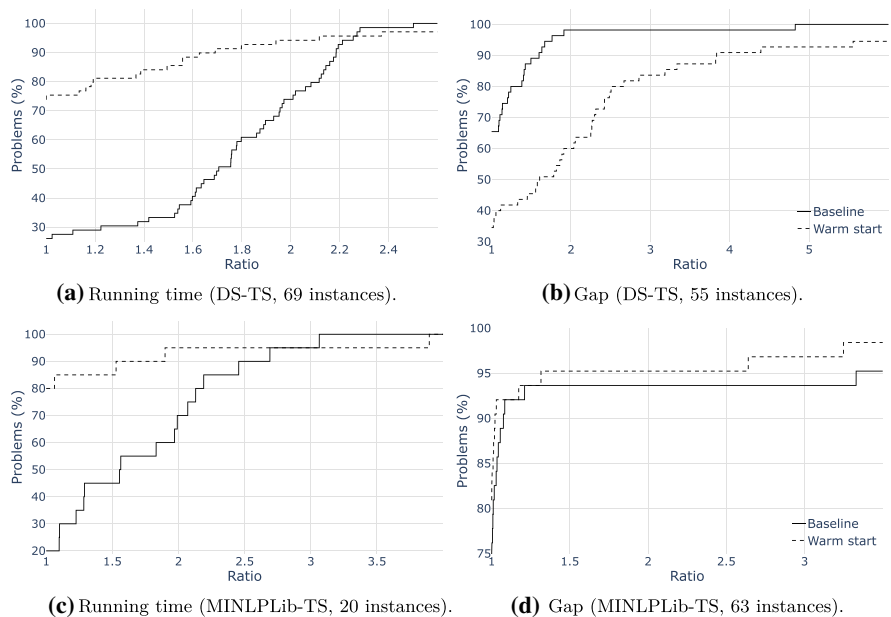
(a) Running time (DS-TS, 125 instances).

(b) Gap (DS-TS, 145 instances).

(c) Running time (MINLPLib-TS, 36 instances).

(d) Gap (MINLPLib-TS, 60 instances).
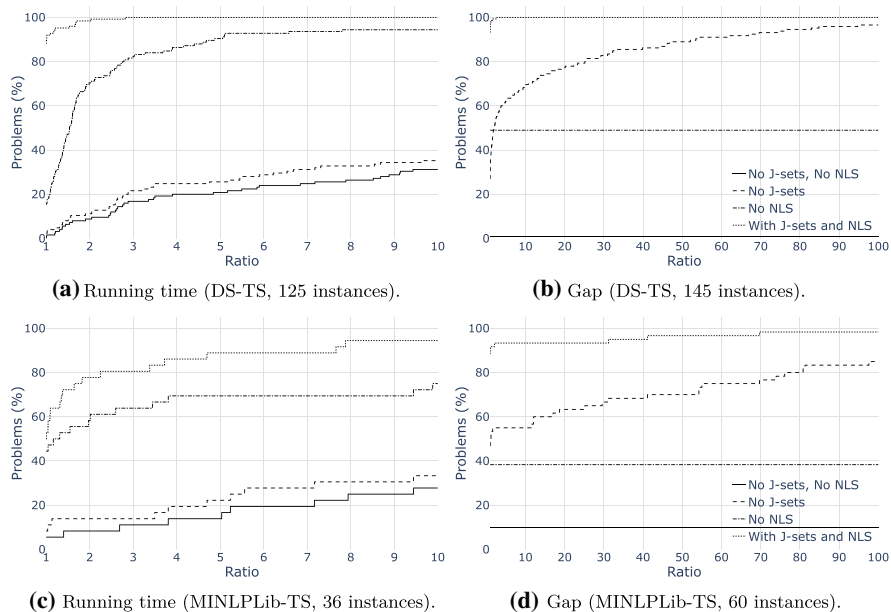
Fig. 5 Performance profiles for J-sets and the use of an auxiliary nonlinear solver

(a) Running time (DS-TS, 75 instances).

(b) Gap (DS-TS, 63 instances).

(c) Running time (MINLPLib-TS, 35 instances).

(d) Gap (MINLPLib-TS, 74 instances).
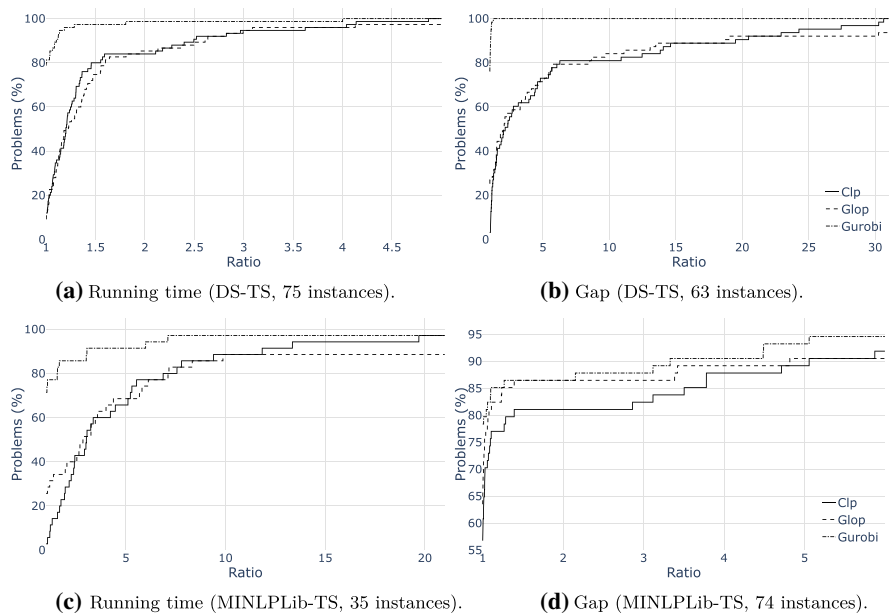
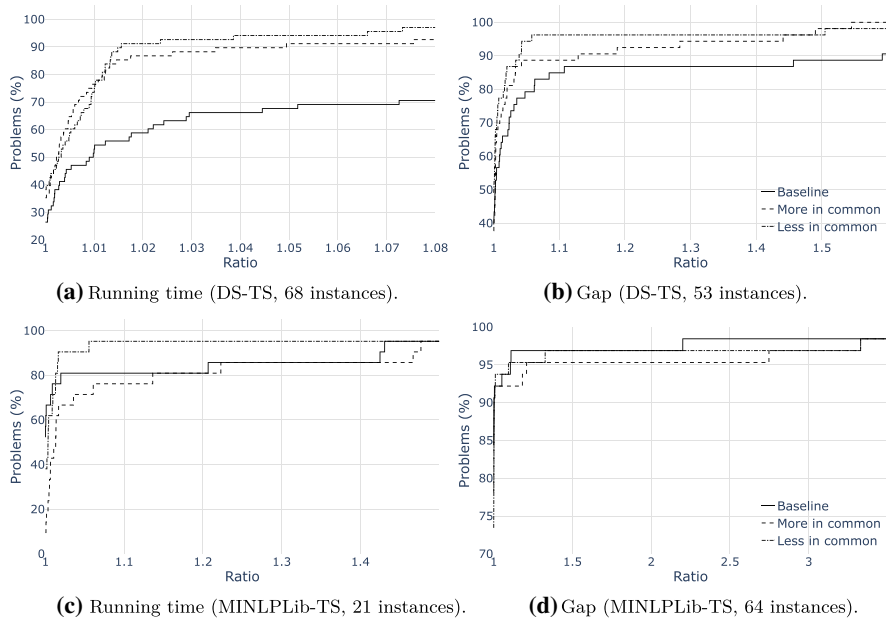**Fig. 6** Performance profiles for the different linear solvers

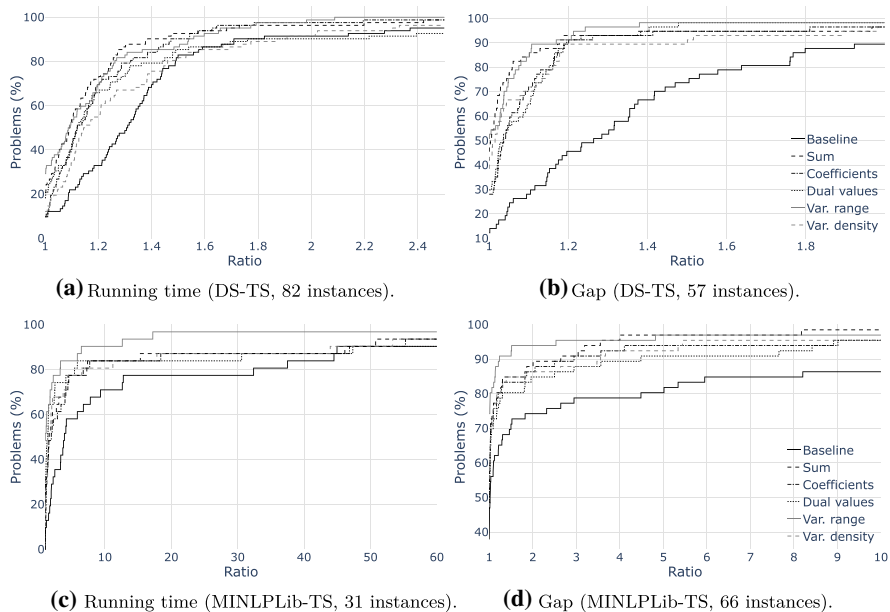**Fig. 7** Performance profiles for the warm generation of J-Sets

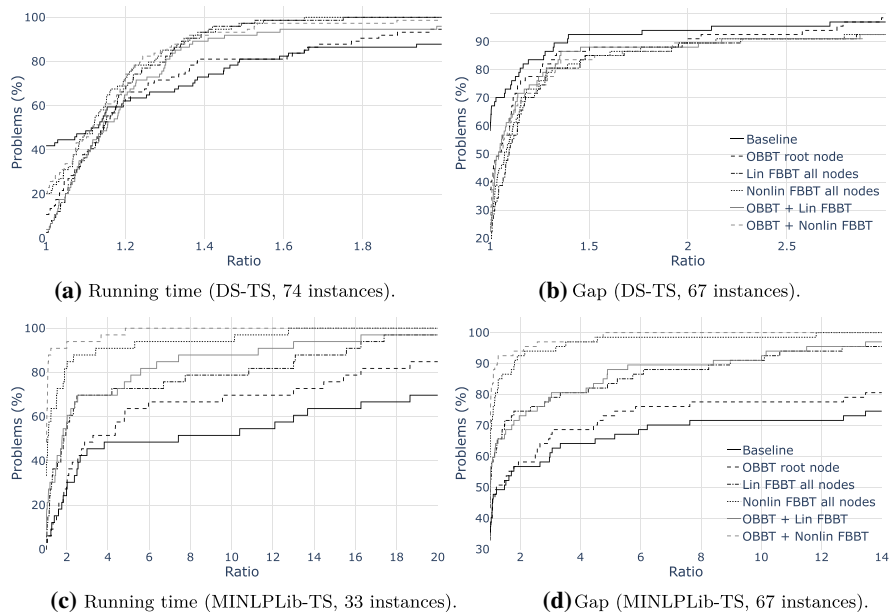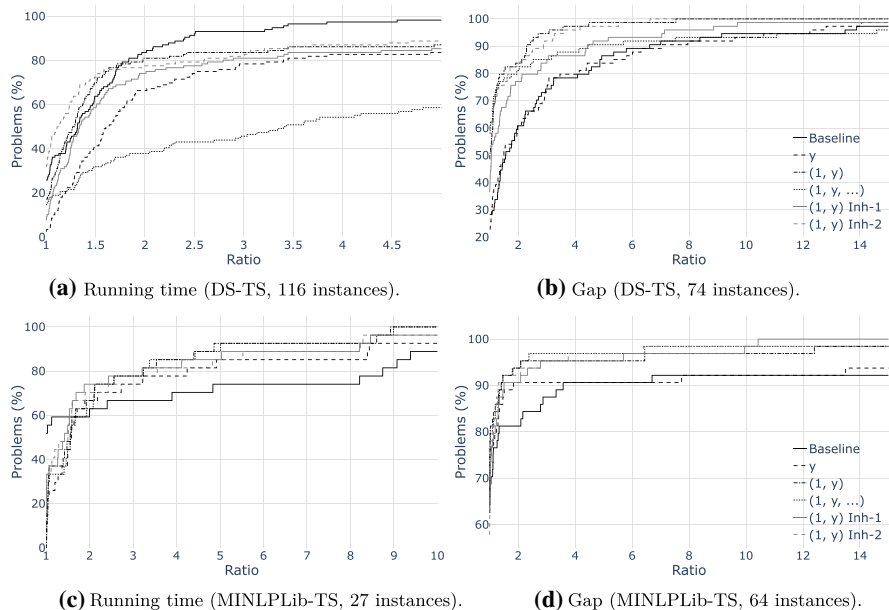**Fig. 8** Performance profiles for the warm start on the LP solver

(a) Running time (DS-TS, 68 instances).

(b) Gap (DS-TS, 53 instances).

(c) Running time (MINLPLib-TS, 21 instances).

(d) Gap (MINLPLib-TS, 64 instances).

Fig. 9 Performance profiles for the products of constraint factors and bound factors

(a) Running time (DS-TS, 82 instances).

(b) Gap (DS-TS, 57 instances).

(c) Running time (MINLPLib-TS, 31 instances).

(d) Gap (MINLPLib-TS, 66 instances).

Fig. 10 Performance profiles for the different branching criteria

(a) Running time (DS-TS, 74 instances).

(b) Gap (DS-TS, 67 instances).

(c) Running time (MINLPLib-TS, 33 instances).

(d) Gap (MINLPLib-TS, 67 instances).

Fig. 11 Performance profiles for the bound tightening techniques

(a) Running time (DS-TS, 116 instances).

(b) Gap (DS-TS, 74 instances).

(c) Running time (MINLPLib-TS, 27 instances).

(d) Gap (MINLPLib-TS, 64 instances).

Fig. 12 Performance profiles for the SDP cuts

References

1. Achterberg, T., Koch, T., Martin, A.: Branching rules revisited. Oper. Res. Lett. **33**, 42–54 (2005)
2. De Backer, B., Didier, F., Guére, E.: Glop: an open-source linear programming solver. In: 22nd International Symposium on Mathematical Programming (2015)
3. Balcan, M.-F., Dick, T., Sandholm, T., Vitercik, E.: Learning to branch. In: Dy, J., Krause, A. (Eds.) Proceedings of the 35th International Conference on Machine Learning, Stockholmsmässan, Stockholm Sweden: PMLR, Volume 80 of Proceedings of Machine Learning Research, pp. 344–353 (2018)
4. Belotti, P.: Bound reduction using pairs of linear inequalities. J. Glob. Optim. **56**, 787–819 (2013)
5. Belotti, P., Cafieri, S., Lee, J., Liberti, L.: On the convergence of feasibility based bounds tightening (2012)
6. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**, 597–634 (2009)
7. Berkelaar, M., Eikland, K., Notebaert, P., et al.: lpsolve: Open Source (Mixed-Integer) Linear Programming System, vol. 63. Eindhoven University of Technology, Eindhoven (2004)
8. Bestuzheva, K., Besançon, M., Chen, W.-K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S.J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M.E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., Witzig, J.: The SCIP Optimization Suite 8.0. Technical report, optimization online (2021)
9. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib—a collection of test models for mixed-integer nonlinear programming. INFORMS J. Comput. **15**, 114–119 (2003)
10. Bynum, M.L., Hackebeil, G.A., Hart, W.E., Laird, C.D., Nicholson, B., Siirola, J.D., Watson, J.-P., Woodruff, D.L.: Pyomo—Optimization Modeling in Python. Springer, Cham (2021)
11. Byrd, R.H., Nocedal, J., Waltz, R.A.: Knitro: an integrated package for nonlinear optimization (2006)
12. Czyzyk, J., Mesnier, M.P., More, J.J.: The NEOS server. IEEE J. Comput. Sci. Eng. **5**, 68–75 (1998)
13. Dalkiran, E., Sherali, H.D.: Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality. J. Glob. Optim. **57**, 1147–1172 (2013)
14. Dalkiran, E., Sherali, H.D.: RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems. Math. Program. Comput. **8**, 337–375 (2016)
15. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
16. Drud, A.: CONOPT: a GRG code for large sparse dynamic nonlinear optimization problems. Math. Program. **31**, 153–191 (1985)
17. Forrest, J.J., Vigerske, S., Ralphs, T., Hafer, L., Santos, H.G., Saltzman, M., Kristjansson, B., King, A.: Clp: Version 1.17.6, Zenodo (2020)
18. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: a mathematical programing language. Manag. Sci. **36**, 519–554 (1990)
19. Gay, D.M.: Hooking your solver to AMPL. Technical report 97-4-06, Bell Laboratories (1997)
20. Gay, D.M.: Writing. nl files. Technical report 2005-7907P, Sandia National Laboratories (2005)
21. Gleixner, A.M., Berthold, T., Müller, B., Weltge, S.: Three enhancements for optimization-based bound tightening. J. Glob. Optim. **67**, 731–757 (2017)
22. Gould, N., Scott, J.: A note on performance profiles for benchmarking software. ACM Trans. Math. Softw. (TOMS) **43**, 1–5 (2016)
23. Gurobi Optimization: Gurobi Optimizer Reference Manual (2022). http://www.gurobi.com
24. Khalil, E.B., Bodic, P.L., Song, L., Nemhauser, G.L., Dilkina, B.: Learning to branch in mixed integer programming. In: Schuurmans, D., Wellman, M.P. (Eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12–17, 2016, Phoenix, Arizona, USA. AAAI Press, pp. 724–731 (2016)
25. Lodi, A., Zarpellon, G.: On learning and branching: a survey. TOP 207–236 (2017)
26. Morrison, D.R., Jacobson, S.H., Sauppe, J.J., Sewell, E.C.: Branch-and-bound algorithms: a survey of recent advances in searching, branching, and pruning. Discret. Optim. **19**, 79–102 (2016)
27. Murtagh, B.A., Saunders, M.A.: Large-scale linearly constrained optimization. Math. Program. **14**, 41–72 (1978)
28. Perron, L., Furnon, V.: OR-tools (2019). https://developers.google.com/optimization/
29. Puranik, Y., Sahinidis, N.V.: Domain reduction techniques for global NLP and MINLP optimization. Constraints **22**, 338–376 (2017)
30. Sahinidis, N.V.: BARON 21.1.13: global optimization of mixed-integer nonlinear programs, user's manual (2017)

31. Sherali, H.D., Dalkiran, E., Desai, J.: Enhancing RLT-based relaxations for polynomial programming problems via a new class of v-semidefinite cuts. Comput. Optim. Appl. **52**, 483–506 (2012)
32. Sherali, H.D., Dalkiran, E., Liberti, L.: Reduced RLT representations for nonconvex polynomial programming problems. J. Glob. Optim. **52**, 447–469 (2012)
33. Sherali, H.D., Tuncbilek, C.H.: A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique. J. Glob. Optim. **103**, 225–249 (1992)
34. Waki, H., Kim, S., Kojima, M., Muramatsu, M., Sugimoto, H.: SparsePOP—a sparse semidefinite programming relaxation of polynomial optimization problems. ACM Trans. Math. Softw. **35**, 15 (2008)
35. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. **106**, 25–57 (2006)