Math. Prog. Comp. (2016) 8:337–375
DOI 10.1007/s12532-016-0099-5

FULL LENGTH PAPER

RLT-POS: Reformulation-Linearization Technique-based optimization software for solving polynomial programming problems

Evrim Dalkiran[1] · **Hanif D. Sherali**[2]

Received: 4 November 2014 / Accepted: 7 January 2015 / Published online: 3 February 2016

Abstract In this paper, we introduce a Reformulation-Linearization Technique-based open-source optimization software for solving polynomial programming problems (RLT-POS). We present algorithms and mechanisms that form the backbone of RLT-POS, including constraint filtering techniques, reduced RLT representations, and semidefinite cuts. When implemented individually, each model enhancement has been shown in previous papers to significantly improve the performance of the standard RLT procedure. However, the coordination between different model enhancement techniques becomes critical for an improved overall performance since special structures in the original formulation that work in favor of a particular technique might be lost after implementing some other model enhancement. More specifically, we discuss the coordination between (1) constraint elimination via filtering techniques and reduced RLT representations, and (2) semidefinite cuts for sparse problems. We present computational results using instances from the literature as well as randomly generated problems to demonstrate the improvement over a standard RLT implementation and to compare the performances of the software packages BARON, COUENNE, and SparsePOP with RLT-POS.

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

✉ Evrim Dalkiran
evrimd@wayne.edu

Hanif D. Sherali
hanifs@vt.edu

[1] Department of Industrial and Systems Engineering, Wayne State University, Detroit, MI 48202, USA

[2] Grado Department of Industrial and Systems Engineering, Virginia Tech, Blacksburg, VA 24061, USA

Keywords Reformulation-Linearization Technique (RLT) · Open-source code · Constraint filtering strategies · Valid inequalities · Reduced RLT representations · Polynomial programming

Mathematics Subject Classification 65K05 · 90-08 · 90C26 · 90C57

1 Introduction

The *Reformulation-Linearization Technique* (**RLT**) is a unifying approach for solving discrete and continuous nonconvex optimization problems (Sherali and Adams [19]). The RLT generates tight linear (or convex) programming relaxations by means of a lifted representation using implied polynomial constraints. In this paper, we present various algorithmic strategies and mechanisms that form the backbone of an RLT-based open-source code for solving general polynomial programming problems, including coordinated model enhancements through constraint filtering techniques, reduced basis techniques, and semidefinite cuts.

To set the stage, consider the following polynomial program PP of order $\delta \geq 2$:

$$\textbf{PP:}\quad \text{Minimize} \quad \phi_0(x) \tag{1a}$$

subject to

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

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

$$Ax = b \tag{1d}$$

$$x \in \Omega \equiv \{x : 0 \leq l_j \leq x_j \leq u_j < \infty, \forall j \in \mathcal{N}\}, \tag{1e}$$

where

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

and where T_r is an index set for the terms defining $\phi_r(.)$, with α_{rt} being real coefficients associated with the monomials $\prod_{j \in J_{rt}} x_j$, $\forall t \in T_r$, $r = 0, \ldots, R$. For any set J, let J^d denote a *multi-set of order* d, which includes all distinct combinations of indices (arranged in nondecreasing order) that belong to the d-degree Cartesian product $J \times \cdots \times J$. Accordingly, we have that $J_{rt} \in \cup_{d=1}^{\delta} \mathcal{N}^d$. Here, we have explicitly identified a linear equality system (1d), where A is $m \times n$ of rank $m < n$, and where (1c) then accommodates any other nonlinear equality constraints. For future reference, we also define $\Delta \equiv \lfloor \delta/2 \rfloor$.

For solving Problem PP, Sherali and Tuncbilek [25] proposed an RLT-based branch-and-bound algorithm and established its convergence to a global optimum when implemented using suitable branching variable selection and partitioning rules. In this procedure, at each node of the branch-and-bound tree that is identified by a bounding sub-hyperrectangle $\Omega' \equiv \{0 \leq l'_j \leq x_j \leq u'_j \leq \infty, \forall j \in \mathcal{N}\} \subset \Omega$, the RLT process

defines *bound-factors* as follows:

$$\left(x_j - l'_j\right) \geq 0 \quad \text{and} \quad \left(u'_j - x_j\right) \geq 0, \quad \forall j \in \mathcal{N}, \tag{2}$$

and accordingly generates implied *bound-factor constraints* by taking the following products of bound-factors δ at a time while allowing repetitions:

$$\prod_{j \in J_1} \left(x_j - l'_j\right) \prod_{j \in J_2} \left(u'_j - x_j\right) \geq 0, \quad \forall (J_1 \cup J_2) \in \mathcal{N}^{\delta}, \tag{3}$$

where the union of the multi-sets J_1 and J_2 preserves all the elements of J_1 and J_2 (we assume this throughout in regard to unions of multi-sets). Upon appending these valid inequalities in (3), a linearization operation is next performed by introducing a new *RLT-variable* for each distinct monomial as given by:

$$X_J = \prod_{j \in J} x_j, \quad \forall J \in \cup_{d=2}^{\delta} \mathcal{N}^d, \tag{4}$$

and substituting this throughout the augmented formulation. We use $[.]_L$ to denote the linearization operation of a polynomial expression $[.]$ by applying the substitution in (4). This process yields the following reformulation, where the corresponding RLT-based LP relaxation is obtained by dropping the identities in (5g):

$$\begin{aligned}
\text{RLT}(\Omega') : \text{Minimize} \quad & [\phi_0(x)]_L && \text{(5a)}\\
\text{subject to} \quad & \\
& [\phi_r(x)]_L \geq \beta_r, \quad \forall r = 1, \ldots, R_1 && \text{(5b)}\\
& [\phi_r(x)]_L = \beta_r, \quad \forall r = R_1 + 1, \ldots, R && \text{(5c)}\\
& Ax = b && \text{(5d)}\\
& \Big[\prod_{j \in J_1} \left(x_j - l'_j\right) \prod_{j \in J_2} \left(u'_j - x_j\right)\Big]_L \geq 0, \quad \forall (J_1 \cup J_2) \in \mathcal{N}^{\delta} && \text{(5e)}\\
& x \in \Omega' \equiv \{x : 0 \leq l'_j \leq x_j \leq u'_j < \infty, \quad \forall j \in \mathcal{N}\} && \text{(5f)}\\
& X_J = \prod_{j \in J} x_j, \quad \forall J \in \cup_{d=2}^{\delta} \mathcal{N}^d. && \text{(5g)}
\end{aligned}$$

Sherali and Tuncbilek [25] also discuss possible enhancements of such RLT-based LP relaxations. First, each equality constraint in (1c) and (1d) could be multiplied by distinct monomials such that the resulting valid equalities are of order no more than δ:

$$\Big[\phi_r(x) \prod_{j \in J} x_j\Big]_L = \beta_r \Big[\prod_{j \in J} x_j\Big]_L, \quad \forall r = R_1 + 1, \ldots, R, \forall J \in \cup_{d=1}^{\delta - \delta_r} \mathcal{N}^d \tag{6a}$$

$$\left[Ax \prod_{j \in J} x_j\right]_L = b\left[\prod_{j \in J} x_j\right]_L, \quad \forall J \in \cup_{d=1}^{\delta-1} \mathcal{N}^d, \tag{6b}$$

where δ_r is the order of the constraint $r, \forall r = 1, \ldots, R$. Second, additional valid inequalities could be produced by multiplying the original inequality constraints in (1b) with themselves as well as with bound-factors such that the resultant restriction is of order δ. Observe that bound-factor constraints (and additional RLT restrictions) of order $\delta' > \delta$ could be likewise generated to produce tighter relaxations, albeit at a higher computational cost, while extending the set of RLT variables to X_J for all $J \in \mathcal{N}^d, d = 2, \ldots, \delta'$.

The main contributions of this paper are as follows. First, we introduce an RLT-based open-source optimization software for solving polynomial programming problems that extends and combines various formerly introduced algorithmic enhancements of the RLT approach, including the constraint filtering technique of [8], the reduced basis techniques of [23], and the semidefinite cuts of [22] and [24]. Second, we propose a new routine for implementing reduced basis techniques for sparse problems. Third, we examine branching versus cut generation decisions and propose a new cut inheritance scheme. Finally, we perform extensive computational experiments to demonstrate the effectiveness of RLT-POS in comparison with the SDP-based optimization software SparsePOP, the open-source optimization software COUENNE, as well as the commercial global optimization software BARON, utilizing a test-bed of problems from the literature and randomly generated problems.

The remainder of this paper is organized as follows. Section 2 begins with a description of generating reduced size RLT representations via constraint filtering techniques and basis partitioning schemes, and discusses the coordination between them. Section 3 covers model enhancements through the generation of semidefinite cuts and examines branching versus cut generation decisions. Section 4 describes the generation of the polynomial programming test problems along with details of the branch-and-bound algorithm including branching variable selection and partitioning strategies, as well as demonstrates the effectiveness of the proposed algorithmic strategies via an extensive computational analysis. Finally, Section 5 concludes the paper with a summary and recommendations for future research.

2 Reduced size RLT formulations

The RLT bound-factor constraints (3) effectively enforce the RLT-defining identities (4) in the limit within a branch-and-bound algorithm. However, using special problem structures, we could eliminate some of these constraints such that the resulting formulation remains sufficiently tight and ensures convergence, while benefiting from a reduction in the solution effort. In this section, we discuss reduced size RLT formulations for two types of problem structures: (a) sparse problems and (b) problems having linear equality subsystems.

First, we discuss the constraint filtering techniques introduced in Dalkiran and Sherali [8], denoted as the $\boldsymbol{N_J^{d_J}}$**-set** and the $\boldsymbol{J}$**-set** of bound-factor constraints, both of which guarantee theoretical convergence to a global optimum, while suitably gen-

erating only a subset of the standard bound-factor constraints. Next, we describe a reduced basis technique and an alternative nonbasic variable space representation as proposed by Sherali et al. [23], which are predicated on an embedded linear equality subsystem (1d) and serve the purpose of necessitating only a strict subset of bound-factor constraints. After this brief introduction of constraint filtering techniques and reduced RLT formulations, we address the coordination between these two model enhancement procedures.

We mention here that in an earlier study, Sherali and Tuncbilek [26] had proposed a two-stage constraint filtering technique for selecting a subset of bound-factor constraints that help impose useful lower or upper bounding expressions for existing monomials. In contrast to the J-set and $N_J^{d_J}$-set of [8], this heuristic technique requires eventually including all bound-factor constraints in (3) to ensure theoretical convergence of the underlying branch-and-bound algorithm. In a recent study, Zorn and Sahinidis [32] similarly proposed selecting a subset of RLT bound-factor constraints that involve only the existing monomials in order to enhance the LP relaxation generated by the standard factorable reformulation used in BARON [18,29]. This enhanced formulation significantly reduced the computational effort in comparison to the standard factorable formulation for solving nonlinear programming problems. Waki et al. [30] also explored sparsity in polynomial programming problems to construct reduced-sized SDP relaxations, which were later implemented in SparsePOP [31]. Lasserre [13] proved that the SDP relaxations proposed in [30] guarantee convergence to a global optimum provided that a certain running intersection property is met.

Liberti [15] explored an embedded linear equality system that eliminates a subset of defining identities in (4) while constructing reduced size LP relaxations for quadratic and bilinear problems. A graph theoretical approach was introduced in [16] for effectively implementing such reduced basis techniques for sparse bilinear programs, and Cafieri et al. [7] extended these ideas to polynomial programming problems.

2.1 The $N_J^{d_J}$-set of constraint filtering technique

The RLT constraints in (3) involve all possible products of bound-factors, regardless of the existing monomials in the original problem. Such a non-specialized treatment results in relaxations having the same set of bound-factor constraints for any polynomial program of the same degree and having the same number of variables. Thus, neither special structures nor sparsity is exploited to generate favorable relaxations. In lieu of such a generic strategy, Dalkiran and Sherali [8] suggested approximating each monomial separately by deriving only those bound-factor constraints that involve simply that monomial itself and proved that the convergence result in [25] continues to hold true in this case.

Toward this end, let N_J and d_J respectively denote the largest nonrepetitive (index-wise) subset of J and the cardinality of J for a given index multi-set J, and let $N_J^{d_J}$ represent a multi-set of order d_J that includes all distinct combinations of indices belonging to the d_J-degree Cartesian product $N_J \times \cdots \times N_J$.

Proposition 1 ([8]) *For each J such that the monomial* $\prod_{j\in J} x_j$ *appears in Problem PP, the convergence result (Theorem 1) in* [25] *holds true if the following RLT bound-factor constraints are present in the linear programming relaxation:*

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \quad \forall\,(J_1 \cup J_2) \in N_J^{d_J}. \tag{7}$$

Dalkiran and Sherali [8] further proposed a systematic routine for generating the $N_J^{d_J}$-set of restrictions while avoiding duplicating constraints and eliminating implied constraints. Note that, as illustrated in [8], the standard RLT constraints that are eliminated by the $N_J^{d_J}$-set routine may further tighten the underlying LP relaxation. However, the main hypothesis is that the branch-and-bound algorithm is better off eliminating these constraints in order to reduce the solution times for solving the LP relaxations. Indeed, the computational analysis at the root node demonstrated that the $N_J^{d_J}$-set routine eliminated a significant subset of bound-factor constraints that resulted in improved LP solution times while yet maintaining sufficiently tight relaxations that provided satisfactory lower bounds. For the resulting RLT-based branch-and-bound algorithm, the $N_J^{d_J}$-set reduced the average computational effort by 13-, 8-, and 9-fold over the standard RLT for solving degree two, four, and six problems, respectively. A closer look at the computational results reveals that the improvements were more significant for sparse problems (density = 0.001) in comparison to relatively dense problems (density = 0.1), where the *density* is defined as the ratio of the number of existing nonlinear monomials to the number of all possible nonlinear monomials.

2.2 The J-set of constraint filtering technique

The $N_J^{d_J}$-set of constraints generates relaxations that are almost as tight as the ones derived using the standard RLT. However, the filtering effect of the $N_J^{d_J}$-set routine quickly diminishes as the density of the problem increases beyond 0.1. Dalkiran and Sherali [8] proposed a more robust theoretical filtering routine, called the *J-set routine*, which maintains its filtering power for more dense problems as well, and assures convergence to a global optimum. The following proposition identifies this set of constraints:

Proposition 2 ([8]) *The convergence result (Theorem 1) in* [25] *holds true if the following RLT bound-factor constraints are appended to the relaxation for each J such that the monomial* $\prod_{j\in J} x_j$ *appears in Problem PP:*

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \quad \forall\,(J_1 \cup J_2) = J. \tag{8}$$

Accordingly, a J-set constraint generation routine was developed as follows, which eliminates constraint replications as well as implied constraints:

J-set Routine:

Initialization: Define a list $\mathscr{L} \equiv \{J : X_J \text{ appears within (5a)–(5c)}\}$.

Step 1: Let $f(J) = 1$ for all maximal (w.r.t inclusion) elements of the set $\mathscr{L}$.

Step 2: For each $J \in \mathscr{L}$ such that $f(J) = 1$, generate the corresponding bound-factor constraints in (8).

The $N_J^{d_J}$-set routine generates bound-factor constraints by composing products of some d_J bound-factors of variables existing in the index set J, allowing repetitions of bound-factors, which results in $\binom{2|N_J|+d_J-1}{d_J}$ number of constraints, where d_J is the cardinality of J. The J-set routine reserves the same number of bound-factor spots for each variable as the degree of that variable in the considered monomial $\prod_{j\in N_J} x_j^{r_j}$, where r_j represents the number of times the variable x_j appears within the monomial. Hence, the J-set of constraints is a subset of the $N_J^{d_J}$-set, where the number of such constraints is given by $\prod_{j\in N_J}(r_j + 1)$.

Dalkiran and Sherali [8] used an illustrative example to demonstrate that the $N_J^{d_J}$-set routine can generate LP relaxations that are strictly tighter than those obtained using the J-set routine, and provided insights into conditions under which this can occur. The given computational results demonstrated that the J-set routine further reduced the average effort of the $N_J^{d_J}$-set routine by factors of two, four, and two, respectively, for solving degree two, four, and six problems using an RLT-based branch-and-bound algorithm.

2.3 Reduced basis techniques

Sherali et al. [23] introduced reduced size RLT-based formulations for polynomial programming problems having an embedded linear equality system. Given a basis B of A in (1d), the authors partition x into basic and nonbasic variables, x_B and x_N, respectively, with associated respective index sets J_B and J_N. They then showed that the constraint-factor restrictions in (6b) along with the RLT defining identities for nonbasic variables imply the remaining RLT identities in (4) as follows:

Proposition 3 ([23]) *Let the equality system $Ax = b$ in (1d) be partitioned as $Bx_B + Nx_N = b$ for any basis B of A, and define*

$$Z = \left\{(x, X) : \text{(1d)}, \text{(6b)}, \text{ and } X_J = \prod_{j\in J} x_j, \ \forall J \in J_N^d, \text{ for } d = 2, \ldots, \delta\right\}. \quad (9)$$

Then, we have (4) holding true for any (x, X) in Z.

According to Proposition 3, the authors eliminated the RLT defining identities involving at least one basic variable in (4) and enforced those that involve only nonbasic variables as in (10h), below. Subsequently, they generated the RLT bound-factor constraints with $(J_1 \cup J_2) \subseteq J_N^\delta$ among the ones in (3). In order to strengthen the underlying LP relaxation upon deleting (10h), the authors further included the implied bounds on RLT variables that involve at least one basic variable as in (10g) below, to

obtain the following reformulation of Problem PP by applying the substitution in (4):

$$\textbf{RRLT:}\quad \text{Min.}\quad [\phi_0(x)]_L \tag{10a}$$

s.t.

$$[\phi_r(x)]_L \geq \beta_r, \quad \forall r = 1, \ldots, R_1 \tag{10b}$$

$$[\phi_r(x)]_L = \beta_r, \quad \forall r = R_1 + 1, \ldots, R \tag{10c}$$

$$Ax = b \tag{10d}$$

$$BX_{(BJ)} + NX_{(NJ)} = bX_J, \quad \forall J \subseteq \mathscr{N}^d, d = 1, \ldots, \delta - 1 \tag{10e}$$

$$\Big[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0, \quad \forall (J_1 \cup J_2) \subseteq J_N^\delta \tag{10f}$$

$$l \leq x \leq u, \text{ and}$$

$$\prod_{j\in J} l_j \leq X_J \leq \prod_{j\in J} u_j, \quad \forall J \subseteq \mathscr{N}^d,\ d = 2, \ldots, \delta,\ |J_B \overline{\cap} J| \geq 1 \tag{10g}$$

$$X_J = \prod_{j\in J} x_j, \quad \forall J \subseteq J_N^d, d = 2, \ldots, \delta, \tag{10h}$$

where (BJ) and (NJ) in (10e) are multi-set vectors that combine each variable index $B_j \in J_B$ and $N_j \in J_N$, respectively, with the indices in J in nondecreasing order, and where $J_B \,\overline{\cap}\, J$ in (10g) is the (multi-) subset of J that contains only the basic variable indices. The reduced RLT-based formulations can be further tightened by including those bound-factor constraints that have associated positive dual variables at an optimal solution to the standard RLT relaxation at the root node. A novel constraint inheritance scheme was proposed in [23] that provides monotonicity in the lower bounds while controlling the number of such inherited restrictions.

As an alternative to the reduced size RLT formulations, Sherali et al. [23] also investigated standard RLT relaxations that are constructed in a reduced subspace $\mathbb{R}^{n-m}$, where the basic variables x_B are eliminated via the substitution $x_B = B^{-1}(b - Nx_N)$. Note that this relaxation is not as tight as the foregoing enhanced reduced RLT formulation by virtue of (10e), but its size is relatively smaller, thereby enabling a significant reduction in computational effort for solving the underlying LP relaxations. The authors therefore suggested a hybrid algorithm that performs a root node analysis to judiciously select either the reduced RLT formulation in $\mathbb{R}^n$ or the standard RLT relaxation in the nonbasic variable space $\mathbb{R}^{n-m}$, depending on the optimality gaps at the root node and the sizes of these relaxations.

2.4 Coordination between constraint filtering and reduced basis techniques

In this section, we explore efficient methods for implementing the reduced basis techniques for sparse problems. We first discuss the necessity for modifying the reduced basis techniques when combining them with constraint filtering strategies. We then propose using a subset of the constraints in (10e), which, along with the RLT defining identities for nonbasic variables, is still sufficient to imply the remaining RLT defining identities. Next, we analyze such alternative subsets and recommend the smallest one

for implementation. We close the section with an illustrative example and a discussion of conditions under which such reduced basis techniques are advantageous.

As discussed in Sects. 2.1–2.3, the constraint filtering techniques and the reduced basis techniques eliminate a subset of bound-factor constraints while maintaining tight relaxations. Constraint filtering techniques exploit sparsity for constraint elimination, whereas the reduced basis techniques achieve this by taking advantage of inherent linear equality constraints. Whereas constraint filtering techniques do not alter the problem structure, the reduced basis techniques require generating a set of additional constraint-factor RLT restrictions (10e). When the original problem is sparse, i.e., many of the possible nonlinear monomials do not exist in the problem, the latter constraint generation process may introduce numerous new RLT variables that are consequently required to be approximated by the J-set and the $N_J^{d_J}$-set of constraints. Hence, the number of RLT constraints eliminated by the reduced basis techniques may be subdued by the number of constraints subsequently introduced to approximate the new RLT variables. In order to avoid such an adverse effect, we propose to generate only a judicious subset of constraint-factor RLT restrictions in (10e). Given any basis B of A of the equality system, we denote the representation of basic variables in terms of nonbasic variables as $x_{B_j} = (B^{-1}b)_{B_j} - (B^{-1}N)_{B_j}x_N, \forall B_j \in J_B$, where $(B^{-1}b)_{B_j}$ and $(B^{-1}N)_{B_j}$ respectively represent the B_j^{th} entry of the vector $(B^{-1}b)$ and the B_j^{th} row of the matrix $(B^{-1}N)$. The following routine identifies the proposed reduced set of constraint-factor RLT restrictions, which is shown in Proposition 4 below to suffice for ensuring a legitimate model representation:

Reduced RLT Routine for Sparse Problems:

Initialization: Define $\mathscr{K} \equiv \{J : X_J$ appears in (5a)–(5c) and $|J \cap J_B| \geq 1\}$, $\mathscr{K}' = \emptyset$, and $\mathscr{L} \equiv \{J : X_J$ appears within (5a)–(5c) and $J \cap J_B = \emptyset\}$.

Step 1: If $\mathscr{K} = \emptyset$, go to Step 4. Else, select an index set $J \in \mathscr{K}$ that involves the maximum number of basic variables, delete it from the list $\mathscr{K}$ and add it to the list $\mathscr{K}'$.

Step 2: Let $B_j \in J$ be a randomly selected basic variable index. Multiply the representation of the basic variable x_{B_j} in terms of the nonbasic variables by $\prod_{J\setminus\{B_j\}} x_j$, and linearize and append this to the relaxation.

Step 3: If $J\setminus\{B_j\}$ involves any basic variable, the multiplication at Step 2 generates monomials (on the right-hand side of the considered basic variable identity) involving basic variables. Include these monomials within the list $\mathscr{K}$. Otherwise, if $J\setminus\{B_j\}$ does not involve any basic variable, then include the resulting monomials on the right-hand side of the considered basic variable identity within the set $\mathscr{L}$. Continue with Step 1.

Step 4: Apply the J-set Routine to the set $\mathscr{L}$ in order to generate the proposed set of filtered bound-factor restrictions for reformulating the model.

Remark 1 Step 2 of the Reduced RLT Routine for Sparse Problems randomly selects a basic variable among the ones in the index set J, which in turn affects the number of monomials involving a basic variable that are generated at Step 3. A judicious approach for choosing a basic variable at this step is to identify one that would generate the minimum number of such monomials at Step 3. Hence, we suggest selecting a basic

variable that has the minimum number of nonzero terms in its associated representation in terms of nonbasic variables. Observe that the number of monomials that do not involve a basic variable would be the same for each alternative implementation.

Note that the monomials that are included at Step 3 within the set $\mathscr{K}$ have one less number of basic variables than present in the index set selected at Step 1. Hence, given the selection rule at Step 1, we have that Step 2 processes each monomial having a basic variable at most once. Since the number of monomials involving at least one basic variable index is finite, the routine terminates after a finite number of iterations. The following result establishes that the resulting reduced set of constraint-factor RLT restrictions, denoted by $(10e)_R$, serve the required purpose of ensuring a legitimate model representation:

Proposition 4 *Let the equality system $Ax = b$ in (1d) be partitioned as $Bx_B + Nx_N = b$ for any basis B of A. Suppose that the Reduced RLT Routine for Sparse Problems is run and the constraint-factor RLT restrictions, $(10e)_R$, are generated for a given Problem PP. Define*

$$Z = \left\{(x, X) : (1d), (10e)_R, \text{ and } X_J = \prod_{j \in J} x_j, \ \forall J \in \mathscr{L}\right\}. \tag{11}$$

Then, we have $X_J = \prod_{j \in J} x_j, \ \forall J \in \mathscr{K}'$ holding true for any (x, X) in Z.

Proof Following the proof of Proposition 3 in [23], we prove this result by induction on the number of basic variables within an existing monomial ranging between $[1, \delta]$. For each $X_{(B_j J)}$, $B_j \cup J \in \mathscr{K}'$ with one basic variable, the routine generates a constraint-factor RLT restriction using the representation of the basic variable in terms of the nonbasic variables according to: $X_{(B_j J)} = (B^{-1}b)_{B_j} X_J - (B^{-1}N)_{B_j} X_{(NJ)}$. Note that J and $N_j \cup J, \forall N_j \in J_N$, do not involve any basic variable indices and appear within $\mathscr{L}$ by Step 3 of the routine. Thus we have $X_J = \prod_{j \in J} x_j$ and $X_{(N_j J)} = x_{N_j} \prod_{j \in J} x_j, \forall N_j \in J_N$, by (11). Substituting X_J and $X_{(NJ)}$ within the foregoing constraint-factor RLT restriction, we get

$$\begin{aligned} X_{(B_j J)} &= (B^{-1}b)_{B_j} \prod\nolimits_{j \in J} x_j - (B^{-1}N)_{B_j} x_N \prod\nolimits_{j \in J} x_j \\ &= \left((B^{-1}b)_{B_j} - (B^{-1}N)_{B_j} x_N\right) \prod\nolimits_{j \in J} x_j \\ &= x_{B_j} \prod\nolimits_{j \in J} x_j, \end{aligned}$$

and so the result holds true in this case. Hence, assume that the result is true for $\{1, \ldots, d\}$, where $1 \le d \le \delta - 1$, i.e.,

$$X_J = \prod_{j \in J} x_j, \quad \forall J \in \mathscr{K}' : |J_B \overline{\cap} J| \le d, \tag{12}$$

and examine the case of $d + 1$. Consider the monomials having $d + 1$ basic variables. Let $X_{(B_j J)}$ be one such monomial where the basic variable index B_j was selected

among some $d+1$ basic variable indices at Step 2 of the routine. At Step 3, the routine then generated the following constraint-factor RLT restriction using the representation of x_{B_j} in terms of nonbasic variables:

$$X_{(B_j J)} = (B^{-1}b)_{B_j} X_J - (B^{-1}N)_{B_j} X_{(NJ)}. \tag{13}$$

Note that J and $N_j \cup J, \forall N_j \in J_N$, have d basic variable indices and appear within $\mathscr{K}'$. Hence, we have $X_J = \prod_{j\in J} x_j$ and $X_{(N_j J)} = x_{N_j} \prod_{j\in J} x_j, \forall N_j \in J_N$, by (12). Accordingly, the constraint-factor restriction in (13) implies that

$$\begin{aligned} X_{(B_j J)} &= (B^{-1}b)_{B_j} \prod\nolimits_{j\in J} x_j - (B^{-1}N)_{B_j} x_N \prod\nolimits_{j\in J} x_j \\ &= \Big((B^{-1}b)_{B_j} - (B^{-1}N)_{B_j} x_N\Big) \prod\nolimits_{j\in J} x_j \\ &= x_{B_j} \prod\nolimits_{j\in J} x_j, \end{aligned}$$

and so we have $X_J = \prod_{j\in J} x_j, \forall J \in \mathscr{K}' : |J_B \overline{\cap} J| = d+1$, which therefore establishes the result. □

Utilizing Proposition 4, we impose $(10e)_R$ instead of (10e) and we further reduce the number of RLT defining identities in (10h) by imposing only those for $J \in \mathscr{L}$ as in (14h). Subsequently, we append the bound-factor constraints in (14f) that are generated by applying the J-set routine to the set $\mathscr{L}$. As in Problem PP1, we explicitly include the implied bounds for the X_J-variables in (14g) for enhancing the underlying RLT-relaxation obtained upon dropping (14h). The reformulated polynomial program is then given as follows:

J-RRLT Minimize $[\phi_0(x)]_L$ (14a)

subject to

$$[\phi_r(x)]_L \geq \beta_r, \quad \forall r = 1, \ldots, R_1 \tag{14b}$$

$$[\phi_r(x)]_L = \beta_r, \quad \forall r = R_1+1, \ldots, R \tag{14c}$$

$$Ax = b \tag{14d}$$

$$\text{Constraint } (10e)_R \tag{14e}$$

$$\Big[\prod_{j\in J_1}(x_j - l_j) \prod_{j\in J_2}(u_j - x_j)\Big]_L \geq 0,$$

$$\forall (J_1 \cup J_2) = J : J \in \mathscr{L} \text{ and } f(J) = 1 \tag{14f}$$

$$l \leq x \leq u, \text{ and } \prod_{j\in J} l_j \leq X_J \leq \prod_{j\in J} u_j, \quad \forall J \in \mathscr{K}' \tag{14g}$$

$$X_J = \prod_{j\in J} x_j, \quad \forall J \in \mathscr{L}. \tag{14h}$$

Remark 2 To strengthen the resulting LP relaxation, we judiciously select additional bound-factor constraints to explicitly approximate the RLT defining identities for $J \in$

$\mathscr{K}'$. At the root node of the branch-and-bound process, we include the bound-factor restrictions generated by applying the J-set routine to $\mathscr{K}'$ as well (instead of just (14f)), and identify those constraints that yield positive associated dual variables as given by

$$K_{+duals} \equiv \left\{(J_1, J_2) : (J_1 \cup J_2) \in \mathscr{K}',\, f(J_1 \cup J_2) = 1, \quad \text{and } w_{(J_1,J_2)} > 0\right\},$$

where $w_{(J_1,J_2)}$ is the value of the associated dual variable with the corresponding constraint at the solution of the strengthened root node problem. At the subsequent node subproblems, we then also generate bound-factor constraints for $(J_1, J_2) \in K_{+duals}$ and append these to Problem J-RRLT to generate J-RRLT+.

The basis selection has paramount importance on the performance of the reduced basis techniques. Sherali et al. [23] proposed constructing a basis B of A where the variables that are less likely to take on values at their bounds are selected as nonbasic variables, so that the bound-factor product relationships are explicitly imposed on such variables. Given an ordered list of variables as described below, the authors suggested selecting B by identifying m linearly independent columns of A that are associated with the variables in the order they appear in the list. For constructing such an ordered list of variables, four methods were examined, where the one proposed for implementation initially solves the full RLT relaxation at the root node and accordingly sorts the bound-factor constraints in nonincreasing order of their associated dual variable values. Denoting Q as the number of bound-factor constraints in (10f) within the original RLT relaxation, the method then counts the number of times each variable appears within the first Q of the foregoing sorted bound-factor constraints, and accordingly sorts the variables in nondecreasing order of this measure.

The number of bound-factor constraints in (14f) within the reduced RLT relaxation (14) for sparse problems depends on the selected basis, which prevents us using the above-mentioned method suggested in [23]. Based on an alternative method proposed in [23], we calculate $\sum_{\substack{J \in \mathscr{L} \\ f(J)=1}} k_{jJ}\lambda_J$, where k_{jJ} is the number of times the variable index j appears in J and λ_J is the maximum of the dual multipliers associated with the J-set of bound-factor constraints that involves the variable $X_J, \forall J \in \mathscr{L}$ and $f(J) = 1$. The variables are then sorted in nondecreasing order of this measure and a basis B of A is selected based on this list.

We next illustrate how alternative reduced basis formulations can be generated, and compare them with the original formulation. Consider the following polynomial programming problem having a linear equality system (15b, 15c) of rank two:

$$\begin{aligned}
\textbf{PP:}\ \text{Minimize}\quad & x_1x_2x_3x_5^2 && (15a)\\
\text{subject to}\quad & x_1 + 0.5x_3 + x_4 = 3 && (15b)\\
& x_2 + x_5 = 6 && (15c)\\
& x_3x_5 - x_4^2 \geq 1.5 && (15d)\\
& l_i \leq x_i \leq u_i, \quad i = 1, 2, 3, 4, 5. && (15e)
\end{aligned}$$

Let x_1 and x_2 be selected as basic variables. Implementing the Reduced RLT Routine, we define the list $\mathscr{K} = \{\{1, 2, 3, 5, 5\}\}$ at the initialization step. At Step 2 of the routine, we may either select {1} or {2}. Assuming that x_2 is selected, the equality constraint (15c) is multiplied by $x_1x_3x_5^2$, which generates two additional monomials $x_1x_3x_5^3$ and $x_1x_3x_5^2$. Since {1, 3, 5, 5} involves a basic variable index, these two monomials are added to $\mathscr{K}$. At the next iteration, we have $\mathscr{K} = \{\{1, 3, 5, 5, 5\}, \{1, 3, 5, 5\}\}$. The Steps 1–3 are repeated for each of the two monomials in $\mathscr{K}$, where the equality constraint (15b) is multiplied respectively by $x_3x_5^3$ and $x_3x_5^2$. At the end of this routine, we generate the J-set of constraints for each monomial that does not involve basic variables (and has $f(J) = 1$ as per the J-set Routine) to obtain a reduced RLT formulation as follows:

J-RRLT-1: Minimize X_{12355} (16a)

subject to
$$x_1 + 0.5x_3 + x_4 = 3 \tag{16b}$$
$$X_{13555} + 0.5X_{33555} + X_{34555} - 3X_{3555} = 0 \tag{16c}$$
$$X_{1355} + 0.5X_{3355} + X_{3455} - 3X_{355} = 0 \tag{16d}$$
$$x_2 + x_5 = 6 \tag{16e}$$
$$X_{12355} + X_{13555} - 6X_{1355} = 0 \tag{16f}$$
$$X_{35} - X_{44} \geq 1.5 \tag{16g}$$
$$\left[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\right]_L \geq 0, \quad \forall (J_1 \cup J_2) \in \mathscr{L} \tag{16h}$$
$$l_i \leq x_i \leq u_i, \quad i = 1, 2, 3, 4, 5, \tag{16i}$$

where $\mathscr{L} = \{\{4, 4\}, \{3, 3, 5, 5, 5\}, \{3, 4, 5, 5, 5\}\}$. The alternative formulation (17) given below is generated in case x_1 is selected at Step 2 of the first iteration. Note that x_1 and x_2 involve two terms and one term, respectively, in their nonbasic variable space representation and require 1+3 and 1+2 constraint-factor RLT restrictions as shown in (17) and (16), respectively.

J-RRLT-2: Minimize X_{12355} (17a)

subject to
$$x_1 + 0.5x_3 + x_4 = 3 \tag{17b}$$
$$X_{12355} + 0.5X_{23355} + X_{23455} - 3X_{2355} = 0 \tag{17c}$$
$$x_2 + x_5 = 6 \tag{17d}$$
$$X_{23355} + X_{33555} - 6X_{3355} = 0 \tag{17e}$$
$$X_{23455} + X_{34555} - 6X_{3455} = 0 \tag{17f}$$
$$X_{2355} + X_{3555} - 6X_{355} = 0 \tag{17g}$$
$$X_{35} - X_{44} \geq 1.5 \tag{17h}$$

$$\left[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\right]_L \geq 0, \quad \forall (J_1 \cup J_2) \in \mathscr{L} \tag{17i}$$

$$l_i \leq x_i \leq u_i, \quad i = 1, 2, 3, 4, 5. \tag{17j}$$

Similarly, another valid polynomial programming formulation can be obtained by eliminating the basic variables from the problem upon using the substitution $x_B = B^{-1}(b - Nx_N)$. This new formulation in the reduced subspace $\mathbb{R}^{n-m}$ may introduce new monomials as observed in the generation of the constraint-factor RLT restrictions in the reduced RLT formulations above. In the foregoing example, we can eliminate the basic variables x_1 and x_2 using the substitutions $x_1 = 3 - 0.5x_3 - x_4$ and $x_2 = 6 - x_5$, which generates a valid formulation in the (x_3, x_4, x_5)-space. The J-set relaxation of this reformulation is given as follows:

$J-$**set in** $\mathbb{R}^{n-m}$: Minimize $18X_{355} - 3X_{3355} - 6X_{3455} - 3X_{3555}$

$$+ 0.5X_{33555} + X_{34555} \tag{18a}$$

subject to $X_{35} - X_{44} \geq 1.5$ (18b)

$$l_1 \leq 3 - 0.5x_3 - x_4 \leq u_1 \tag{18c}$$

$$l_2 \leq 6 - x_5 \leq u_2 \tag{18d}$$

$$\left[\prod_{j\in J_1}(x_j - l_j)\prod_{j\in J_2}(u_j - x_j)\right]_L \geq 0, \quad \forall (J_1 \cup J_2) \in \mathscr{L} \tag{18e}$$

$$l_i \leq x_i \leq u_i, \quad i = 3, 4, 5. \tag{18f}$$

The optimal objective value of Problem PP in (15) is 15.4, whereas the RLT relaxation obtained upon dropping the defining identities in (5g) within (5) provides a lower bound of 5.3. The lower bound improves to 15.0 by appending the constraint-factor restrictions (6b), which reduces the optimality gap from 65.2 to 2.6 %, thus demonstrating the potential strength of these constraints.

Next, we compare the relaxations generated by the J-set constraint filtering technique and the reduced basis technique. The J-set relaxation obtained by (8) produces a lower bound of 2.8 with an optimality gap of 81.6 % by generating only 27 bound-factor constraints among 2002. The reduced RLT formulation based on (10) has a lower bound of 2.0 with an optimality gap of 87 % while generating 252 of 2002 bound-factor constraints along with 250 constraint-factor restrictions. When the bound-factor constraints that have associated positive duals at the root node are appended to the latter RLT formulation as per Remark 2 (there are 55 such constraints), the lower bound increases to 15.0 and the optimality gap reduces from 87 to 2.6 %.

The three reduced RLT relaxations for sparse problems as given in (16), (17), and (18) provide a lower bound of -182.0 while generating only 31 bound-factor constraints. When enhanced by the bound-factor constraints having associated positive duals as per Remark 2 (20 and 30 such constraints, respectively), lower bounds based

on (16) and (17), respectively improve to 6.9 and 6.1, reducing the optimality gaps from 1279.5 to 52.3 % and 60.4 %.

The J-set, the J-RRLT+, and the J-NB formulations are the three alternatives discussed above for constructing an RLT-based LP relaxation for polynomial programming problems having a linear equality subsystem. Similar to the hybrid algorithm proposed in [23], we compare these three relaxations at the root node to assess their strengths based on the size of the relaxation and the optimality gap at the root node, which is computed as $(UB - LB)/max(1, |UB|)$, where UB and LB are the upper and lower bounds at the root node. In this context, the *size of the relaxation* is defined as the product of the number of variables and the number of RLT constraints, where the latter is the sum of RLT bound factor constraints and two times of the RLT equality constraint-factor restrictions. Next, we multiply the size of the relaxation with the optimality gap at the root node, and a relaxation having the smallest such metric is chosen for branch-and-bound implementation. We call this implementation of constraint filtering techniques coordinated with the reduced basis techniques as *J-Hybrid*.

3 Semidefinite cuts

Recent works underscore the strength of SDP relaxations for polynomial programming problems (e.g., see [2,12,14]). However, SDP relaxations, or RLT relaxations enhanced with SDP constraints in the form of linear matrix inequalities, are computationally challenging to solve, as compared with LP-based RLT relaxations. Yet, as demonstrated by Anstreicher [3], a combination of RLT and SDP relaxations theoretically dominates many other alternative relaxations presented in the literature, which variously utilize convex envelopes, convex under-approximations such as the αBB approach [1], and purely SDP relaxations. Hence, instead of relying on SDP solvers, we enhance RLT-based LP relaxations using v-semidefinite cuts [22,24] that capture the strength of the underlying SDP relaxation, but retain linearity, and thereby facilitate the use of robust, efficient LP solvers. Given an RLT relaxation, such cuts are generated by imposing positive semidefiniteness on suitable dyadic variable-product matrices, where, given that the resulting matrix evaluated at the current LP relaxation solution is not positive semidefinite, the procedure automatically generates a cut to delete this LP solution. There are several possible alternatives for deriving such variable-product matrices for imposing positive semidefiniteness, thus yielding different classes of cuts.

Sherali and Fraticelli [24] tightened the standard RLT relaxations for solving quadratic programming problems via semidefinite cuts by imposing positive semidefiniteness on the following dyadic variable-product matrices:

$$M_0 = \left[xx^t\right]_L \quad \text{and} \quad M_1 = \left[x_{(1)}x_{(1)}^t\right]_L,$$

where $x_{(1)}^t = [1\ x^t]$. Let $\bar{M}_0$ and $\bar{M}_1$ evaluate M_0 and M_1, respectively, at the solution $(\bar{x}, \bar{X})$ to the relaxed problem. Employing the superdiagonalization algorithm for checking positive semidefiniteness (see [5]), the authors constructed a polynomial-time routine of complexity $\mathcal{O}(n^3)$, where n denotes the number of variables in the quadratic program. This routine automatically generates SDP cuts in the form:

$$\alpha_0{}^t M_0 \alpha_0 \geq 0 \quad \text{or} \quad \alpha_1{}^t M_1 \alpha_1 \geq 0,$$

whenever the respective linear matrix inequality $\bar{M}_0 \succeq 0$ or $\bar{M}_1 \succeq 0$ is violated. Sherali et al. [22] further extended the class of SDP cuts for enhancing RLT relaxations for general polynomial programming problems to so-called *v-semidefinite* cuts by imposing positive semidefiniteness on dyadic variable-product matrices of the form $M = [vv^t]_L$ predicated on suitable v-vectors, including the following:

$$v^{(1)} = [\,1, \text{all monomials of order } \Delta \text{ using } x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^{1+\binom{n+\Delta-1}{\Delta}}, \tag{19a}$$

$$v^{(2)} = \begin{bmatrix} 1, \{x_1, x_2, \ldots, x_n\}, \{\text{all quadratic monomials using } x_1, x_2, \ldots, x_n\}, \ldots, \\ \{\text{all monomials of order } \Delta \text{ using } x_1, x_2, \ldots, x_n\} \end{bmatrix}^T \in \mathbb{R}^{\binom{n+\Delta}{\Delta}}, \tag{19b}$$

$$v^{(3)} = \begin{bmatrix} 1, \{x_j,\ j \in J^*\}, \{\text{all quadratic monomials using } x_j,\ j \in J^*\}, \ldots, \\ \{\text{all monomials of order } \Delta \text{ using } x_j,\ j \in J^*\} \end{bmatrix}^T,$$

$$\text{where } J^* \in \arg\max \left\{ \sum_{\substack{r \in \{0,1,\ldots,R\}: \\ J_{rt} = J \text{ for some } t \in T_r}} \left| \alpha_{rt} \left(\bar{X}_{J_{rt}} - \prod_{j \in J_{rt}} \bar{x}_j \right) \right| : J \subseteq \mathcal{N}^\delta \right\}. \tag{19c}$$

Given a variable product matrix that is constructed via some v-vector in (19), Sherali and Fraticelli [24] proposed utilizing look-ahead, diagonal-sort, or full permutation re-arrangement strategies for SDP cut generation, whereas Sherali et al. [22] introduced a (restricted) eigenvalue-eigenvector strategy. Sherali et al. [22] also generated constraint-factor SDP cuts by imposing positive semidefiniteness on $[(\phi_r(x) - \beta_r)vv^t]_L$, where v is in the form of $v^{(1)}$ having a suitable degree, and where $\phi_r(x) \geq \beta_r$ is selected among violated constraints such that it has the maximum discrepancy between the original constraint evaluated at $\bar{x}$ and the linearized counterpart evaluated at $(\bar{x}, \bar{X})$. A composite cut generation strategy using $v^{(2)}$ (or $v^{(3)}$ for sparse problems) and constraint-factor SDP cuts was found to be highly effective, and is therefore chosen for implementation in our RLT-based optimization code.

For the illustrative example problem PP given by (15), two candidate dyadic product-variable matrices based on $v^{(3)}$ corresponding to J^* equal to $\{3, 5\}$ and $\{4, 4\}$ respectively, are given as follows:

$$\begin{pmatrix} 1 & x_3 & x_5 & X_{33} & X_{35} & X_{55} \\ x_3 & X_{33} & X_{35} & X_{333} & X_{335} & X_{355} \\ x_5 & X_{35} & X_{55} & X_{335} & X_{355} & X_{555} \\ X_{33} & X_{333} & X_{335} & X_{333} & X_{3335} & X_{3355} \\ X_{35} & X_{335} & X_{355} & X_{3335} & X_{3355} & X_{3555} \\ X_{55} & X_{355} & X_{555} & X_{3355} & X_{3555} & X_{5555} \end{pmatrix} \text{ and } \begin{pmatrix} 1 & x_4 & X_{44} \\ x_4 & X_{44} & X_{444} \\ X_{44} & X_{444} & X_{4444} \end{pmatrix}.$$

Let v_1 and v_2 denote the respective corresponding v-vectors. Assume that the dyadic variable-product matrix $M = [v_2(v_2)^t]_L$ is used in concert with $\alpha_2 \in \mathbb{R}^3$ to generate

SDP cuts in the form of ${\alpha_2}^t M \alpha_2 \geq 0$. If the third entry of α_2 is nonzero, the coefficient of X_{4444} in the SDP cut becomes nonzero. While the standard RLT relaxation includes bound-factor constraints involving X_{4444}, neither the J-set nor the $N_J^{d_J}$-set discussed in Sect. 2 includes these constraints. Whereas this does not affect theoretical convergence to a global optimum, it might lessen the effectiveness of SDP cuts, now that X_{4444} is only restricted via simple lower and upper bounds. A similar argument is valid for X_{444} if the second and third entries of α_2 are nonzero.

Next, consider the variable-product matrix $M = [v_1(v_1)^t]_L$ and let $\alpha_1 \in \mathbb{R}^6$ be the vector that defines the generated SDP cut. If the second or third entry of α_1 is nonzero, then the coefficient of X_{33} or X_{55} in the SDP cut becomes nonzero. While the $N_J^{d_J}$-set of constraints includes bound-factor constraints involving X_{33} and X_{55}, the J-set of constraints does not include these restrictions. Once again, the effectiveness of the generated SDP cut might be lessened when appended to the J-set-based relaxation.

One way to ensure effectiveness of the derived SDP cuts is to generate additional bound-factor constraints related to new monomials that are introduced through the SDP cut generation process. However, this approach is in conflict with the philosophy of constraint filtering routines since the relaxation might quickly expand to the form of full standard RLT relaxation. Alternatively, we could use $\Delta_{J^*} = \lfloor |J^*|/2 \rfloor$ in lieu of Δ in (19c) when the J-set or $N_J^{d_J}$-set are implemented, and which results in SDP cuts based on $v^{(4)}$ defined as follows:

$$v^{(4)} = \left[\begin{array}{c} 1, \{x_j,\, j \in J^*\}, \{\text{all quadratic monomials using } x_j,\, j \in J^*\}, \ldots, \\ \{\text{all monomials of order } \Delta_{J^*} \text{ using } x_j,\, j \in J^*\} \end{array} \right]^T,$$

$$\text{where } J^* \in \arg\max \left\{ \sum_{\substack{r \in \{0,1,\ldots,R\}: \\ J_{rt} = J \text{ for some } t \in T_r}} |\alpha_{rt}(\bar{X}_{J_{rt}} - \prod_{j \in J_{rt}} \bar{x}_j)| : J \subseteq \mathcal{N}^\delta \right\}. \quad (20)$$

Note that the size of the matrices based on $v^{(4)}$ are smaller than the size of the matrices based on $v^{(3)}$. Instead of using one $v^{(4)}$-vector, we found it computationally effective in preliminary tests to list the monomials in decreasing order of violation with respect to their respective defining identities in (4), and then generate SDP cuts using the first of min $\{10, \lceil \text{the number of existing nonlinear monomials}/2 \rceil\}$ monomials sequentially.

As the number of existing monomials increases, it becomes more advantageous to use $v^{(2)}$ instead of $v^{(4)}$, whereby we impose semidefiniteness using all monomials defining $v^{(2)}$ versus targeting individual monomials identified by J^* in (20). In our computations, the ratio between the existing nonlinear monomials (including the ones introduced via bound-factor constraints) and all possible nonlinear monomials of order up to δ is used to set a threshold. If this ratio is greater than 0.2, we recommend using $v^{(2)}$ for generating SDP cuts.

The implemented SDP cut generation process commences with the evaluation of the aforementioned dyadic variable-product matrices at the optimal solution $(\bar{x}, \bar{X})$ to the LP relaxation at a node subproblem and continues with checking the semidefiniteness of these matrices. Note that multiple matrices are utilized for SDP cut generation via

$v^{(4)}$-vectors. The cut generation cycle is completed by re-optimizing the LP relaxation enhanced by the resulting SDP cuts. The SDP cut generation cycle could be repeated a specified maximum number of times or until the identified variable-product matrices evaluated at $(\bar{x}, \bar{X})$ become positive-semidefinite. Due to the cost of re-solving the LP relaxation during each cycle, we perform at most one SDP cut cycle at each node as explained in detail in Sect. 3.1.

The SDP cuts that are generated for any node subproblem are valid at the other nodes of the branch-and-bound tree. In order to limit the number of SDP cuts appended to an LP relaxation and to select a set of effective SDP cuts among all generated ones, we implement a *cut inheritance scheme*. In this scheme, for each node subproblem, we manage four sets of SDP cuts: the cuts inherited from parent and other upstream nodes (C^1_{SDP}), the cuts generated at the current node (C^2_{SDP}), the cuts generated at the sibling node (C^3_{SDP}), and the cuts that are to be subsequently appended to the child nodes (C^4_{SDP}). When we solve the initial RLT-based LP relaxation at a node subproblem, we append the available cuts in C^1_{SDP} and C^3_{SDP}, if any. Based on the resulting optimal solution, we generate the set of (C^2_{SDP}) SDP cuts and append them to the current RLT relaxation. After re-optimizing the resulting LP relaxation, we compose the set of cuts (C^4_{SDP}) to be inherited by the child nodes as (C^2_{SDP}) $\cup$ (C^3_{SDP}) $\cup$ { the set of active cuts among C^1_{SDP}}. When the two child nodes are created, the cuts in C^4_{SDP} at the parent node are used to define the set C^1_{SDP} corresponding to the child nodes. With this cut inheritance scheme, we aim to reuse promising cuts that have the potential of being active at the current node subproblem.

3.1 Cut generation versus branching

In a branch-and-bound/cut algorithm, one of the critical decisions is the timing and frequency of cut generation. In the context of mixed-integer programs, Balas et al. [4] studied the balance between cut generation versus branching decisions. One important observation emanating from this study is the correlation between the optimal cut generation frequency and the quality of cuts at the root node, defined as the Euclidean distance between the optimal solution to the LP relaxation and the cutting plane. If the cuts at the root node are of good quality, it is better to generate cuts relatively more frequently. Based on computational results using 16 test instances from the literature, they suggested generating cuts at every eight or sixteen nodes. As an alternative, they proposed an automatic strategy where the frequency of cut generation is determined based on the quality of cuts at the root node, the number of discrete variables, and the number of fractional variables at the root node.

In branch-and-bound algorithms, a long-tail convergence is often observed. The quality of cuts and the optimality gap closed by cuts diminishes as the algorithm progresses deeper into the branch-and-bound tree. The improvement in lower bounds upon branching as well as via SDP cuts generated at the root node are key indicators of the relative effectiveness of these two lower bound improvement devices. However, the relative effectiveness of these strategies might change along different branches of

the tree. With this in mind, we propose below a simple decision rule for choosing between branching and SDP cut generation.

Our objective is to automatically modify the cut generation routine along the branch-and-bound tree in order to solve the problems more effectively. As SDP cuts are incorporated, the computational effort spent at a node increases due to cuts inherited from upstream nodes as well as the re-optimization step executed after appending the SDP cuts generated at the current node, while a significant reduction in the number of nodes explored is expected due to tightened relaxations.

The bound improvements at the parent node as well as at other upstream nodes are combined in estimating the lower bound improvement with branching and with SDP cut generation by using exponential smoothing with a parameter value of 0.75. If branching is implemented, the immediate required computational effort is to solve two LP relaxations at the child nodes, while cut generation requires a re-optimization of the current LP relaxation. Hence, the expected improvement with branching should be at least twice the expected improvement with SDP cuts to dominate cut generation. If branching dominates SDP cut generation, the imposed limit on the maximum number of SDP cuts generated is reduced. If the number of nodes for which branching dominates SDP cut generation is at least three times more than the converse, SDP cuts are generated but the re-optimization is not executed. The optimality gap closed by the SDP cuts cannot be calculated when the re-optimization step is skipped. For these instances, we compare the average values of the dual variables associated with the bound-factor constraints and with the SDP cuts. If the former is greater than the latter, the limit on the maximum number of SDP cuts generated is reduced. If the number of nodes for which SDP cuts are found less effective than the bound-factor constraints exceeds the converse by a factor of three, SDP cut generation is terminated. This implementation of SDP cuts is called *Routine* 1. For Routine 2, we skip the re-optimization step throughout the algorithm, and generate and inherit SDP cuts when they perform well relative to bound-factor constraints as explained above. In order to assess the performance of these two routines, we introduce Routine 3 that generates cuts and re-optimizes the LP relaxation, and Routine 4 that generates and inherits SDP cuts, at each node subproblem. The performances of these routines are discussed in Sect. 4.

4 Computational analysis

Our discussion on constraint filtering and reduced basis techniques in Sect. 2 induces eleven types of RLT-based relaxations, including as foundational forms the standard RLT formulation in (5) and the J-set based formulation in (8). Table 1 summarizes these various reformulation strategies. In what follows, we shall refer to the branch-and-bound/cut procedures implemented using Formulation F as the *F algorithm*.

In our RLT-based branch-and-bound/cut algorithms, we utilized the dual optimizer of CPLEX 12.5 [11] for solving the LP relaxations (a discussion on using the barrier optimizer is presented in the sequel). The upper bounds on the polynomial programming problems were obtained via SNOPT Version 7 [9] by using the optimal LP relaxation solution as an initial point for local optimization. Whenever SNOPT failed to optimize the problem, we computed the largest possible value for each monomial

Table 1 RLT formulation strategies for polynomial programming problems

	Bound-factor constraints (base)	Bound-factor constraints (enhanced)	Equality constraints
RLT	(5e)		
RLT-E	(5e)		(6b)
RRLT	(10f)		(6b)
RRLT+	(10f)	(5e)\(10f) with $u_{(J_1,J_2)} > 0$	(6b)
RLT-NB	(10f) in $\mathbb{R}^{n-m}$		
RLT-Hybrid	Selects either the RRLT+ or RLT-NB		
J-set	(8) with J-set Routine		
J-RRLT	(14f)		(14e)
J-RRLT+	(14f)	(8) for K_{+duals} in Remark 2	(14e)
J-NB	(14f) in $\mathbb{R}^{n-m}$		
J-Hybrid	Selects among the J-set, J-RRLT+, and J-NB		

in the objective function subject to the given lower and upper bounds on the variables, which were then added to calculate an upper bound on the objective function (assuming feasibility). Note that such an upper bound is typically very loose and the RLT relaxations that provide tight lower bounds would be treated unfairly while selecting the underlying relaxation for composing the hybrid algorithms; hence, the selection of an effective local optimizer is important in this regard. Furthermore, we used Matlab (Version 2011a) [17] to determine the rank of the candidate matrices while implementing the reduced basis techniques. The CPU times reported in this section include the effort expended by Matlab, which happens to be insignificant in comparison to the LP solution times. All runs were made on two identically configured workstations having 3.1 GHz processor with 16 GB of RAM and running Windows 7.

In our implemented branch-and-bound/cut algorithm, we fathom a feasible node subproblem whenever $(UB - LB) \leq \epsilon \max\{1, |UB|\}$, where UB is the current upper bound, LB is the optimal solution value at the node relaxation, and ϵ is the prescribed optimality gap tolerance (we set $\epsilon = 0.01$ in our computational analysis). For branching, we select among the existing nodes one having the smallest lower bound, and identify the associated branching variable using the following rule:

$$j^* \in \arg\max_{j \in \mathcal{N}} \{\Theta_j\}, \text{ where } \Theta_j \equiv \theta_j \min\left\{u'_j - \bar{x}_j, \bar{x}_j - l'_j\right\},$$

$$\text{and where } \theta_j \equiv \sum_{J : J \cup j \text{ exists in PP}} \left\{ |\bar{X}_{J \cup j} - \bar{X}_J \bar{x}_j| + |\prod_{i \in J \cup j} \bar{x}_i - \bar{X}_J \bar{x}_j| \right\}, \forall j \in \mathcal{N}.$$

Here, $(\bar{x}, \bar{X})$ is the optimal solution to the LP relaxation at the selected node, θ_j represents a measure of the cumulative violation in the RLT defining identities (4) with respect to the variable x_j, and Θ_j is obtained by scaling the computed θ_j with the minimum distance to the current bounds l'_j and u'_j on the variable x_j from $\bar{x}_j$. Once a branching variable x_{j^*} is selected, two child nodes are created by partitioning the

Table 2 Characteristics of the test problems

(Degree (δ), # of variables (n), # of inequalities (R_1))	Problem density	# of linear equalities (m)	Density of the linear equality constraints
(2, 28, 10)	0.005	0	0.5
(3, 16, 9)	0.01	$n/4$	1
(4, 12, 7)	0.05	$n/2$	
(5, 8, 6)	0.1		
(6, 6, 6)	0.5		
(7, 5, 6)	1		

current bounding sub-hyperrectangle $\Omega' \equiv \{x : 0 \leq l'_j \leq x_j \leq u'_j, \forall j \in \mathcal{N}\}$ of the selected node into $\Omega' \cap \{x : l'_{j^*} \leq x_{j^*} \leq \bar{x}_{j^*}\}$ and $\Omega' \cap \{x : \bar{x}_{j^*} \leq x_{j^*} \leq u'_{j^*}\}$, if $\min\{u'_{j^*} - \bar{x}_{j^*}, \bar{x}_{j^*} - l'_{j^*}\} \geq 0.05(u'_{j^*} - l'_{j^*})$. Else, we use the midpoint $(l'_{j^*} + u'_{j^*})/2$ in lieu of $\bar{x}_{j^*}$ for partitioning.

In order to comparatively assess the 11 different model representations listed in Table 1 in a practical, controlled computational environment using test cases having specific problem structures, we randomly generated a suitable test-bed of instances of Problem PP. The relative effectiveness of constraint filtering techniques over the standard RLT is expected to be more significant for sparse problems. As the problems become denser, the former model representation approaches the latter, where the density of a problem is defined as the ratio between the existing nonlinear monomials within the problem formulation and all possible nonlinear monomials for the given number of variables. Similarly, the relative effectiveness of reduced basis techniques over the standard RLT is expected to be more significant for problems having high-rank linear equality constraint systems. The degree of the problem is also another parameter that interacts with the density of the problem and the number of linear equality constraints in determining the relative effectiveness of a model representation. Using the density of the problem, the number of linear equalities, the density of the linear equality constraints, the degree of the problem, the number of variables, and the number of polynomial inequality constraints as the primary parameters, we constructed a test-bed of problems according to the parameter choices specified in Table 2 in order to assess the different model representations. We generated 180 polynomial programming test instances in total, with 30 instances for each degree. The degree of the ith inequality constraint was set to $[(i-1) \mod \delta] + 1, \forall i = 1, \ldots, R_1$, whereas all equality constraints were taken as linear. The absolute objective and constraint coefficients were randomly generated on [0, 10], each with a 0.1 probability of being negative. The right-hand side value of each constraint was set equal to the value of the constraint expression evaluated at the mid-point of the variable ranges. The lower and upper bounds on each variable were generated randomly on [0, 1] and [2, 3], respectively. These polynomial programming problem instances are available for other researchers via the following google drive:https://googledrive.com/host/0B4xVNzR-k0aeU1VMZjQ2WjY3U0k.

4.1 Computational analysis of the J-Hybrid algorithm

In this section, we discuss the performance of the J-Hybrid algorithm, which is an implementation of the constraint filtering techniques coordinated with the reduced basis techniques as introduced in Sect. 2.4. The first part of the analysis investigates the effectiveness of the proposed relaxations generated by their corresponding routines, whereas the second part mainly deals with the algorithm selection among the J-set, the J-RRLT+, and the J-NB reformulations. Later in Sect. 4.3, we compare the J-Hybrid and the RLT-Hybrid algorithms, and develop simple rules for selecting a superlative RLT formulation among the ones listed in Table 1.

The computational results pertaining to the mentioned algorithms (which include the SDP cut generation Routine 2) are summarized in Table 3. This table displays for each algorithm, the number of problems solved while consuming the minimum CPU time among all three algorithms. Among 144 polynomial problems with linear equality constraints (24 of each degree), six were solved by the J-set algorithm at the root node. Hence, the algorithm terminated with the optimal solution and the reduced basis techniques were not implemented. For the remaining 138 problems, the J-set, the J-RRLT+, and the J-NB optimized 51, 43, and 46 problems, respectively, in the minimum CPU time among all three. Another observation is that the J-set was relatively more successful for lower degree problems, whereas the J-NB performed better than the others for higher degree problems. This result can be attributed to the significant reduction in the number of variables and constraints for higher degree problems when basic variables are eliminated via substitution.

We used the average CPU time as a second performance measure to assess the effectiveness of the algorithms. These results are summarized in Table 4. Even though the J-RRLT+ algorithm outperformed the J-set and J-NB in only 43 problems among 138, it provided the minimum average CPU time for each degree of problems. In contrast to solving 46 problems using the minimum time, the J-NB performed the worst of the three with respect to average CPU time. The results in Tables 3 and 4 demonstrate that the reduced basis techniques for sparse problems have the potential to improve the performance of the J-set algorithm. However, none of the algorithms have a uniform dominance over the others. Hence, the J-Hybrid algorithm was implemented, which selects a relaxation among the J-set, the J-RRLT+, and the J-NB based on the performance and size of the root node relaxation.

Table 3 The number of problems solved within the minimum CPU time among the J-set, J-RRLT+, and the J-NB algorithms (or reformulations)

Degree	J-set	J-RRLT+	J-NB	J-Hybrid (offline)
2	9	12	0	12
3	12	9	2	15
4	11	9	4	20
5	7	6	11	16
6	5	3	16	16
7	7	4	13	19
Total	51	43	46	98

Table 4 Average CPU time (in seconds) with the reduced basis techniques for sparse problems

Degree	J-set	J-RRLT+	J-NB	Minimum	J-Hybrid (offline)	J-Hybrid
2	171.0	168.3	309.4	117.8	139.7	138.7
3	192.5	161.5	277.1	80.6	128.2	125.0
4	231.8	228.0	316.5	110.1	123.5	136.3
5	169.4	102.8	199.7	41.9	58.3	66.7
6	94.8	41.0	97.7	22.6	40.9	46.8
7	131.2	71.9	180.1	35.7	59.8	65.4
Average	165.1	128.9	230.1	68.1	91.7	96.5

We now continue to assess the performance of the J-Hybrid algorithm. The trade-off between the success rate in selecting the best performing algorithm and the additional computational effort spent for the extended root node analysis determine the performance of the J-Hybrid algorithm. In order to investigate these two effects independently, we commence our analysis with a *J-Hybrid (offline)* procedure, for which the root node optimality gap and problem sizes for each algorithm were recorded, and the underlying relaxation was selected offline using the stated decision rule. The computational time of the selected algorithm was accordingly then assigned to the J-Hybrid (offline) algorithm. The J-Hybrid (offline) algorithm successfully identified the best performing algorithm for 98 out of 138 problems (Table 3) and reduced the average CPU times for problems of each degree in comparison to the other three algorithms (Table 4). These computational results demonstrate the effectiveness of the selection rule.

Furthermore, in order to select the underlying relaxation within an online implementation, the J-set, the J-RRLT, and the J-RRLT+ relaxations were constructed and solved in sequence at the root node. The two additional relaxations solved at the root node increased the computational time required by the J-Hybrid algorithm in comparison to the J-Hybrid (offline) (slight exceptions occurred when tolerance-check differences resulted in different SDP cuts being generated in the process). In order to control the additional effort, the maximum time to solve the J-RRLT and the J-RRLT+ root node relaxations were set to 10 times the effort required to solve the J-set root node relaxation. If a relaxation was not optimized due to the time limit, it was not included within the list of candidate relaxations for the J-Hybrid algorithm implementation. The resulting additional cost was negligible for problems of degree two and three, whereas it became significant, yet controllable, for higher degree problems as displayed in Table 4. Nevertheless, the J-Hybrid algorithm reduced the average computational effort in comparison to the J-set, the J-RRLT+, and the J-NB reformulations on average by 41.6, 25.2, and 58.1 %, respectively.

4.2 Computational analysis of SDP cut generation routines

We next compare the four SDP cut generation routines described in Sect. 3.1, where we also report the results when SDP cuts are not generated. Utilizing the J-Hybrid

Table 5 Performances of SDP cut generation routines

Degree	Average CPU time (in seconds)					Number of unsolved problems out of 30				
	No SDP	Routine 1	Routine 2	Routine 3	Routine 4	No SDP	Routine 1	Routine 2	Routine 3	Routine 4
2	120.2	104.6	104.5	104.6	104.7	6	6	6	6	6
3	148.6	118.7	111.1	122.2	122.4	6	3	3	4	4
4	177.1	134.5	129.7	142.9	143.3	5	3	3	3	3
5	126.3	67.6	67.4	81.8	72.0	3	1	0	1	1
6	72.6	43.6	41.3	46.8	44.3	2	0	0	1	1
7	97.8	71.5	65.1	75.9	69.5	3	2	2	2	2
Average	123.8	90.1	86.5	95.7	92.7	4.2	2.5	2.3	2.8	2.8

algorithm, the average CPU times and number of unsolved problems within the maximum allowed time limit (500 s) are displayed in Table 5. All cut generation routines improved the performance of the branch-and-bound/cut algorithm. Among the four routines, Routine 2 uniformly dominated others for solving problems and was chosen for implementation.

4.3 RLT-POS for solving randomly generated test problems

In this section, we aim to develop some simple selection rules for identifying a superlative RLT formulation to implement within the branch-and-bound/cut algorithm. In the sequel, we discuss and compare the **RLT-Hybrid** and the J-**Hybrid** algorithms composed respectively of {RRLT+ and RLT-NB} and {J-set, J-RRLT+, and J-NB}, and propose a simple selection rule based on the computational results using the aforementioned randomly generated polynomial programming instances. The hypothesis is that the RLT-Hybrid algorithm performs better than the J-Hybrid when the problem is dense and the ratio of the rank of the equality system to the number of original variables is relatively large. These methods are later evaluated in Sect. 4.6 using test problems from the literature.

4.3.1 Problems without equality constraints

We first conduct a computational comparison of the RLT-Hybrid and the J-Hybrid algorithms using polynomial programming test problems that have no linear equality constraints. Table 6 presents the results obtained. The RLT-E, RRLT+, and RLT-NB relaxations reduce to the RLT relaxation for problems without linear equality constraints. Similarly, the J-RRLT+ and J-NB relaxations reduce to the J-set relaxation for the mentioned problems. Hence, the RLT-Hybrid and J-Hybrid algorithms essentially adopt the RLT and J-set relaxations, respectively, which is indicated within parentheses in Table 6.

The J-Hybrid algorithm performed at least as well as the RLT-Hybrid algorithm for solving these problems, except for six out of thirty-six instances, as displayed in

Table 6 Performance of the RLT algorithms for solving polynomial problems without equality constraints (in CPU seconds)

Degree	Algorithm	Problem density					
		0.005	0.01	0.05	0.1	0.5	1
Two	RLT-Hybrid (RLT)	0.05	0.06	0.06	0.17	0.75	499.2
	J-Hybrid (J-set)	0.03	0.03	0.03	0.08	0.41	499.1
Three	RLT-Hybrid (RLT)	0.17	0.19	0.83	11.5	331.5	78.6
	J-Hybrid (J-set)	0.05	0.08	0.31	4.8	375.5	77.6
Four	RLT-Hybrid (RLT)	24.8	59.0	222.8	81.7	69.4	500.1
	J-Hybrid (J-set)	0.7	0.7	31.5	29.0	55.4	500.0
Five	RLT-Hybrid (RLT)	412.5	158.8	78.5	21.2	137.2	123.6
	J-Hybrid (J-set)	4.9	2.9	4.6	7.8	202.1	123.1
Six	RLT-Hybrid (RLT)	113.0	46.7	46.2	79.5	28.5	43.3
	J-Hybrid (J-set)	1.4	0.5	9.6	23.0	39.3	44.1
Seven	RLT-Hybrid (RLT)	30.7	298.5	146.1	311.1	90.9	64.6
	J-Hybrid (J-set)	0.2	6.9	21.0	232.6	51.5	72.2

Table 6. The J-Hybrid algorithm reduced the computational effort required by RLT-Hybrid by 39, 40, 61, 51, 51, and 57 %, on average, for solving degree-two, -three, -four, -five, -six, and -seven problems, respectively. We therefore suggest implementing the J-Hybrid algorithm whenever the polynomial programming problem does not have any linear equality constraints.

For future reference, note that the RLT-NB formulation is obtained by eliminating the basic variables via substitution, where the resulting model does not have any linear equality constraints. On the basis of the superiority of the J-set formulation over the standard RLT as displayed in Table 6, the J-set of constraints were generated for implementing RLT-NB (likewise for RLT-Hybrid whenever RLT-NB is selected as the underlying formulation). The only difference between the RLT-NB and the J-NB implementations is that the LP relaxation solved at the root node for selecting a basis is given by the RLT-E and the J-set relaxations, respectively.

4.3.2 Problems with equality constraints

We next compare the RLT-Hybrid and J-Hybrid algorithms for solving problems having equality constraints. The density of the linear equality constraints was not found to be significantly influential. For clarity of presentation, the average CPU time required for solving instances with the same problem density and the number of equality constraints are reported in Tables 7 and 8.

Before presenting the computational results, we discuss the two instances for which CPLEX experienced numerical problems while implementing the RLT-Hybrid algorithm. In this context, recall that RLT-Hybrid algorithm solves the root node RLT-E relaxation, based on which, either the RRLT+ or RLT-NB relaxation is adopted. The root node RLT-E relaxation of one particular degree-three problem having $n/2$ equal-

ity constraints with a unit density value was found to be infeasible using CPLEX's default feasibility tolerance of 10^{-6}. When the feasibility tolerance was increased to 10^{-5}, the problem was successfully solved by the RLT-E algorithm within 72.7 CPU seconds, which is the result used within Table 7. In addition, CPLEX experienced numerical problems while solving one degree-seven problem with the RLT-NB-based branch-and-bound algorithm, which terminated with a 4.7 % optimality gap after running for 500 CPU seconds (the maximum allowed time). This result is used while compiling Table 8.

We commence the analysis with results for quadratic and cubic problems with equality constraints. Table 7 presents the related computational results. The RRLT+ and RLT-NB were found to be not as effective as the RLT-E algorithm. The weakening effect of the reduced basis techniques on the RLT relaxations is more prominent than the reduction in the computational effort required to solve node relaxations. Therefore, we suggest using the RLT-E algorithm instead of RLT-Hybrid for solving quadratic and cubic problems having equality constraints. The particular selection between the RLT-E and J-Hybrid algorithms is based on the density of the problem and the rank of the equality system. We partition the problems into three ranges with respect to the density of the problems: (0, 0.04], (0.04, 0.4], (0.4, 1]. If the density of the problem is within the first or the third regions, we suggest implementing the J-Hybrid and the RLT-E algorithms, respectively. If the problem falls in the second region, we check the rank of the equality system. If the number of equality constraints is less than $3n/8$, we implement the J-Hybrid algorithm; else, we select the RLT-E algorithm. The resulting procedure composed via these rules is called the *Proposed Algorithm (RLT-POS)* in Table 7, and is summarized in the flow-chart presented in Fig. 1 for convenience in reference.

We continue the computational analysis by considering higher order polynomial programming problems having linear equality subsystems. Table 8 presents the results obtained. Whereas the J-Hybrid algorithm performed better than the RLT-Hybrid algorithm for sparse problems, the latter required less effort to optimize dense problems. Hence, we propose the J-Hybrid algorithm for solving problems having density less than 0.04, and the RLT-Hybrid algorithm for denser problems, (this is referred to as the *Proposed Algorithm (RLT-POS)* in Table 8, and is likewise displayed in Fig. 1).

The Proposed Algorithm delineated in Fig. 1 (hereinafter referred to as **RLT-POS**) adopts the J-set relaxations whenever the problem does not have equality constraints. For the quadratic and cubic problems, either the RLT-E or the J-Hybrid algorithm is implemented based on the density and the number of linear equality constraints. For higher degree problems with linear equality constraints, we select either the RLT-Hybrid or the J-Hybrid algorithm depending on the density of the problem.

4.4 Comparison of the dual and barrier optimizers of CPLEX

When RLT-based LP relaxations of size in the order of 10^3 variables and 10^4 constraints were solved with the barrier and dual simplex optimizers of ILOG CPLEX 12.5, the former solver option reduced the computational effort for a significant portion of the test problems in comparison to the latter. For the remaining problems, however,

Table 7 Performance of the RLT algorithms for solving quadratic and cubic problems with equality constraints (in CPU seconds)

# of equality cons.		$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	Avg.
Degree	Density	0.005	0.005	0.01	0.01	0.05	0.05	0.1	0.1	0.5	0.5	1	1	
Two	RLT-E	0.17	0.31	0.18	0.96	0.87	1.72	2.97	14.6	48.9	23.1	251.0	10.5	29.6
	J-Hybrid	0.05	0.09	0.05	0.27	0.16	1.41	0.77	22.4	268.2	499.1	251.5	273.5	109.8
	Proposed Alg.	0.05	0.10	0.05	0.37	0.16	1.72	0.71	14.6	48.8	23.1	251.0	10.6	29.3
Three	RLT-E	8.28	257.5	286.2	74.6	37.8	78.2	86.7	36.4	148.5	180.0	46.4	73.4	109.5
	J-Hybrid	0.13	1.9	13.1	10.7	8.3	148.9	40.6	168.6	499.2	116.3	323.7	105.7	119.8
	Proposed Alg.	0.14	2.0	13.6	11.0	8.1	78.7	41.3	36.9	149.3	180.2	47.3	72.1	53.4

Table 8 Performance of the Hybrid RLT algorithms for solving degree-four, -five, -six, and -seven problems with equality constraints (in CPU seconds)

# of equality cons.		$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	$n/4$	$n/2$	Avg.
Degree	Density	0.005	0.005	0.01	0.01	0.05	0.05	0.1	0.1	0.5	0.5	1	1	
Four	RLT-Hybrid	258.0	54.8	341.4	20.6	137.4	21.1	146.0	19.5	225.8	125.3	128.3	30.5	125.7
	J-Hybrid	0.7	10.1	47.1	85.8	107.5	266.1	56.3	359.6	144.4	67.8	281.9	209.0	136.3
	Proposed Alg.	0.6	9.6	46.9	87.8	137.5	21.2	146.2	19.5	226.3	125.6	128.0	30.7	81.7
Five	RLT-Hybrid	28.2	23.9	11.8	5.9	67.3	4.4	36.0	5.3	51.3	5.0	93.0	5.6	28.1
	J-Hybrid	13.2	7.9	2.0	24.6	109.4	23.7	244.6	161.5	75.4	17.7	91.6	29.1	66.7
	Proposed Alg.	13.4	8.1	2.2	24.9	67.3	4.4	36.1	5.5	51.5	5.3	92.9	5.8	26.4
Six	RLT-Hybrid	16.2	4.8	14.1	5.2	62.8	4.2	95.7	4.1	39.7	4.1	26.1	8.6	23.8
	J-Hybrid	1.9	2.4	2.9	6.2	41.8	4.0	95.3	22.5	270.9	18.2	79.1	16.0	46.8
	Proposed Alg.	2.0	2.5	3.1	6.3	62.3	3.9	95.3	4.2	39.5	4.4	26.2	9.1	21.6
Seven	RLT-Hybrid	19.4	9.0	19.9	8.4	17.6	8.3	261.4	22.0	26.1	9.1	29.8	10.9	36.8
	J-Hybrid	2.1	2.7	6.1	2.8	23.5	268.1	272.0	57.6	32.5	16.0	76.1	25.2	65.4
	Proposed Alg.	2.1	2.6	6.4	2.6	16.6	7.4	260.9	21.2	25.2	8.6	29.3	10.7	32.8

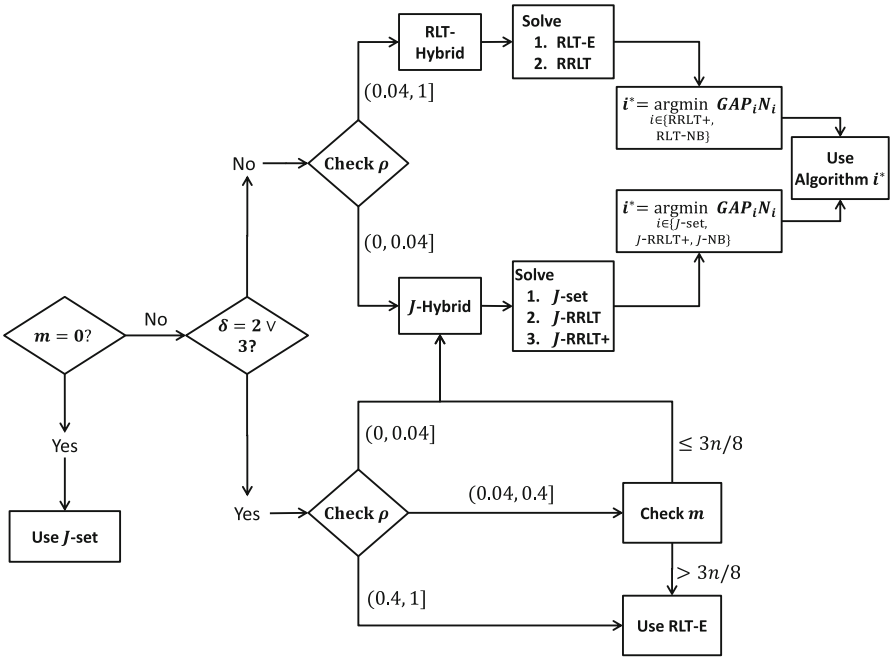

Fig. 1 Flow-chart for composing the proposed algorithm: RLT-POS

we observed extremely long computational times due to the crossover step in the barrier optimizer, which is required to construct an optimal basis for obtaining a set of basic primal and dual solutions. There are two main problems with incomplete dual information for the reduced RLT algorithms. First, a basis B of A is selected to partition the variables into x_B and x_N using primal and dual information. The lower bounds obtained via the reduced RLT formulations in (10) and (14) using a basis constructed via incomplete dual information are generally worse than the ones obtained with complete dual information. Second, the dual information is utilized to select the bound-factor constraints that have associated positive duals at the root node to enhance the reduced RLT formulations as per Remark 2. With incomplete information, almost all bound-factor constraints have positive dual values at the root node. To overcome the first problem, we perform the crossover step at the root node only when we implement the reduced basis techniques. For the second issue, we set two times the average dual value as a threshold and append those bound-factor constraints involving at least one basic variable only if their associated dual values are greater than the computed threshold at the root node.

In order to compare the dual and barrier optimizers of CPLEX, we computed the percentage CPU time improvement of the latter with respect to the former as: $100\frac{\text{CPU}_D - \text{CPU}_B}{\text{CPU}_D}\%$, where CPU_D and CPU_B denote the CPU time required to solve the instance with the dual and the barrier optimizers, respectively. We then computed the average of these percentage CPU time improvement values for each set of problems having the same degree and density. The results are displayed in Fig. 2. The barrier

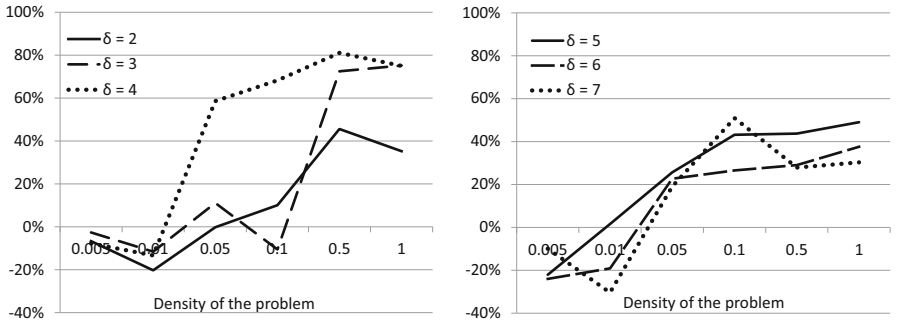

Fig. 2 Average percentage CPU time improvement for the barrier optimizer in comparison to the dual optimizer of CPLEX for solving polynomial programming problems using RLT-POS

optimizer spent 2–30 % more computational effort for solving problems having densities 0.005 and 0.01 in comparison to the dual optimizer of CPLEX. The relative effectiveness of the barrier optimizer increases as the problems become denser. For completely dense problems, the barrier optimizer spent 30–75 % less effort compared to the dual optimizer. Nonetheless, considering the occasional numerical problems we experienced with the barrier optimizer, we suggest using the dual optimizer for the sake of robustness, despite the reduction in required computational effort with the former solver. Notwithstanding this conservative choice, the comparison in Fig. 2 clearly demonstrates the barrier optimizer's potential for solving polynomial programming problems.

4.5 Comparison with BARON, COUENNE, and SparsePOP

In this section, we perform a computational analysis using a new set of randomly generated test instances to compare the performances of BARON (Version 9.3.1) [29], COUENNE (Version 0.3) [6], and SparsePOP (Version 2.99) [31] with RLT-POS. The new test-bed of problems was generated using the parameters listed in Sect. 4. In particular, (the degree of the problems, the number of variables, the number of inequality constraints, and the number of equality constraints) for these problems were specified as (3, 9, 9, 0), (3, 9, 7, 2), (4, 9, 9, 0), (4, 9, 7, 2), (6, 7, 6, 0), and (6, 7, 6, 1). For each specific size, we generated three problems with densities 0.05, 0.1, and 1.

The CPU times for solving the 18 problems with BARON, COUENNE, and RLT-POS are displayed in Fig. 3. Each problem is represented separately, where the degrees of the problems are differentiated by the marker types as displayed in the legend. The problems with densities 0.05, 0.1, and 1 are represented with small, medium, and large marker sizes, respectively. Whereas BARON was more successful in solving sparse problems, RLT-POS dominated BARON for the more dense test problems. Among the 18 problems, RLT-POS performed better than BARON for 10 problems and dominated COUENNE for all the problems. BARON, COUENNE, and RLT-POS failed to optimize five, eight, and two problems, respectively, within the maximum

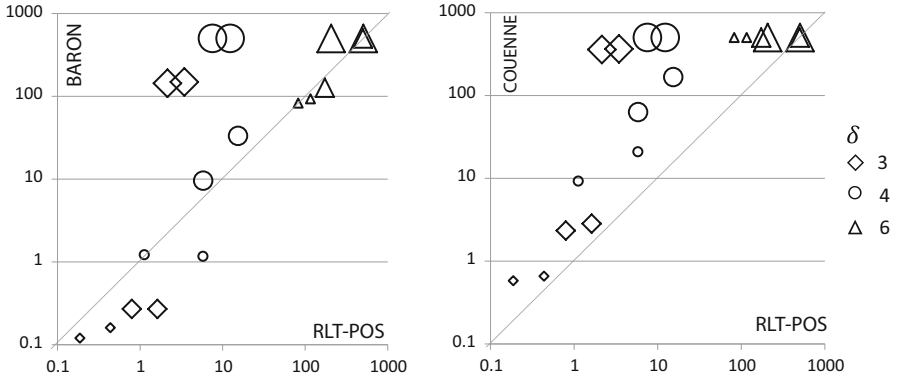

Fig. 3 The required computational effort for solving polynomial programming instances with RLT-POS, BARON, and COUENNE on a logarithmic scale (in seconds). The *line* $x = y$ is plotted for reference

allowed time. RLT-POS reduced the average optimality gap to 4.8 % in comparison to 36.5 % for BARON and 125.0 % for COUENNE for the two common unsolved test problems. The other three problems for which BARON and COUENNE terminated prematurely after the time limit of 500 CPU seconds, the average optimality gaps were 28.6 and 83.3 %, respectively, whereas RLT-POS solved these three instances within 75.7 CPU seconds, on average. The three problems for which COUENNE failed to optimize to the desired accuracy were solved with BARON and RLT-POS within 101.0 and 124.2 CPU seconds, on average, respectively. The remaining 10 problems were optimized within 33.9, 98.9, and 3.7 CPU seconds, on average, using BARON, COUENNE, and RLT-POS, respectively.

We next compare RLT-POS with SparsePOP v2.99, which exploits sparsity within polynomial optimization problems and generates relatively tractable SDP relaxations that foster the solution of polynomial optimization problems. SparsePOP was run in Matlab R2011a utilizing SeDuMi 1.3 [28] as the SDP solver. SparsePOP cannot be directly implemented to run with a specified termination percentage optimality gap. Hence, we first ran SparsePOP with the default parameter values and computed the % optimality gap using the upper bound obtained by RLT-POS and the objective value of the lower bounding SDP relaxation corresponding to the default relaxation order. If the resulting gap was not within the specified tolerance of 1 %, we incremented the relaxation order by one and re-solved the problem until the desired accuracy of 1 % was achieved. Table 9 displays the average CPU time and average % optimality gap attained by SparsePOP, along with the computational effort required by RLT-POS. For clarity of presentation, we have partitioned the set of problems as those that were solved and those that were not solved to the desired accuracy by SparsePOP for each relaxation order. Since RLT-POS solved all the problems to the desired accuracy of 1 %, the optimality gaps for RLT-POS are not reported.

SparsePOP was not able to solve any of the 18 instances to the desired accuracy with the default relaxation order. In particular, SparsePOP terminated with an optimality gap of 481.7 % within 18.7 CPU seconds, on average, using the default SDP relaxation order, whereas these instances were solved by RLT-POS in 168.9 CPU seconds, on

Table 9 Average CPU time (in seconds) and % optimality gap for solving sparse problems using SparsePOP along with increasing relaxation orders and using RLT-POS

	Solved to desired accuracy by SparsePOP				Not solved to desired accuracy by SparsePOP			
Relaxation	# Of	SparsePOP		RLT-POS	# Of	SparsePOP		RLT-POS
Order	instances	CPU time	% opt. gap	CPU time	instances	CPU Time	% opt. gap	CPU time
Default	0				18	18.7	481.7	168.9
+1	14	2249.9	0.07	142.1	4	1525.0	17.0	262.8

average. In order to solve the instances to the desired accuracy as stated above, the relaxation order was increased by one and SparsePOP was able to solve 14 of the 18 instances within 2249.9 CPU seconds, on average, whereas RLT-POS optimized these 14 instances within 142.1 CPU seconds, on average, which is a 15-fold speedup. However, note that the average optimality gap using SparsePOP was 0.07 %, whereas that for RLT-POS (using the specified 1 % tolerance) was actually 0.82 % at termination. For the remaining four instances, SparsePOP consumed 1525.0 CPU seconds and terminated with an average optimality gap of 17.0 %, in comparison to 262.8 CPU seconds for RLT-POS, on average. When the relaxation order was further increased by one, SparsePOP experienced out-of-memory problem for these four instances.

4.6 Computational experience with solving problems from the literature

In this section, we present a computational analysis for solving problems from the literature via the RLT algorithms listed in Table 1, the SDP routines introduced in Sect. 3.1, the Proposed Algorithm RLT-POS, as well as with BARON, COUENNE, and SparsePOP. Tables 10, 11, and 12 display the results obtained. We report the com-

Table 10 Performance of the RLT algorithms for solving linear equality constrained problems from the literature

Algorithm	PP1	PP2	PP3	PP4	PP5
RLT	95.3	39.6 %*	Memory Err.	209.7	2.6
RLT-E	N/A	Unknown Err.	Memory Err.	39.5	0.5
RRLT+	N/A	Unknown Err.	Memory Err.	8.4	0.7
RLT-NB	N/A	Unknown Err.	Memory Err.	3.1	0.3
RLT-Hybrid	N/A	Unknown Err.	Memory Err.	4.6	0.4
J-set	5.5	51.4	239.6	1.3	1.3
J-RRLT+	56.7	24.9 %*	Memory Err.	2.8	0.4
J-NB	3.4E+04 %*	1.6E+08 %*	Memory Err.	Num.Err.	0.7
J-Hybrid	9.6	83	Memory Err.	2.7	0.8
RLT-POS	9.6	83	239.6	2.7	0.4

* Percentage optimality gap at termination

Table 11 Performance of RLT-POS in concert with the differentSDP cut routines for solving linear equality constrained problems from the literature (in CPU seconds)

SDP routine	PP1	PP2	PP3	PP4	PP5
Routine 0	58.8	387.9	5.9%*	68.3	0.9
Routine 1	16.5	98.1	355.4	0.7	0.4
Routine 3	14.5	86.2	367.3	0.6	0.4
Routine 4	9.3	84.8	278.4	0.5	0.4
Routine 2	9.6	83	239.6	2.7	0.4

* Percentage optimality gap at termination

putational time whenever the problem was solved within the specified time limit of 500 CPU seconds; else, the optimality gap at termination is specified. The particular errors reported by CPLEX upon premature termination such as out-of-memory, unknown, or numerical errors are also indicated.

Problem PP1 is a degree-four polynomial problem having 16 variables and 8 linear equality constraints [10]. Using default parameter values, the dual optimizer of CPLEX found the root node RLT-E relaxation infeasible, whereas the relaxation was successfully solved when the feasibility tolerance was changed to 10^{-5} from its default value of 10^{-6}. Since it took more than 1400 CPU seconds to solve the RLT-E root node relaxation, the optimality gap at termination is not reported (indicated in the table with N/A). Given that the RRLT+, the RLT-NB, and the RLT-Hybrid relaxations are based on the RLT-E root node relaxation, the notation *N/A* is also specified for these algorithms in Table 10. Based on its root node analysis, the J-Hybrid algorithm selected the J-set relaxation. Problem PP1 has only 168 of the possible 4828 nonlinear monomials and due to the resulting density parameter value of 0.034, RLT-POS selected the J-Hybrid algorithm and solved the problem in 9.6 CPU seconds.

We also solved Problem PP1 using RLT-POS algorithm in concert with the different SDP cut routines (where the proposed default choice as noted previously in Sect. 4.2 is Routine 2). Note that when the SDP cut generation step was entirely skipped (Routine 0), the CPU time increased 6-fold in comparison to Routine 2 (see Table 11).

In order to assess the effect of problem degree and size, we modified the objective function of Problem PP1 and derived Problems PP2 and PP3, which are listed in the "Appendix". Problem PP2 is a degree-five polynomial program having the same constraint set as Problem PP1 and a density of 0.009. The RLT relaxation for Problem PP2 has more than 376,000 bound-factor constraints and 20,000 variables, and the standard RLT algorithm terminated prematurely with a 39.6 % optimality gap at the end of the set time limit. More than 38,000 equality constraint-factor restrictions were appended to the RLT relaxation to obtain the RLT-E relaxation. Both the dual and barrier optimizers of CPLEX terminated with an "*unknown error*" while solving the RLT-E root node relaxation. Hence, the performance statistics for the RLT-E, RRLT+, RLT-NB, and the RLT-Hybrid algorithms are not reported. The J-set algorithm optimized Problem PP2 in 51.4 CPU seconds. The J-RRLT+ and J-NB algorithms terminated prematurely with optimality gaps of 24.9 and 1.6E+8 %, respectively. Based on its root node analysis, the J-Hybrid algorithm selected the J-set as the underlying relaxation and optimized the problem in 83 CPU seconds. Since only 184 of the possible 20,332 nonlinear monomials exist in Problem PP2, RLT-POS identified J-Hybrid as the underlying algorithm in its automatic selection process. Moreover, all the SDP

cut generation routines in concert with RLT-POS outperformed the implementation without SDP cuts by reducing the computational effort by at least 3.9-fold.

Problem PP3 is a degree-six polynomial program having the same constraints as for Problem PP1 and a density of 0.005. The RLT relaxation for Problem PP3 has more than 2.3 million bound-factor constraints and 74,000 variables. The number of equality constraints in (6b) for this instance exceeds 162,000. Due to the size of the relaxations, CPLEX terminated with "*out-of-memory error*" while generating the RLT and the RLT-E root node relaxations. The J-set filtering algorithm successfully constructed a tractable relaxation based on which PP3 was optimized within 239.6 CPU seconds, whereas an out-of-memory error termination resulted while generating the reduced RLT relaxations for sparse problems. Hence, the J-set relaxation was selected by both the J-Hybrid and the RLT-POS algorithms. Furthermore, when the SDP cut generation step was not implemented, RLT-POS terminated with an optimality gap of 5.9 % after the time limit of 500 CPU seconds.

Problem PP4 is a degree-six problem having five variables and two linear equality constraints (see the "Appendix"). The RLT-Hybrid algorithm selected the RLT-NB relaxation based on its root node analysis and optimized the problem in 4.6 CPU seconds, reducing the computational effort required by the RLT, the RLT-E, and the RRLT+ algorithms by 97, 88, and 45 %, respectively. The J-set constraint filtering-based algorithms further reduced the computational effort for solving Problem PP4. In particular, the J-Hybrid algorithm solved the problem in 2.7 CPU seconds. Noting that this instance has a density of 0.026, RLT-POS automatically selected the J-Hybrid algorithmic option and optimized PP4 within 2.7 CPU seconds. Moreover, when the SDP cut generation step was skipped, RTL-POS experienced numerical difficulties, which were overcome by changing the feasibility tolerance of CPLEX to 10^{-9} from its default value of 10^{-6}. Differing from the results obtained with Problems PP1, PP2, and PP3, the SDP cut routines 1, 3, and 4 performed better than Routine 2, consuming less than a second to solve this instance.

Problem PP5 is a degree-four problem having five variables and three linear equality constraints (see the "Appendix"). The RLT-Hybrid algorithm identified RLT-NB as the underlying relaxation and optimized the problem in 0.4 CPU seconds, reducing the computational effort required by RLT, RLT-E, and RRLT+. CPLEX experienced numerical problems while solving the J-NB relaxations within the branch-and-bound tree, which were overcome by changing the feasibility tolerance of CPLEX to 10^{-9}. Since the density of Problem PP5 is 0.1, RLT-POS automatically selected RLT-Hybrid for its implementation and solved the problem in 0.4 CPU seconds. All SDP cut routines performed similarly and reduced the computational effort by about 50 % as compared to skipping the generation of SDP cuts.

Table 12 CPU time (in seconds) for solving linear equality constrained problems from the literature using RLT-POS, BARON, COUENNE, and SparsePOP

Algorithm	PP1	PP2	PP3	PP4	PP5
RLT-POS	9.6	83	239.6	2.7	0.4
BARON	0.6	1.1	31.0	3.7	0.3
COUENNE	7.3	8.1	27.8	5.7	0.5
SparsePOP	15.2	541.5	Memory Err.	0.5	0.4

The CPU times required to solve the five polynomial programming instances from the literature using BARON, COUENNE, and SparsePOP are displayed in Table 12. BARON and COUENNE solved Problems PP1, PP2, and PP3 using less effort in comparison to RLT-POS, whereas the CPU times consumed by BARON and COUENNE for solving Problems PP4 somewhat exceeded that required by RLT-POS. Note that these test problems are relatively sparse, which favors BARON and COUENNE as observed in the foregoing section. On the other hand, SparsePOP solved Problems PP4 and PP5 quickly, but its computational effort for solving the larger problem instances PP1 and PP2 was significantly higher than that for the other optimization software tested. Moreover, SparsePOP reported an out-of-memory termination while solving Problem PP3. Note that all these problems have convex feasible regions and optimization algorithms that exploit this feature would be advantageous. Currently, RLT-POS does not implement a special algorithm for problems having convex feasible regions (although the related theory exists—see [20]).

5 Conclusions

We have introduced an RLT-based open-source optimization software, RLT-POS, for solving polynomial programming problems. RLT-POS incorporates reduced RLT formulations, constraint filtering techniques, and SDP cuts that were proven to significantly improve the performance of the standard RLT approach when implemented individually. However, their joint implementation requires further considerations and coordination as discussed in this paper. The proposed reduced basis techniques for sparse problems judiciously balances additional bound-factor constraints required for theoretical convergence with equality constraint-based RLT restrictions to devise an effective algorithmic scheme. Additionally, four new cut generation routines were introduced and tested using both randomly generated instances as well as problems from the literature.

The composed software RLT-POS judiciously selects an RLT relaxation based on the problem characteristics such as the density and the degree of the problem, and the number of linear equality constraints. When solving polynomial programming problems without linear equality constraints, RLT-POS reduced the computational effort by about half in comparison to the standard RLT algorithm by appropriately adopting the J-set relaxation. For quadratic and cubic problems having linear equality constraints, RLT-POS is designed to select the J-Hybrid algorithm or the RLT-E relaxation depending on the density of the problem and the number of linear equality constraints relative to the number of variables in the problem. The computational analysis demonstrated that RLT-POS decreased the effort by 57 and 48 %, for quadratic and cubic problems, respectively, in comparison to using the alternative choice of the aforementioned procedures. For higher degree problems having equality constraints, RLT-POS automatically selects between the J-Hybrid and RLT-Hybrid algorithms depending simply on the basis of problem density. For such problems, RLT-POS reduced the CPU time by 28, 33, and 48 % for degree-five, -six, and -seven problems, respectively, whereas the CPU time somewhat increased by 9 % for degree-four problems, in comparison to using the alternative algorithmic choice.

The computational performance of RLT-POS was also compared with the optimization software packages BARON ([29]), COUENNE ([6]), and SparsePOP ([31]). The results demonstrated that RLT-POS was competitive against BARON for solving sparse problems, whereas it reduced the computational effort significantly by 67.8% on average for solving dense problems. RLT-POS performed better than COUENNE for solving all types of test instances and reduced the average CPU time by 67.1%. SparsePOP struggled for solving many instances, even the most sparse ones, and experienced memory problems for relatively dense problems. RLT-POS dominated SparsePOP for solving the randomly generated problems, reducing the average effort by 15-fold.

We also tested RLT-POS for solving instances from the literature. RLT-POS automatically identified the superior implementation among RLT-Hybrid and J-Hybrid for all of the five test problems. The proposed SDP implementation reduced the computational effort between 2-fold and 25-fold for the five problems as compared to skipping the generation of SDP cuts.

The source codes of the present version of RLT-POS as described in this paper is available at https://googledrive.com/host/0B4xVNzR-k0aeU1VMZjQ2WjY3U0k. In further enhancements, RLT-POS can be extended to generate bound-grid-factor constraints [21] for solving polynomial programming problems. RLT-POS could also be extended to solve more general nonlinear programming problems, including factorable programs [27] and mixed-integer nonlinear programming problems [19].

Acknowledgments The authors gratefully acknowledge Nick Sahinidis at Carnegie Mellon University for permitting the use of the BARON solver.

Appendix

PP1 (Problem 119 in [10], where the parameter data is specified in (22) and in Table 13):

$$\begin{aligned} \text{Minimize} \quad & \sum_{i=1}^{16}\sum_{j=1}^{16} a_{ij}\left(x_i^2 + x_i + 1\right)\left(x_j^2 + x_j + 1\right) \\ \text{subject to} \quad & \\ & \sum_{j=1}^{16} b_{ij}x_j - c_i = 0, \quad \forall i = 1, \ldots, 8 \\ & 0 \le x_i \le 5, \quad \forall j = 1, \ldots, 16. \end{aligned}$$

$$\begin{aligned} a_{ij} = 1, \forall (i, j) \in \{&(1, 1), (1, 4), (1, 7), (1, 8), (1, 16), (2, 2), (2, 3), (2, 7), (2, 10), (3, 3), \\ &(3, 7), (3, 9), (3, 10), (3, 14), (4, 4), (4, 7), (4, 11), (4, 15), (5, 5), \\ &(5, 6), (5, 10), (5, 12), (5, 16), (6, 6), (6, 8), (6, 15), (7, 7), (7, 11), \\ &(7, 13), (8, 8), (8, 10), (8, 15), (9, 9), (9, 12), (9, 16), (10, 10), \\ &(10, 14), (11, 11), (11, 13), (12, 12), (12, 14), (13, 13), (13, 14), (14, 14), \\ &(15, 15), (16, 16)\} \end{aligned} \tag{22}$$

Table 13 Parameter data for Problems PP1, PP2, and PP3 (Problem 119 in [10])

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
b_{1j}	0.22	0.2	0.19	0.25	0.15	0.11	0.12	0.13	1	0	0	0	0	0	0	0
b_{2j}	−1.46	0	−1.3	1.82	−1.15	0	0.8	0	0	1	0	0	0	0	0	0
b_{3j}	1.29	−0.89	0	0	−1.16	−0.96	0	−0.49	0	0	1	0	0	0	0	0
b_{4j}	−1.1	−1.06	0.95	−0.54	0	−1.78	−0.41	0	0	0	0	1	0	0	0	0
b_{5j}	0	0	0	−1.43	1.51	0.59	−0.33	−0.43	0	0	0	0	1	0	0	0
b_{6j}	0	−1.72	−0.33	0	1.62	1.24	0.21	−0.26	0	0	0	0	0	1	0	0
b_{7j}	1.12	0	0	0.31	0	0	1.12	0	−0.4	0	0	0	0	0	1	0
b_{8j}	0	0.45	0.26	−1.1	0.58	0	−1.03	0.1	0	0	0	0	0	0	0	1
c_j	2.5	1.1	−3.1	−3.5	1.3	2.1	2.3	−1.5								

PP2 (Problem 119 in [10]—as given by Problem PP1, but with a modified objective function):

$$\text{Minimize} \quad \sum_{i=1}^{16}\sum_{j=1}^{16} a_{ij}\left(x_i^3 + x_i + 1\right)\left(x_j^2 + x_j + 1\right)$$

subject to

$$\sum_{j=1}^{16} b_{ij}x_j - c_i = 0, \quad \forall i = 1, \ldots, 8$$

$$0 \le x_i \le 5, \quad \forall j = 1, \ldots, 16.$$

PP3 (Problem 119 in [10]—as given by Problem PP1, but with a modified objective function):

$$\text{Minimize} \quad \sum_{i=1}^{16}\sum_{j=1}^{16} a_{ij}\left(x_i^3 + x_i^2 + 1\right)\left(x_j^3 + x_j^2 + 1\right)$$

subject to

$$\sum_{j=1}^{16} b_{ij}x_j - c_i = 0, \quad \forall i = 1, \ldots, 8$$

$$0 \le x_i \le 5, \quad \forall j = 1, \ldots, 16.$$

PP4 (Problem 49 in [10]—, along with imposed variable bounds):

$$\text{Minimize} \quad (x_1 - x_2)^2 + (x_3 - 1)^2 + (x_4 - 1)^4 + (x_5 - 1)^6$$

subject to

$$x_1 + x_2 + x_3 + 4x_4 = 7$$

$$x_3 + 5x_5 = 6$$
$$0 \le x_j \le 5, \quad \forall j = 1, \ldots, 5.$$

PP5 (Problem 50 in [10]—, along with imposed variable bounds):

$$\begin{aligned} \text{Minimize} \quad & (x_1 - x_2)^2 + (x_2 - x_3)^2 + (x_3 - x_4)^4 + (x_4 - x_5)^2 \\ \text{subject to} \quad & \\ & x_1 + 2x_2 + 3x_3 = 6 \\ & x_2 + 2x_3 + 3x_4 = 6 \\ & x_3 + 2x_4 + 3x_5 = 6 \\ & 0 \le x_j \le 5, \quad \forall j = 1, \ldots, 5. \end{aligned}$$

References

1. Androulakis, I.P., Maranas, C.D., Floudas, C.A.: αBB: a global optimization method for general constrained nonconvex problems. J. Global Optim. **7**, 337–363 (1995)
2. Anstreicher, K.M.: Semidefinite programming versus the Reformulation-Linearization Technique for nonconvex quadratically constrained quadratic programming. J. Global Optim. **43**(2–3), 471–484 (2009)
3. Anstreicher, K.M.: On convex relaxations for quadratically constrained quadratic programming. Math. Program. **136**(2), 233–251 (2012)
4. Balas, E., Ceria, S., Cornuejols, G.: Mixed 0–1 programming by lift-and-project in a branch-and-cut framework (1996)
5. Bazaraa, M.S., Sherali, H.D., Shetty, C.M.: Nonlinear Programming: Theory and Algorithms, 3rd edn. Wiley, New York (2006)
6. Belotti, P., Lee, J., Liberti, L., Margot, F., Wächter, A.: Branching and bounds tightening techniques for non-convex MINLP. Optim. Methods Softw. **24**(4–5), 597–634 (2009)
7. Cafieri, S., Hansen, P., Létocart, L., Liberti, L., Messine, F.: Compact relaxations for polynomial programming problems. In: Klasing, R. (ed.) Experimental Algorithms, Lecture Notes in Computer Science, vol. 7276, pp. 75–86. Springer, Berlin (2012)
8. Dalkiran, E., Sherali, H.: Theoretical filtering of RLT bound-factor constraints for solving polynomial programming problems to global optimality. J. Global Optim. **57**(4), 1147–1172 (2013)
9. Gill, P.E., Murray, W., Saunders, M.A.: SNOPT: an SQP algorithm for large-scale constrained optimization. SIAM Rev. **47**(1), 99–131 (2005)
10. Hock, W., Schittkowski, K.: Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin Heidelberg, New York (1981)
11. Ibm, ILOG CPLEX Optimization Studio. http://www.ilog.com/products/cplex
12. Lasserre, J.B.: Semidefinite programming vs. LP relaxations for polynomial programming. Math. Operations Res. **27**(2), 347–360 (2002)
13. Lasserre, J.B.: Convergent SDP-relaxations in polynomial optimization with sparsity. SIAM J. Optim. **17**(3), 822–843 (2006)
14. Laurent, M., Rendl, F.: Semidefinite Programming and Integer Programming. In: Aardal, K., Nemhauser, G., Weismantel, R. (eds.) Handbook on Discrete Optimization, pp. 393–514. Elsevier, Amsterdam (2005)
15. Liberti, L.: Linearity embedded in nonconvex programs. J. Global Optim. **33**, 157–196 (2005)
16. Liberti, L., Pantelides, C.C.: An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. J. Global Optim. **36**, 161–189 (2006)
17. MATLAB: version 7.12.0 (R2011a). The MathWorks Inc., Natick, Massachusetts (2011)
18. Ryoo, H.S., Sahinidis, N.V.: A branch-and-reduce approach to global optimization. J. Global Optim. **8**(2), 107–138 (1996)
19. Sherali, H.D., Adams, W.P.: A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems. Kluwer Academic Publishers, Boston (1999)

20. Sherali, H.D., Adams, W.P.: A Reformulation-Linearization Technique (RLT) for semi-infinite and convex programs under mixed 0–1 and general discrete restrictions. Discrete Appl. Math. **157**(6), 1319–1333 (2009)
21. Sherali, H.D., Dalkiran, E.: Combined bound-grid-factor constraints for enhancing RLT relaxations for polynomial programs. J. Global Optim. **51**(3), 377–393 (2011)
22. 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**(2), 483–506 (2012)
23. Sherali, H.D., Dalkiran, E., Liberti, L.: Reduced RLT representations for nonconvex polynomial programs. J. Global Optim. **52**(3), 447–469 (2012)
24. Sherali, H.D., Fraticelli, B.M.P.: Enhancing RLT relaxations via a new class of semidefinite cuts. J. Global Optim. **22**(1–4), 233–261 (2002)
25. Sherali, H.D., Tuncbilek, C.H.: A global optimization algorithm for polynomial programming problems using a Reformulation-Linearization Technique. J. Global Optim. **2**(1), 101–112 (1992)
26. Sherali, H.D., Tuncbilek, C.H.: New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. Operations Res. Lett. **21**(1), 1–9 (1997)
27. Sherali, H.D., Wang, H.: Global optimization of nonconvex factorable programming problems. Math. Program. **89**(3), 459–478 (2001)
28. Sturm, J.F.: Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. Optimiz. Methods Softw. **11**(1–4), 625–653 (1999)
29. Tawarmalani, M., Sahinidis, N.V.: A polyhedral branch-and-cut approach to global optimization. Math. Program. **103**(2), 225–249 (2005)
30. Waki, H., Kim, S., Kojima, M., Muramatsu, M.: Sums of squares and semidefinite program relaxations for polynomial optimization problems with structured sparsity. SIAM J. Optim. **17**(1), 218–242 (2006)
31. 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**(2), 15:1–15:13 (2008)
32. Zorn, K., Sahinidis, N.V.: Global optimization of general nonconvex problems with intermediate polynomial structures. J. Global Optim. **59**(2–3), 673–693 (2014)