Sparse-BSOS: a bounded degree SOS hierarchy for large scale polynomial optimization with sparsity

Tillmann Weisser, Jean-Bernard Lasserre, Kim-Chuan Toh

▶ **To cite this version:**

Tillmann Weisser, Jean-Bernard Lasserre, Kim-Chuan Toh. Sparse-BSOS: a bounded degree SOS hierarchy for large scale polynomial optimization with sparsity. Mathematical Programming Computation, 2018, 10, pp.1–32. ⟨hal-01341931v3⟩

HAL Id: hal-01341931

https://hal.science/hal-01341931v3

Submitted on 26 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

Sparse-BSOS: a bounded degree SOS hierarchy for large scale polynomial optimization with sparsity *

Tillmann Weisser† Jean B. Lasserre‡ and Kim-Chuan Toh§

May 26, 2017

Abstract

We provide a sparse version of the bounded degree SOS hierarchy BSOS [6] for polynomial optimization problems. It permits to treat large scale problems which satisfy a structured sparsity pattern. When the sparsity pattern satisfies the running intersection property this Sparse-BSOS hierarchy of semidefinite programs (with semidefinite constraints of fixed size) converges to the global optimum of the original problem. Moreover, for the class of SOS-convex problems, finite convergence takes place at the first step of the hierarchy, just as in the dense version.

Keywords: Global Optimization, Semidefinite Programming, Sparsity, Large Scale Problems, Convex Relaxations, Positivity Certificates
MSC: 90C26, 90C22

1 Introduction

We consider the polynomial optimization problem:

$$f^* := \min_{\mathbf{x}} \{ f(\mathbf{x}) \ : \ \mathbf{x} \in \mathbf{K} \} \tag{P}$$

where $f \in \mathbb{R}[\mathbf{x}]$ is a polynomial and $\mathbf{K} \subset \mathbb{R}^n$ is the basic semi-algebraic set

$$\mathbf{K} := \{\mathbf{x} \in \mathbb{R}^n : g_j(\mathbf{x}) \geq 0, j = 1, \ldots, m\}, \tag{1}$$

for some polynomials $g_j \in \mathbb{R}[\mathbf{x}]$, $j = 1, \ldots, m$. In [6] Lasserre et al. have provided BSOS, a new hierarchy of semidefinite programs $(\boldsymbol{Q}_d^k)$ indexed by $d \in \mathbb{N}$ and parametrized by $k \in \mathbb{N}$ (fixed), whose associated (monotone non-decreasing) sequence of optimal values $(\rho_d^k)_{d\in\mathbb{N}}$ converges to f^* as $d \to \infty$, i.e., $\rho_d^k \to f^*$ as $d \to \infty$.

One distinguishing feature of the BSOS hierarchy (when compared to the LP-hierarchy defined in [8]) is *finite convergence* for an important class of convex problems. That is, when $f, -g_j$ are SOS-convex polynomials of degree bounded by $2k$, then the first semidefinite relaxation of

*The work of the first author is partially supported by a PGMO grant from *Fondation Mathématique Jacques Hadamard* and a grant from the *ERC council* for the Taming project (ERC-Advanced Grant #666981 TAMING).

†LAAS-CNRS, University of Toulouse, LAAS, 7 Avenue du Colonel Roche, 31031 Toulouse cédex 4, France (tweisser@laas.fr).

‡LAAS-CNRS and Institute of Mathematics, University of Toulouse, LAAS, 7 Avenue du Colonel Roche, 31031 Toulouse cédex 4, France (lasserre@laas.fr).

§Department of Mathematics, National University of Singapore, 10 Lower Kent Ridge Road, Singapore 119076 (mattohkc@nus.edu.sg).

the hierarchy $(\boldsymbol{Q}_d^k)_{d\in\mathbb{N}}$, is exact, i.e., $\rho_1^k = f^*$. (In contrast the LP-hierarchy cannot converges in finitely many steps for such convex problems).

The BSOS hierarchy has also two other important distinguishing features, now when compared to the standard SOS hierarchy defined in [7, 11, 12] (let us call it PUT as it is based on Putinar's Positivstellensatz).

- For each semidefinite relaxation $\boldsymbol{Q}_d^k$, $d \in \mathbb{N}$, the size of the semidefinite constraint is $O(n^k)$, hence fixed and controlled by the parameter k (fixed and chosen by the user). With $k = 0$ one retrieves the LP-hierarchy based on a positivity certificate due to Stengle; see [6] and [8] for more details.

- For the important class of quadratic/quadratically constrained problems, the first relaxation of the BSOS hierarchy is at least as good as the first relaxation of the PUT hierarchy.

In this paper we introduce a sparse version of the BSOS hierarchy to help solve large scale polynomial optimization problems that exhibit some structured sparsity pattern. This hierarchy has exactly the same advantages as BSOS, now when compared with the sparse version of the standard SOS hierarchy introduced in Waki et al. [24].

Motivation

In general the standard SOS hierarchy PUT [7] is considered very efficient (and hard to beat) when it can be implemented. For instance, PUT's first relaxation solves at optimality several small size instances of the Optimal Power Flow problem (OPF), an important problem in management of energy networks (essentially a quadratic/quadratically constrained optimization problem); see e.g. [16]. But even for relatively small size OPFs, PUT's second relaxation cannot be implemented because the size of some matrices required to be positive semidefinite (psd) is too large for state-of-the art semidefinite solvers.

This was an important motivation for introducing the BSOS hierarchy [6]: That is, to provide an alternative to PUT, especially in cases where PUT's second relaxation cannot be implemented. In particular, in *all* relaxations of the BSOS hierarchy (with parameter $k = 1$) the size of the matrix required to be psd is only $O(n)$ (in contrast to $O(n^2)$ for PUT's second relaxation). Thus for quadratic/quadratically constrained problems where PUT's second relaxation cannot be implemented, the second and even higher order relaxations of BSOS can be implemented and provide better lower bounds. [1] When they are strictly less than the optimal value these lower bounds might still be useful as they can be exploited in some other procedure.

The motivation for introducing the sparse version of BSOS is exactly the same as for BSOS, but now for large scale polynomial optimization problems that exhibit some sparsity pattern. For such problems the sparse version of the SOS hierarchy introduced in Waki et al. [24] (that we call "Sparse-PUT" in the sequel) is in general very efficient, in particular when its second relaxation can be implemented. So there is a need to provide an alternative to the latter, especially in cases where its second relaxation cannot be implemented – for instance for some large scale OPF problems (where indeed the second relaxation cannot be implemented, at least in its initial form), e.g as described in [15].

[1] Recently Marandi et al. [14] have shown that BSOS provides better bounds than the first relaxation of PUT for the Pooling Problem - another instance of quadratic/quadratically constrained problems. However their experiments also show that BSOS (i) does not always solve these difficult problems to optimality at a low level "d" relaxation and (ii) does not scale well.

Contribution

Even though in the BSOS hierarchy [6] the size $O(n^k)$ of the semidefinite constraint of $\boldsymbol{Q}_d^k$ is fixed for all d and permits to handle problems (P) of size larger than with the standard SOS-hierarchy, its application is still limited to problems of relatively modest size (say medium size problems). Our contribution is to provide a *sparse* version "Sparse-BSOS" of the BSOS hierarchy which permits to handle large size problems (P) that satisfy some (structured) sparsity pattern. The Sparse-BSOS hierarchy is to its dense version BSOS what the Sparse-PUT hierarchy is to the standard SOS-hierarchy PUT. Again as in the dense case, a distinguishing feature of the Sparse-BSOS hierarchy (and in contrast to Sparse-PUT) is that the size of the resulting semidefinite constraints is *fixed in advance* at the user convenience and does *not* depend on the rank in the hierarchy. However, such an extension is not straightforward because in contrast to Putinar's SOS-based certificate [18] (where the g_j's appear separately), the positivity certificate used in the dense BSOS algorithm [6] potentially mixes all polynomials g_j that define $\mathbf{K}$, that is, if f is positive on $\mathbf{K}$ then

$$f \;=\; \sigma + \sum_{\alpha,\beta\in(\mathbb{N}_0)^m} c_{\alpha\beta} \prod_{j=1}^{m} g_j^{\alpha_j}(1-g_j)^{\beta_j}, \tag{2}$$

for some SOS polynomial σ and positive scalar weights $c_{\alpha\beta}$. Therefore in principle the sparsity as defined in [24] may be destroyed in σ and in the products $\prod_j g_j^{\alpha_j}(1-g_j)^{\beta_j}$. In fact, one contribution of this paper is to provide a specialized sparse version of (2). In particular, we prove that if the sparsity pattern satisfies the so-called *Running Intersection Property* (RIP) then the Sparse-BSOS hierarchy also converges to the global optimum f^* of (P). A sufficient rank-condition also permits to detect finite convergence. Last but not least, we also prove that the Sparse-BSOS hierarchy preserves two distinguishing features of the dense BSOS hierarchy. Namely:

- Finite convergence at the first step of the hierarchy for the class of SOS-convex problems. (Recall that the standard LP hierarchy cannot converge in finitely many steps for such problems [8, 12].)
- For quadratic/quadratically constrained problems (with a sparsity pattern), the first relaxation of the Sparse-BSOS hierarchy is at least as good as that of Sparse-PUT. Hence when Sparse-PUT's second relaxation cannot be implemented the Sparse-BSOS hierarchy (with parameter $k=1$) will provide better lower bounds.

This also suggests that the Sparse-BSOS relaxations could be useful in Branch & Bound algorithms for solving large scale Mixed Integer Non Linear programs (MINLP). Indeed for such algorithms the quality of lower bounds computed at each node of the search tree is crucial for the overall computational efficiency.

The sparsity pattern. Roughly speaking we say that problem (P) satisfies a structured sparsity pattern, if the set $I_0 := \{1,\ldots,n\}$ of all (indices of) variables is some union $\cup_{\ell=1}^{p} I_k$ of smaller blocks of (indices of) variables I_ℓ such that each monomial of the objective function only consists of variables in one of the blocks. In addition, each polynomial g_j in the definition (1) of the feasible set, is also a polynomial only in variables of one of the blocks. Of course the blocks $I_1,\ldots,I_p$ may overlap, i.e., variables may appear in several blocks. Together with the maximum degree appearing in the data of (P), the number and size of the blocks $I_1,\ldots,I_p$ as well as the size of their overlaps, are the characteristics of the sparsity pattern which have the strongest influence on the performance of our algorithm.

Computational experiments. We have tested the Sparse-BSOS hierarchy against both its dense version BSOS and Sparse-PUT [24]. To compare Sparse-BSOS and BSOS we consider problems of small and medium size (≤ 100 variables) and different sparsity patterns. Also we show that Sparse-BSOS can solve sparse large scale problems with up to 3000 variables which by far is out of the scope of the dense version in a reasonable time on a standard lap-top.[2]

To compare Sparse-BSOS and Sparse-PUT we have considered several problems from the literature in non-linear optimization and random medium size non-linear problems. For several such problems the first or second relaxation of the Sparse-PUT hierarchy cannot be solved whereas the first Sparse-BSOS relaxations $d = 1, 2 \ldots$ with a small parameter $k = 1$ or 2 can[3]. Therefore in such cases a good approximation (or even the optimal value) can be obtained by the Sparse-BSOS hierarchy. In our numerical experiments the problems were chosen in such a way that only the first relaxation of Sparse-PUT can be implemented, and indeed for such cases the Sparse-BSOS relaxations provide better lower bounds (and sometimes even the optimal value).

2 Preliminaries

2.1 Notation and definitions

Let $\mathbb{R}[\mathbf{x}]$ be the ring of polynomials in the variables $\mathbf{x} = (x_1, \ldots, x_n)$. Denote by $\mathbb{R}[\mathbf{x}]_d \subset \mathbb{R}[\mathbf{x}]$ the vector space of polynomials of degree at most d, which has dimension $s(d) := \binom{n+d}{d}$, with e.g., the usual canonical basis $(\mathbf{x}^\gamma)_{\gamma \in \mathbb{N}^n_d}$ of monomials, where $\mathbb{N}^n_d := \{\gamma \in (\mathbb{N}_0)^n : |\gamma| \leq d\}$, $\mathbb{N}_0$ is the set of natural numbers including 0 and $|\gamma| := \sum_{i=1}^n \gamma_i$. Also, denote by $\Sigma[\mathbf{x}] \subset \mathbb{R}[\mathbf{x}]$ (resp. $\Sigma[\mathbf{x}]_d \subset \mathbb{R}[\mathbf{x}]_{2d}$) the cone of sums of squares (SOS) polynomials (resp. SOS polynomials of degree at most $2d$). If $f \in \mathbb{R}[\mathbf{x}]_d$, we write $f(\mathbf{x}) = \sum_{\gamma \in \mathbb{N}^n_d} \mathrm{f}_\gamma \mathbf{x}^\gamma$ in the canonical basis and denote by $\mathbf{f} = (\mathrm{f}_\gamma)_\gamma \in \mathbb{R}^{s(d)}$ its vector of coefficients. Finally, let $\mathcal{S}^n$ denote the space of $n \times n$ real symmetric matrices, with inner product $\langle \mathbf{A}, \mathbf{B} \rangle = \operatorname{trace} \mathbf{AB}$. We use the notation $\mathbf{A} \succeq 0$ to denote that $\mathbf{A}$ is positive semidefinite (psd). With $g_0 := 1$, the quadratic module $Q(g_1, \ldots, g_m) \subset \mathbb{R}[\mathbf{x}]$ generated by polynomials $g_1, \ldots, g_m$, is defined by

$$Q(g_1, \ldots, g_m) := \left\{ \sum_{j=0}^m \sigma_j \, g_j \ : \ \sigma_j \in \Sigma[\mathbf{x}] \right\}.$$

With a real sequence $\boldsymbol{y} = (y_\gamma)_{\gamma \in \mathbb{N}^n_d}$, one may associate the linear functional $L_{\boldsymbol{y}} : \mathbb{R}[\mathbf{x}]_d \to \mathbb{R}$ defined by

$$f \left(= \sum_\gamma \mathrm{f}_\gamma \, \mathbf{x}^\gamma \right) \quad \mapsto L_{\boldsymbol{y}}(f) := \sum_\gamma \mathrm{f}_\gamma \, y_\gamma,$$

which is called the Riesz functional. If $d = 2a$ denote by $\mathbf{M}_a(\boldsymbol{y})$ the moment matrix associated with $\boldsymbol{y}$. It is a real symmetric matrix with rows and columns indexed in the basis of monomials $(\mathbf{x}^\gamma)_{\gamma \in \mathbb{N}^n_a}$, and with entries

$$\mathbf{M}_a(\boldsymbol{y})(\alpha, \beta) := L_{\boldsymbol{y}}(\mathbf{x}^{\alpha+\beta}) = y_{\alpha+\beta}, \qquad \forall \, \alpha, \beta \in \mathbb{N}^n_a.$$

If $\boldsymbol{y} = (y_\gamma)_{\gamma \in \mathbb{N}^n}$ is the sequence of moments of some Borel measure μ on $\mathbb{R}^n$ then $\mathbf{M}_a(\boldsymbol{y}) \succeq 0$ for all $a \in \mathbb{N}$. However the converse is not true in general and it is related to the well-known fact

[2]The numerical experiments were run on a standard lap-top from the year 2015. For a more detailed description see page 13.

[3]If in the description (P) some degree "s" is strictly larger than 2 then for Sparse-PUT's first relaxation, the size of the largest positive semidefinite variable is already at least $O(\kappa^{\lceil s/2 \rceil})$ where κ is the size of the largest clique in some graph associated with the problem; if κ is not small it can be prohibitive. In contrast, for all the Sparse-BSOS relaxations (with parameter $k = 1$), this largest size is only $O(\kappa)$.

that there are positive polynomials that are not sums of squares. For more details the interested reader is referred to e.g. [11, Chapter 3].

A polynomial $f \in \mathbb{R}[\mathbf{x}]$ is said to be SOS-convex if its Hessian matrix $\mathbf{x} \mapsto \nabla^2 f(\mathbf{x})$ is an SOS matrix-polynomial, that is, $\nabla^2 f = \boldsymbol{L}\,\boldsymbol{L}^T$ for some real matrix polynomial $\boldsymbol{L} \in \mathbb{R}[\mathbf{x}]^{n\times a}$ (for some integer a). In particular, for SOS-convex polynomials and sequences $\boldsymbol{y}$ with positive semidefinite moment matrix $\mathbf{M}_a(\boldsymbol{y}) \succeq 0$, a Jensen-type inequality is valid:

Lemma 1. *Let $f \in \mathbb{R}[\mathbf{x}]_{2a}$ be SOS-convex and let $\boldsymbol{y} = (y_\gamma)_{\gamma\in\mathbb{N}^n_{2a}}$ be such that $y_0 = 1$ and $\mathbf{M}_a(\boldsymbol{y}) \succeq 0$. Then*

$$L_{\boldsymbol{y}}(f) \;\geq\; f(L_{\boldsymbol{y}}(\mathbf{x})), \quad \textit{with} \quad L_{\boldsymbol{y}}(\mathbf{x}) := (L_{\boldsymbol{y}}(x_1), \ldots, L_{\boldsymbol{y}}(x_n)).$$

For a proof see Theorem 13.21, p. 209 in [12].

2.2 A sparsity pattern

Given $I \subset \{1,\ldots,n\}$ denote by $\mathbb{R}[\mathbf{x}; I]$ the ring of polynomials in the variables $\{x_i : i \in I\}$, which we understand as a subring of $\mathbb{R}[\mathbf{x}]$. Hence, a polynomial $g \in \mathbb{R}[\mathbf{x}; I]$ canonically induces two polynomial functions $g : \mathbb{R}^{\#I} \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}$, where $\#I$ is the number of elements in I.

Assumption 1 (Sparsity pattern)**.** *There exists $p \in \mathbb{N}$ and subsets $I_\ell \subseteq \{1,\ldots,n\}$ and $J_\ell \subseteq \{1,\ldots,m\}$ for all $\ell \in \{1,\ldots,p\}$ such that*

- *$f = \sum_{\ell=1}^p f^\ell$, for some $f^1,\ldots,f^p$ such that $f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]$, $\ell \in \{1,\ldots,p\}$,*
- *$g_j \in \mathbb{R}[\mathbf{x}; I_\ell]$ for all $j \in J_\ell$ and $\ell \in \{1,\ldots,p\}$,*
- *$\bigcup_{\ell=1}^p I_\ell = \{1,\ldots,n\}$,*
- *$\bigcup_{\ell=1}^p J_\ell = \{1,\ldots,m\}$;*
- *for all $\ell = 1,\ldots,p-1$ there is an $s \leq \ell$ such that $\left(I_{\ell+1} \cap \bigcup_{r=1}^{\ell} I_r\right) \subseteq I_s$ (Running Intersection Property).*

On the RIP. The RIP can be understood in the following framework. Suppose that we are given probability measures $(\mu_\ell)_{\ell=1,\ldots,p}$, where each μ_ℓ only *sees* the variables $\{x_i : i \in I_\ell\}$, and the μ_ℓ's are consistent in the sense that if $I_{\ell,k} := I_\ell \cap I_k \neq \emptyset$ then $\mu_{\ell,k} = \mu_{k,\ell}$, where $\mu_{\ell,k}$ (resp. $\mu_{k,\ell}$) is the marginal of μ_ℓ (resp. μ_k) with respect to the variables $\{x_i : i \in I_{\ell,k}\}$. If the RIP property holds then from the μ_ℓ's one is able to build up a probability measure ϕ that sees *all* variables $x_1,\ldots,x_n$, and such that for every $\ell = 1,\ldots,p$, the marginal ϕ_ℓ of ϕ with respect to the variables $\{x_i : i \in I_\ell\}$ is exactly μ_ℓ. Put differently, the "local" information (μ_ℓ) is part of a "consistent and global" information ϕ, without knowing ϕ. In this set up the RIP condition appears naturally when one tries to build up ϕ from the μ_ℓ's; see e.g. the proof in Lasserre [10].

Note that if the sets $I_1,\ldots,I_p$ satisfy all requirements of Assumption 1 except the Running Intersection Property (RIP) it is always possible to enlarge the I_ℓ until the RIP is satisfied. In the worst case however $I_\ell = \{1,\ldots,n\}$ which is not interesting. In [24] no assumption on the sparsity is made. Instead Waki et al. provide an algorithm to create a sparsity pattern satisfying Assumption 1 from any POP. They consider a graph with nodes $\{1,\ldots,n\}$ which has an edge (i,j) if the objective function has a monomial involving x_i and x_j or there is a constraint involving x_i and x_j. If this graph is chordal, the sets I_ℓ correspond to the maximum cliques of this graph (and hence we sometimes call the I_ℓ "cliques"). If the graph is not chordal it is replaced by its chordal extension.

Finally and importantly, even if the RIP property does not hold, the Sparse-PUT or Sparse-BSOS hierarchies can still be implemented and provide valid lower bounds on the global optimum f^*. However convergence of the lower bounds to f^* is not guaranteed any more.

2.3 A preliminary result

For the uninitiated reader we start this section by citing two important Positivstellensätze. For polynomials $g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$ and $\alpha, \beta \in (\mathbb{N}_0)^m$, let $h_{\alpha\beta} \in \mathbb{R}[\mathbf{x}]$ be the polynomial:

$$\mathbf{x} \mapsto \quad h_{\alpha\beta}(\mathbf{x}) := \prod_{j=1}^{m} g_j(\mathbf{x})^{\alpha_j}(1 - g_j(\mathbf{x}))^{\beta_j}, \qquad \mathbf{x} \in \mathbb{R}^n.$$

Lemma 2 (Krivine/Stengle/Vasilescu/Handelmann Positivstellensatz [5, 19, 23, 1])**.** *Let f, $g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$ and $\mathbf{K} = \{\mathbf{x} \in \mathbb{R}^n : 0 \le g_j(\mathbf{x}) \le 1, \quad j = 1, \ldots, m\}$ be compact. If the polynomials 1 and $(g_j)_{j=1\ldots,m}$ generate $\mathbb{R}[\mathbf{x}]$ as an $\mathbb{R}$-algebra and f is (strictly) positive on $\mathbf{K}$, then there exist finitely many positive weights $c_{\alpha\beta}$ such that:*

$$f = \sum_{\alpha,\beta \in (\mathbb{N}_0)^m} c_{\alpha\beta}\, h_{\alpha\beta}.$$

We next provide a sparse version of this Positivstellensatz which to the best of our knowledge is new. Before we recall the sparse version of Putinar's Positivstellensatz [18] proved in [10].

Lemma 3 (Sparse Putinar Positivstellensatz [4])**.** *Let $f, g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$ satisfy Assumption 1 and let the polynomials $1 - M_\ell^{-1} \sum_{i \in I_\ell} x_i^2$ for some positive numbers M_ℓ be among the g_j. If f is (strictly) positive on $\mathbf{K} = \{\mathbf{x} \in \mathbb{R}^n : g_j(\mathbf{x}) \ge 0, j = 1, \ldots, m\}$, then $f = \sum_{\ell=1}^{p} f^\ell$ for some polynomials $f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]$, $\ell = 1, \ldots, p$, and*

$$f^\ell \in \left\{ \sigma_0^\ell + \sum_{j \in J_\ell} \sigma_j^\ell\, g_j \;:\; \sigma_0^\ell, \sigma_j^\ell \in \Sigma[\mathbf{x}, I_\ell] \right\}.$$

From now on, we assume that $\mathbf{K}$ is described by polynomials $g_1, \ldots, g_m$ such that

$$\mathbf{K} = \{\mathbf{x} \in \mathbb{R}^n : 0 \le g_j(\mathbf{x}) \le 1, \quad j = 1, \ldots, m\}. \tag{3}$$

Note that this is not a restriction when $\mathbf{K}$ defined in (P) is compact, as the constraint polynomials can be scaled by a positive factor without adding or losing information.

Let Assumption 1 hold, define $n_\ell := |I_\ell|$, $m_\ell := |J_\ell|$, and let $\mathbf{K}_\ell \subset \mathbb{R}^{n_\ell}$, $\ell = 1, \ldots, p$, be the sets:

$$\mathbf{K}_\ell := \{\mathbf{x} \in \mathbb{R}^{n_\ell} : 0 \le g_j(\mathbf{x}) \le 1, \quad j \in J_\ell\}, \qquad \ell = 1, \ldots, p. \tag{4}$$

Define $\pi_\ell : \mathbb{R}^n \to \mathbb{R}^{n_\ell}, \mathbf{x} \mapsto (x_i)_{i \in I_\ell}$. Then

$$\mathbf{K} = \{\mathbf{x} \in \mathbb{R}^n : \pi_\ell(\mathbf{x}) \in \mathbf{K}_\ell \text{ for all } \ell = 1, \ldots, p\}. \tag{5}$$

Assumption 2. *(a) The sets $\mathbf{K}_\ell$ defined in (4) are compact and the polynomials 1 and $(g_j)_{j \in J_\ell}$ generate $\mathbb{R}[\mathbf{x}; I_\ell]$ as an $\mathbb{R}$-algebra for each $\ell = 1, \ldots, p$.*

(b) For each ℓ there exists $M_\ell > 0$ and $j \in J_\ell$ such that $g_j = 1 - M_\ell^{-1} \sum_{i \in I_\ell} x_i^2$.

Note that if Assumption 2(a) holds then Assumption 2(b) can be satisfied easily. Indeed, since $\mathbf{K}_\ell$ is assumed to be compact, the polynomial $\mathbf{x} \mapsto \sum_{i \in I_\ell} x_i^2$ attains its maximum M_ℓ on $\mathbf{K}_\ell$. Defining $\mathbf{x} \mapsto g_+^\ell(\mathbf{x}) := 1 - M_\ell^{-1} \sum_{i \in I_\ell} x_i^2$ and adding the redundant constraint $0 \le g_+^\ell(\mathbf{x}) \le 1$ to the description of $\mathbf{K}_\ell$ for each $\ell = 1, \ldots, p$, Assumption 2(b) is satisfied.

Let $N^\ell := \{(\alpha, \beta) \in (\mathbb{N}_0)^{2m} : \operatorname{supp}(\alpha) \cup \operatorname{supp}(\beta) \subseteq J_\ell\}$, where $\operatorname{supp}(\alpha) := \{j \in \{1, \ldots, m\} : \alpha_j \neq 0\}$.

Theorem 1 (Sparse Krivine-Stengle Positivstellensatz)**.** *Let $f, g_1, \ldots, g_m \in \mathbb{R}[\mathbf{x}]$ satisfy Assumption 1 and 2(a). If f is (strictly) positive on $\mathbf{K}$ then $f = \sum_{\ell=1}^{p} f^\ell$ for some polynomials $f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]$, $\ell = 1, \ldots, p$, and there exists finitely many positive weights $c^\ell_{\alpha\beta}$ such that*

$$f^\ell = \sum_{(\alpha,\beta)\in N^\ell} c^\ell_{\alpha\beta}\, h_{\alpha\beta}, \qquad \ell = 1, \ldots, p. \tag{6}$$

Note that for each ℓ the representation (6) only involves $g_j \in \mathbb{R}[\mathbf{x}; I_\ell]$ since the corresponding exponents in the definition of $h_{\alpha\beta}$ are 0 if $j \notin J_\ell$.

Proof. As f is positive on $\mathbf{K}$ there exist $\varepsilon > 0$ such that $f - \varepsilon > 0$ on $\mathbf{K}$. As remarked just after Assumption 2, we can add a redundant constraint $0 \le g^\ell_+(\mathbf{x}) \le 1$, with $g^\ell_+ \in \mathbb{R}[\mathbf{x}; I_\ell]$, to the description of each of the $\mathbf{K}_\ell$ so as to satisfy Assumption 2(b). Hence we can apply Lemma 3 to obtain the representation

$$f - \varepsilon = \sum_{\ell=1}^{p} \left(\underbrace{\sigma^\ell_0 + \sigma^\ell_+ g^\ell_+ + \sum_{j\in J_\ell} \sigma^\ell_j\, g_j}_{\in \mathbb{R}[\mathbf{x}; I_\ell] \text{ and } \ge 0 \text{ on } \mathbf{K}_\ell} \right),$$

for some SOS polynomials $\sigma^\ell_j, \sigma^\ell_+$. Next, let

$$f^\ell := \varepsilon/p + \sigma^\ell_0 + \sigma^\ell_+ g^\ell_+ + \sum_{j\in J_\ell} \sigma^\ell_j\, g_j, \qquad \ell = 1, \ldots, p.$$

Notice that $f = \sum_{\ell=1}^{p} f^\ell$ and each $f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]$ is strictly positive on $\mathbf{K}_\ell$, $\ell = 1, \ldots, p$. Removing the redundant constraints $0 \le g^\ell_+ \le 1$ from the description of $\mathbf{K}_\ell$ again does not change the fact that f^ℓ is strictly positive on $\mathbf{K}_\ell$. Furthermore Assumption 2(a) still holds. Hence, we can apply Lemma 2 to each of the f^ℓ, which yields (6) for each $\ell = 1, \ldots, p$.

□

3 Main result

3.1 The Sparse Bounded-Degree SOS-hierarchy (Sparse-BSOS)

Consider problem (P) and let Assumption 1 & 2 hold. For $d \in \mathbb{N}$ let $N^\ell_d := \{(\alpha, \beta) \in N^\ell : \sum_j (\alpha_j + \beta_j) \le d\}$, which is of size $\binom{2m_\ell + d}{d}$. Let $k \in \mathbb{N}$ be fixed and define

$$d_{\max} := \max\{\deg(f), 2k, d \max_j \{\deg(g_j)\}\}.$$

We consider the family of optimization problems indexed by $d \in \mathbb{N}$:

$$\begin{aligned} q^k_d := \sup_{\substack{t, \boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^p, \\ f^1, \ldots, f^p}} \Big\{ t \;:\; & f^\ell - \sum_{(\alpha,\beta)\in N^\ell_d} \lambda^\ell_{\alpha\beta} h_{\alpha\beta} \in \Sigma[\mathbf{x}; I_\ell]_k, \quad \ell = 1, \ldots, p, \\ & f - t = \sum_{\ell=1}^{p} f^\ell,\ \boldsymbol{\lambda}^\ell \in \mathbb{R}^{|N^\ell_d|}_+,\ t \in \mathbb{R},\ f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]_{d_{\max}} \Big\}, \end{aligned} \tag{7}$$

where the scalars $\lambda^\ell_{\alpha\beta}$ are the entries of the vector $\boldsymbol{\lambda}^\ell \in \mathbb{R}^{|N^\ell_d|}_+$.

Observe that when k is fixed, then for each $d \in \mathbb{N}$, computing q_d^k in (7) reduces to solving a semidefinite program and $q_{d+1}^k \geq q_d^k$ for all $d \in \mathbb{N}$. Therefore (7) defines a *hierarchy* of semidefinite programs of which the associated sequence of optimal values is monotone non decreasing. Let us call it the Sparse-BSOS hierarchy, as it is the *sparse version* of the BSOS hierarchy [6].

For practical implementation of (7) there are at least two possibilities depending on how the polynomial identities in (7) are treated in the resulting semidefinite program. To state that two polynomials $p, q \in \mathbb{R}[\mathbf{x}]_d$ are identical one can either:

- *equate their coefficients* (e.g. in the monomial basis, i.e., $\mathrm{p}_\gamma = \mathrm{q}_\gamma$ for all $\gamma \in \mathbb{N}^n_d$), or

- *equate their values* (i.e an implementation by *sampling*) on $\binom{n+d}{d}$ generic points (e.g. randomly generated on the box $[-1,1]^n$).

Both strategies have drawbacks: To equate coefficients one has to take powers of the polynomials g_j and $(1-g_j)$ which leads to an ill-conditioning of the coefficients of the polynomials $h_{\alpha\beta}$ (in the monomial basis) as some of them are multiplied by binomial coefficients which become large quickly when the relaxation order d increases. On the other hand, when equating values the resulting linear system may become ill-conditioned because (depending on the points of evaluation and the g_j) the constraints may be nearly linear dependent. The authors of [6] chose point evaluation for the implementation of BSOS because the SDP solver SDPT3[4] is able to exploit the structure of the SDP generated in that way and hence problems with psd variables of larger size can be solved. However, this feature cannot be used in our case and so in Sparse-BSOS we have implemented the equality constraints of (7) by comparing coefficients.

Indeed, equating coefficients is reasonable in the present context because we expect the number of variables n_ℓ in each block to be rather small. The drawback of this choice is that the resulting relaxations with high order d can become time consuming (and even ill-conditioned as explained above).

Define $\mathcal{I}_\ell^d := \{\gamma \in \mathbb{N}^n_d \,:\, \mathrm{supp}(\gamma) \in I_\ell\}$, $\Gamma^d := \{\gamma \,:\, \gamma \in \mathcal{I}_\ell^d, \text{for some } \ell \in \{1,\ldots,p\}\}$ and let $\mathbf{0}$ be the all zero vector of size n. Then for k fixed and for each d, we get

$$
\begin{aligned}
q_d^k = \sup_{\substack{t, Q^1,\ldots,Q^p,\\ \boldsymbol{\lambda}^1,\ldots,\boldsymbol{\lambda}^p,\\ \mathrm{f}^1,\ldots,\mathrm{f}^p}} \Big\{ t \quad \text{s.t.} \quad & \sum_{\ell:\, \gamma \in \mathcal{I}_\ell^{d_{\max}}} \mathrm{f}_\gamma^\ell = \mathrm{f}_\gamma, \quad \forall \gamma \in \Gamma^{d_{\max}} \backslash \{\mathbf{0}\}, \quad \sum_{\ell=1}^p \mathrm{f}_{\mathbf{0}}^\ell = \mathrm{f}_{\mathbf{0}} - t \\
& \mathrm{f}_\gamma^\ell - \sum_{(\alpha,\beta)\in N_d^\ell} \lambda_{\alpha\beta}^\ell \, (h_{\alpha\beta})_\gamma - \langle Q^\ell, \left(\boldsymbol{v}_k^\ell (\boldsymbol{v}_k^\ell)^T\right)\rangle_\gamma = 0, \qquad (8) \\
& \qquad\qquad \forall \gamma \in \mathcal{I}_\ell^{d_{\max}}, \quad \ell = 1,\ldots,p, \\
& Q^\ell \in \mathcal{S}_+^{s(\ell,k)},\ \boldsymbol{\lambda}^\ell \in \mathbb{R}_+^{|N_d^l|},\ \mathbf{f}^\ell \in \mathbb{R}^{s(\ell,d_{\max})},\ \ell = 1,\ldots,p,\ t \in \mathbb{R} \Big\},
\end{aligned}
$$

where $s(\ell,k) := \binom{n_\ell+k}{k}$, and $\boldsymbol{v}_k^\ell$ is the vector of the canonical (monomial) basis of the vector space $\mathbb{R}[\mathbf{x}; I_\ell]_k$. Here we use the convention that the coefficient q_γ of a polynomial q is 0 if $|\gamma| > \deg(q)$. For a matrix polynomial $\mathbf{q} = (q_{ij})_{1\leq i,j\leq s} \in \mathbb{R}[\mathbf{x}]^{s\times s}$ the coefficient $\mathbf{q}_\gamma$ is the matrix $((q_{ij})_\gamma)_{1\leq i,j\leq s} \in \mathbb{R}^{s\times s}$. Note that the semidefinite matrix variables have fixed size $s(\ell,k)$, independent of $d \in \mathbb{N}$. This is a crucial feature for computational efficiency of the approach.

The dual of the semidefinite program (8) reads:

$$
\begin{aligned}
\tilde{q}_d^k := \inf_{\boldsymbol{\theta}^1,\ldots,\boldsymbol{\theta}^p, \boldsymbol{y}} \Big\{ L_{\boldsymbol{y}}(f) \quad \text{s.t. } & y_\gamma = \theta_\gamma^\ell, \quad \forall \ell : \gamma \in \mathcal{I}_\ell^{d_{\max}}; \quad \forall \gamma \in \Gamma^{d_{\max}}, \quad y_0 = 1, \\
& \mathbf{M}_k(\boldsymbol{\theta}^\ell) \succeq 0, \quad L_{\boldsymbol{\theta}^\ell}(h_{\alpha\beta}) \geq 0, \quad (\alpha,\beta) \in N_d^\ell; \quad \ell = 1,\ldots,p \qquad (9) \\
& \boldsymbol{\theta}^\ell \in \mathbb{R}^{s(\ell,d_{\max})},\ \ell = 1,\ldots,p, \quad \boldsymbol{y} \in \mathbb{R}^{|\Gamma^d|} \Big\}.
\end{aligned}
$$

[4] In both [6] and this paper SDPT3 [21, 22] is used as SDP solver.

By standard weak duality of convex optimization, $\tilde{q}_d^k \geq q_d^k$ for all $d \in \mathbb{N}$. Moreover (9) is a relaxation of (P). Indeed, if $\hat{\mathbf{x}}$ is feasible in (P), define $\theta_\gamma^\ell := \hat{\mathbf{x}}^\gamma$ for all $\gamma \in \mathcal{I}_\ell^{d_{\max}}$ and all $\ell \in \{1,\ldots,p\}$. Define $\boldsymbol{y}$ according to the constraints in (9). Then $y_0 = \hat{\mathbf{x}}^0 = 1$. Let v_k^ℓ be the vector of the canonical (monomial) basis of the vector space $\mathbb{R}[\mathbf{x}; I_\ell]_k$, understood as a function $\mathbb{R}^n \to \mathbb{R}^{s(\ell,k)}$. Then $\mathbf{M}_k(\boldsymbol{\theta}^\ell) = v_k^\ell(\hat{\mathbf{x}})(v_k^\ell(\hat{\mathbf{x}}))^T \succeq 0$. Finally, regarding the Riesz functionals $L_{\boldsymbol{\theta}^\ell}(h_{\alpha\beta}) = h_{\alpha\beta}(\hat{x}) \geq 0$ for all $(\alpha,\beta) \in N_d^\ell$, $\ell = 1,\ldots,p$ and $L_{\boldsymbol{y}}(f) = f(\hat{\mathbf{x}})$. Thus to every feasible point $\hat{\mathbf{x}}$ of (P) corresponds a feasible point of (9) giving the same value $f(\hat{\mathbf{x}})$ and therefore $f^* \geq \tilde{q}_d^k \geq q_d^k$ for all $d \in \mathbb{N}$. In fact we have an even more precise and interesting result:

Theorem 2 ([9])**.** *Consider problem* (P) *and let Assumption 1 & 2 hold. Let $k \in \mathbb{N}$ be fixed. Then the sequence $(q_d^k)_{d\in\mathbb{N}}$, defined in* (7) *is monotone non-decreasing and $q_d^k \to f^*$ as $d \to \infty$.*

Proof. Monotonicity of the sequence $(q_d^k)_{d\in\mathbb{N}}$ follows from its definition. Let $\varepsilon > 0$ be fixed arbitrary. Then the polynomial $f - f^* + \varepsilon$ is positive on $\mathbf{K}$. By Theorem 1 there exist some polynomials $f^1 \ldots, f^p$ such that (6) holds, i.e.,

$$f - \underbrace{(f^* - \varepsilon)}_{t} = \sum_{\ell=1}^{p} f^\ell \quad \text{with} \quad f^\ell = \sum_{(\alpha,\beta)\in N^\ell} c_{\alpha\beta}^\ell \, h_{\alpha\beta}, \quad \ell = 1,\ldots,p,$$

for finitely many positive weights $c_{\alpha\beta}^\ell$. Hence $(f^* - \varepsilon, f^\ell, c_{\alpha\beta}^\ell)$ is a feasible solution for (7) as soon as d is sufficiently large, and therefore $q_d^k \geq f^* - \varepsilon$. Combining this with $q_d^k \leq f^*$ and noting that $\varepsilon > 0$ was arbitrary, yield the desired result $q_d^k \to f^*$ as $d \to \infty$. □

Note that we optimize over all possible representations of f of the form $f = \sum_{\ell=1}^p f^\ell$ with $f^\ell \in \mathbb{R}[\mathbf{x}; I_\ell]$. By Assumption 1 such a representation exists; however it does not need to be unique.

We next show that a distinguishing feature of the dense BSOS hierarchy [6] is also valid for its sparse version Sparse-BSOS.

Theorem 3. *Assume the feasible set $\mathbf{K}$ in problem* (P) *is non-empty and let Assumption 1 & 2 hold. Let $k \in \mathbb{N}$ be fixed and assume that for every $\ell = 1,\ldots,p$, the polynomials f^ℓ and $-g_j$ are all SOS-convex polynomials of degree at most $2k$. (If $k > 1$ we assume (with no loss of generality) that for each $\ell = 1,\ldots,p$, and some sufficiently large $\kappa_\ell > 0$, the redundant (SOS-convex) constraints $1 - \kappa_\ell^{-1} \sum_{i\in I_\ell} x_i^{2k} \geq 0$, $\ell = 1,\ldots,p$, are present in the description (3) of $\mathbf{K}$.)*

Then the semidefinite program (9) *has an optimal solution $((\boldsymbol{\theta}^{*\ell}), \boldsymbol{y}^*)$ such that $f^* = \tilde{q}_1^k = L_{\boldsymbol{y}^*}(f)$ and $\mathbf{x}^* := (L_{\boldsymbol{y}^*}(x_1),\ldots,L_{\boldsymbol{y}^*}(x_n)) \in \mathbf{K}$ is an optimal solution of (P). Hence finite convergence takes place at the first step of the hierarchy.*

Proof. Let $d = 1$ (so that $d_{\max} = 2k$) and consider the semidefinite program (9). Note, that $\boldsymbol{\theta}^\ell = (\theta_\gamma^\ell)_{\gamma\in\mathcal{I}_\ell^{2k}}$. Recall that by Assumption 2, for every $\ell = 1,\ldots,p$, there exists $j \in J_\ell$ such that $g_j(\mathbf{x}) = 1 - M_\ell^{-1} \sum_{i\in I_\ell} x_i^2$. In addition if $k > 1$ then there also exists r such that $g_r(\mathbf{x}) = 1 - \kappa_\ell^{-1} \sum_{i\in I_\ell} x_i^{2k}$. Hence, feasibility in (9) (with an appropriate choice of $(\alpha,\beta) \in N_d^\ell$) implies that $L_{\boldsymbol{\theta}^\ell}(g_j) \geq 0$ and $L_{\boldsymbol{\theta}^\ell}(g_r) \geq 0$, which in turn imply:

$$L_{\boldsymbol{\theta}}(x_i^2) \leq M_\ell \, \theta_0^\ell \, (= M_\ell) \quad \text{and} \quad L_{\boldsymbol{\theta}}(x_i^{2k}) \leq \kappa_\ell \, \theta_0^\ell \, (= \kappa_\ell) \quad (\text{if } k > 1), \quad \forall i \in I_\ell, \quad \forall \ell = 1,\ldots,p,$$

where we have used that $\theta_0^\ell = y_0 = 1$ for all $\ell = 1,\ldots,p$.

Combining this with $\mathbf{M}_k(\boldsymbol{\theta}^\ell) \succeq 0$ and invoking Proposition 2.38 in [12] yields that $|\theta_\gamma^\ell| \leq \max[M_\ell, \kappa_\ell, 1]$ for every $|\gamma| \leq 2k$ and $\ell = 1,\ldots,p$. Consequently, the set of feasible solutions

$(\boldsymbol{\theta}^\ell, \boldsymbol{y})$ of (9) is bounded, hence compact. This implies that (9) has an optimal solution $(\boldsymbol{\theta}^{*\ell}, \boldsymbol{y}^*)$. Notice that among the constraints $L_{\boldsymbol{\theta}^{*\ell}}(h_{\alpha\beta}) \geq 0$ are the constraints $L_{\boldsymbol{\theta}^{*\ell}}(g_j) \geq 0$ for all $j \in J_\ell$. As f^ℓ and $-g_j$ are SOS-convex, invoking Lemma 1 yields

$$f^\ell(\mathbf{x}_\ell^*) \leq L_{\theta^{*\ell}}(f^\ell) \quad \text{and} \quad 0 \leq L_{\boldsymbol{\theta}^{*\ell}}(g_j) \leq g_j(\mathbf{x}_\ell^*), \quad \forall j \in J_\ell, \quad \ell = 1, \ldots, p,$$

where $\mathbf{x}_\ell^* := (L_{\boldsymbol{\theta}^{*\ell}}(x_i))_{i \in I_\ell} \in \mathbf{K}_\ell$, $\ell = 1, \ldots, p$. In addition, the constraint $y_\gamma^* = \theta_\gamma^{*\ell}$, for all ℓ such that $\gamma \in \mathcal{I}_\ell^{d_{\max}}$, implies that $(\mathbf{x}_\ell^*)_i = (\mathbf{x}_{\ell'}^*)_i$ whenever $i \in I_\ell \cap I_{\ell'}$. Therefore defining $x_i^* := (\mathbf{x}_\ell^*)_i$ whenever $i \in I_\ell$, one obtains $g_j(\mathbf{x}^*) \geq 0$ for all j, i.e., $\mathbf{x}^* \in \mathbf{K}$. Finally,

$$f^* \geq \tilde{q}_1^k = L_{\boldsymbol{y}^*}(f) = \sum_{\ell=1}^p L_{\boldsymbol{\theta}^{*\ell}}(f^\ell) \geq \sum_{\ell=1}^p f^\ell(\mathbf{x}_\ell^*) = f(\mathbf{x}^*),$$

which shows that $\mathbf{x}^* \in \mathbf{K}$ is an optimal solution of (P). Hence $f^* = f(\mathbf{x}^*) = \tilde{q}_1^k$. □

3.2 Sufficient condition for finite convergence

By looking at the dual (9) of the semidefinite program (8) one obtains a sufficient condition for finite convergence. Choose $\omega \in \mathbb{N}$ minimal such that $2\omega \geq \max\{\deg(f), \deg(g_1), \ldots, \deg(g_m)\}$. We have the following lemma.

Lemma 4. *Let* $(\boldsymbol{\theta}^{*1}, \ldots, \boldsymbol{\theta}^{*p}, \boldsymbol{y}^*) \in \mathbb{R}^{s(1,d_{max})} \times \cdots \times \mathbb{R}^{s(p,d_{max})} \times \mathbb{R}^s$ *be an optimal solution of* (9). *If* $\operatorname{rank} \mathbf{M}_\omega(\boldsymbol{\theta}^{*\ell}) = 1$ *for every* $\ell = 1, \ldots, p$, *then* $\tilde{q}_d^k = f^*$ *and* $\mathbf{x}^* = (y_{\boldsymbol{\gamma}}^*)_{|\gamma|=1}$ *is an optimal solution of problem* (P).

Proof. If $\operatorname{rank} \mathbf{M}_\omega(\boldsymbol{\theta}^{*\ell}) = 1$, then $(\theta_\gamma^{*\ell})_{|\gamma| \leq 2\omega}$, is the vector of moments (up to order 2ω) of the Dirac measure $\delta_{\mathbf{x}^\ell}$ at the point $\mathbf{x}^\ell := (\theta_\gamma^{*\ell})_{|\gamma|=1} \in \mathbb{R}^{n_\ell}$. Note that from the constraints $y_\gamma = \theta_\gamma^\ell$ in (9)

$$x_\gamma^{\ell_1} = x_\gamma^{\ell_2}, \quad \forall \ell_1, \ell_2 : \gamma \in \mathcal{I}_{\ell_1}^1 \cap \mathcal{I}_{\ell_2}^1.$$

Hence we can define $x_\gamma^* := x_\gamma^\ell$ for all γ such that $|\gamma| = 1$ and independent of the specific choice of ℓ such that $\gamma \in \mathcal{I}_\ell$. For the same reason $\mathbf{x}^* = (y_{\boldsymbol{\gamma}}^*)_{|\gamma|=1}$. Consequently, for all $q \in \mathbb{R}[\mathbf{x}; I_\ell]_{2\omega}$

$$q(\mathbf{x}^*) = q(\mathbf{x}^\ell) = \int q \, \delta_{\mathbf{x}^\ell} = L_{\boldsymbol{\theta}^{*\ell}}(q) = L_{\boldsymbol{y}^*}(q)$$

Let $j \in J_\ell$. The constraints $L_{\boldsymbol{\theta}^{*\ell}}(h_{\alpha\beta}) \geq 0$ imply in particular $L_{\boldsymbol{\theta}^{*\ell}}(g_j) \geq 0$. Since $\deg(g_j) \leq 2\omega$, $0 \leq L_{\boldsymbol{\theta}^{*\ell}}(g_j) = g_j(\mathbf{x}^\ell) = g_j(\mathbf{x}^*)$, and so as $j \in J_\ell$ was arbitrary, $\mathbf{x}^* \in \mathbf{K}$. Finally, and again because $\deg(f) \leq 2\omega$,

$$f^* \geq \tilde{q}_d^k = L_{\boldsymbol{y}^*}(f) = f(\mathbf{x}^*),$$

from which we may conclude that $\mathbf{x}^* \in \mathbf{K}$ is an optimal solution of problem (P). □

4 Computational issues

4.1 Comparing coefficients

As already outlined earlier we have implemented polynomial equality constraints in (7) by comparison of coefficients. The resulting constraints in the SDP are sparse and can be treated efficiently by the SDP solver. A crucial issue for the implementation of the Sparse-BSOS relaxations is how to equate the coefficients. The bottleneck for such an implementation is that one has to gather all occurrences of the same monomials.

As in [6], we use the following data format for representing a polynomial f in n variables:

$$\texttt{F(i,1:n+1)} = [\gamma^T, \mathrm{f}_\gamma],$$

stating that f_γ is the ith coefficient of f corresponding to the monomial $\mathbf{x}^\gamma$. Adding two polynomials is done by concatenating their representations. Hence, equating the coefficients of $\mathbf{x}^\gamma$ is basically finding all indices i of a polynomial F, such that $\texttt{F(i,1:n)} = \gamma^T$.

Matlab is providing the function `ismember(A,B,'rows')` to find a row `A` in a Matrix `B`. This however is too slow for our purpose. Instead of using this function, we reduce the problem to finding all equal entries of a vector, which can be handled much more efficiently. To that end we multiply `F(:,1:n)` by a random vector. Generically this results in a vector whose entries are different if and only if the corresponding rows in `F(:,1:n)` are different.

4.2 Reducing problem size

By looking at (8) more closely one may reduce the number of free variables and the number of constraints. It is likely that there are some indices $i \in \{1, \dots, n\}$, that only appear in one of the I_ℓ, say $i \in I_{\ell_i}$. Hence, for all $\gamma \in \bigcup_{j=1}^p \mathcal{I}_j^{d_{\max}}$ such that $\gamma_i \neq 0$ the second equality constraint in (8) reduces to $\mathrm{f}_\gamma^{\ell_i} = f_\gamma$. Consequently, there is a number of variables that are or can be fixed from the beginning. We do this in our implementation. However, in order to be able to certify optimality by Lemma 4, one needs to trace back these substitutions, to recover the moment sequences $\boldsymbol{y}^\ell$ from the solution of the dual problem. Removing these fixed variables occasionally leads to equality constraints $0 = 0$ in the SDP. We remove those constraints for better conditioning.

5 Numerical experiments

In this section we provide some examples to illustrate the performance of our approach and to compare with others. It is natural to compare Sparse-BSOS with its dense counterpart BSOS [6] whenever possible. We also compare Sparse-BSOS with the sparse standard SOS-hierarchy [24] (Sparse-PUT). We use the following notation to refer to both the different SDP relaxations of Problem (P) and to the particular implementations used in our experiments. Note that for the sparse versions, we suppose that Assumption 1 holds.

- BSOS: The implementation of the dense version of BSOS as described in [6].

$$\sup_{t,Q,\boldsymbol{\lambda}} \Big\{ t \quad \text{s.t. } f(x^{(\tau)}) - \sum_{(\alpha,\beta)\in\mathbb{N}_d^{2m}} \lambda_{\alpha\beta}\, h_{\alpha\beta}(x^{(\tau)}) - \langle Q, \Big(v_k(x^{(\tau)})(v_k(x^{(\tau)}))^T\Big)\rangle = 0,$$
$$\tau = 1, \dots, |\mathbb{N}^n_{d_{\max}}|, \quad Q \in \mathcal{S}_+^{|\mathbb{N}_k^n|},\ \boldsymbol{\lambda} \in \mathbb{R}_+^{|\mathbb{N}_d^{2m}|},\ t \in \mathbb{R} \Big\},$$

  where d is the relaxation order, k is a parameter fixed in advance to control the size of the psd variable Q, v_k is the vector of the canonical (monomial) basis of the vector space $\mathbb{R}[\mathbf{x}]_k$ understood as a function $\mathbb{R}^n \to \mathbb{R}^{|\mathbb{N}_k^n|}$, and $\{x^{(\tau)} : \tau = 1, \dots, |\mathbb{N}^n_{d_{\max}}|\}$ is a set of generic points in $[-1, 1]^n$. We emphasize that BSOS uses sampling to implement the equality constraints in the above problem (see discussion §3.1). We do the same implementation as is used for the numerical experiments in [6]. As explained in [6, Section 3], some of the constraints in the SDP stated above might by nearly redundant, i.e. they might be "almost" linearly dependant of others. Such constraints are removed before handing the problem over to the SDP solver. A close look to the code also reveals that the maximum

number of point evaluations considered is limited to approximately 5000. This choice has been made to prevent the solver from running out of memory. As a consequence, when $|\mathbb{N}^{2m}_d| > 5000$ the implemented SDP is a relaxation of the relaxation stated above.

- Sparse-BSOS: Our implementation of the sparse version of BSOS as described in (8).

  Again d is the relaxation order, k is a parameter fixed in advance to control the size of the psd variables Q^ℓ. We use the technique described in the previous section to reduce the size of the SDP before handing it over to the SDP solver. However we do not remove any information from the SDP.

- Sparse-PUT: Our implementation of the sparse version of the standard SOS-hierarchy [24].

$$\begin{aligned}\sup_{\substack{t,\boldsymbol{Q}^1,\ldots,\boldsymbol{Q}^p,\\ \mathrm{f}^1,\ldots,\mathrm{f}^p}} \Big\{ t \quad \text{s.t.} \quad & \sum_{\ell:\,\gamma\in\mathcal{I}_\ell^{d_{\max}}} \mathrm{f}^\ell_\gamma = \mathrm{f}_\gamma, \quad \forall \gamma \in \Gamma^{2d}\backslash\{\mathbf{0}\}, \quad \sum_{\ell=1}^p \mathrm{f}^\ell_{\mathbf{0}} = \mathrm{f}_{\mathbf{0}} - t \\ & \mathrm{f}^\ell_\gamma - \langle Q^\ell_0, \left(\boldsymbol{v}^\ell_d(\boldsymbol{v}^\ell_d)^T\right)\rangle_\gamma - \sum_{j\in J_\ell} \langle Q^\ell_j, \left(\boldsymbol{v}^\ell_{d_j}(\boldsymbol{v}^\ell_{d_j})^T\right)\rangle_\gamma = 0, \\ & \qquad\qquad \forall \gamma \in \mathcal{I}^{2d}_\ell, \quad \ell = 1,\ldots,p, \\ & Q^\ell_0 \in \mathcal{S}^{s(\ell,d)}, Q^\ell_j \in \mathcal{S}^{s(\ell,d_j)}_+, j \in J_\ell,\ \ell = 1,\ldots,p, \\ & \mathbf{f}^\ell \in \mathbb{R}^{s(\ell,d_{\max})},\ \ell = 1,\ldots,p,\ t \in \mathbb{R}\Big\},\end{aligned}$$

  where d is the relaxation order and d_j is the largest integer less or equal to $\frac{2d-\deg(g_j)}{2}$. We use the same technique to reduce the size of the SDP as for Sparse-BSOS.

The aim of the experiments is twofold. On the one hand we compare Sparse-BSOS with BSOS on small and medium size problems (as BSOS cannot handle large scale problems) with different sparsity patterns. In particular we are interested in the following issues:

- the influence of the block sizes (depending on the sparsity pattern) when the size of overlaps between blocks of variables is fixed.
- the influence of various block and overlap sizes for a fixed number of variables ($n = 90$).
- does the finite convergence of the dense version occur systematically earlier than for the sparse version? (As it cannot occur later.)

On the other hand we compare Sparse-BSOS with Sparse-PUT on high degree small size and lower degree medium and large scale problems. This comparison requires some care because the feasible set for Sparse-PUT is $\mathbf{K} = \{\mathbf{x} : g_j(\mathbf{x}) \geq 0\}$ while for Sparse-BSOS (and BSOS) it $\mathbf{K} = \{\mathbf{x} : 0 \leq g_j(\mathbf{x}) \leq 1\}$. Hence we code the information about the feasible set in the constraints $0 \leq g_j(\mathbf{x})$ and scale the constraint polynomials g_j to be less than 1 on the feasible set. In general one expects that if Sparse-PUT gives a good result, Sparse-BSOS will not do better. However we have identified at least three scenarii where Sparse-BSOS can beat Sparse-PUT (and does it at least in some examples).

- The first possible Sparse-PUT relaxation[5] yields the optimal value of the polynomial optimization problem, and some degrees d_j of the SOS weights are potentially greater than 0. This happens, when the degree of the objective function is larger than the degree of

[5]In Sparse-PUT the maximum size of the psd variables is $\binom{n^*+d}{n^*} \times \binom{n^*+d}{d}$ where $n^* = \max_\ell |I_\ell|$ and $2d \geq \max[\deg(f), \deg(g_j)]$.

some constraint plus 2. Then setting the parameter $k = \deg(f)/2$, the first Sparse-BSOS relaxations ($d = 1, 2, \ldots$) are faster than Sparse-PUT and may also reach the optimal value; this is illustrated in Tables 5 & 6.

- The first possible Sparse-PUT relaxation does not reach the optimal value of the POP and the second relaxation cannot be solved (because its size is too large and/or is too costly to implement). If the SOS weights in the first Sparse-PUT relaxation are all of degree 0, then again setting the parameter $k = \deg(f)/2$, the first Sparse-BSOS relaxation gives the same result and it is possible to obtain better bounds by going higher in the relaxation order. In particular, this is the case for the important class of quadratic/quadratically constrained programs (that is when $\max[\deg(f), \deg(g_j)] \leq 2$); this is illustrated in Table 10.

- The first possible Sparse-PUT relaxation cannot be solved. Then setting the parameter $k < \deg(f)/2$, the first Sparse-BSOS relaxations ($d = 1, 2, \ldots$) may be solvable and so provide lower bounds on the optimal value of the polynomial optimization problem whereas Sparse-PUT cannot; this is illustrated in Table 11.

To summarize the above cases, in Sparse-BSOS we take full advantage of the facts that (a) the constraints enter the certificate only with non-negative weights in contrast to SOS weights in Sparse-PUT, (b) the size of the psd variables is fixed and does not increase with the relaxation order. In particular, while the minimal size of the largest psd variable in Sparse-PUT is determined by the polynomial data, in Sparse-BSOS one can always set the parameter k to 1 which implies that the maximum size of the semidefinite matrices in the SDP (8) is always at most $O(n^*)$ where $n^* = \max_\ell n_\ell$, for all d. This is because by assumption, the g_j generate the algebra $\mathbb{R}[\mathbf{x}]$ and so a polynomial of arbitrary degree and positive on $\mathbf{K}$, can be obtained as a positive linear combination of the $g_j^{\alpha_j}(1 - g_j)^{\beta_j}$ (with no SOS involved). So even if $\max[\deg(f), \max_j \deg(g_j)] > 2$, the optimal value of (7) (with $k = 1$) is finite as soon as d is large enough, and so provides a non trivial lower bound.

The results on numerical experiments described in the next sections are biased by the (limited) sample of examples that we have considered. Therefore they should be understood as partial indications rather than definite conclusions. The latter would require much more computational experiments beyond the scope of the present paper.

All experiments were performed on an Intel Core i7-5600U CPU @ 2.60GHz × 4 with 16GB RAM. Scripts are executed in Matlab 8.5 (R2015b) 64bit on Ubuntu 14.04 LTS operating system. The SDP solver used is SDPT3-4.0[21, 22].

The results are presented in tables below. They provide the following information:

- A pattern or problem code to identify the example.

- The relaxation order d and the chosen parameter k for the psd constraints.

- The maximal degree $d_{\max}$, appearing in the certificate.

- The numbers of non-negative variables (corresponding to $\lambda_{\alpha\beta}$), unrestricted (free) variables (corresponding to t and the coefficients of the f^ℓ), and the number (and size) of the positive semidefinite variables (corresponding to the sums of squares).

- The number of (equality) constraints in the SDP.

- The (primal) solution of the SDP.

- The time in seconds, including the times to generate and solve the SDP as well as computing the optimality condition.
- The abbreviation rk stands for the rank of the moment matrices according to Section 3.2. In the case of Sparse-BSOS and Sparse-PUT, rk is the average rank of all moment matrices and can hence be a decimal number. When reporting an integer the rank is acutally integer, i.e. if we write rk= 1, the rank is actually 1, if however we write 1.0 the rank is strictly bigger than 1.
- If the primal solution is written in bold, it was certified by the rank condition and coincides[6] with the global optimum of the POP.
- Primal solutions were marked with * when the solver stopped because *steps were too short*, the *maximum number of iterations* was achieved, or *lack of progress*. In these cases one has to consider the result carefully.

5.1 BSOS vs. Sparse-BSOS

5.1.1 Dense small size examples

Table 1: We compare the sparse and the dense version of BSOS on a set of examples introduced in [6]. In the problem description the first number of the name indicates the number of variables, the second the degree of the problem. The examples are relatively small size, i.e. $n \leq 20$. The degree of the objective function and the constraints is between 2 and 8. As the test sample is from the dense version, no sparsity pattern is present and we pass the information $I = \{1, \ldots, n\}$ and $J = \{1, \ldots, m\}$ to Sparse-BSOS. Consequently both hierarchies compute the same certificate. The only difference comes from the implementation of the equality constraints and the different handling of the psd variable in SDPT3 (see discussion p. 3.1).

We see that Sparse-BSOS is able to solve the same problems as BSOS. As the problems are dense, Sparse-BSOS uses a trivial sparsity pattern and both certificates are the same. Consequently the optimal value coincides unless one of the SDPs stopped because of numerical issues. In most cases Sparse-BSOS is faster than BSOS.

5.1.2 Sparse quadratic examples (medium and large scale)

For the remaining examples in §5 we consider sparsity patterns having some banded structure. The patterns are described by a vector $\mathbf{n} \in \mathbb{N}^p$ and a natural number $\mathbf{o}$. The vector $\mathbf{n}$ determines the size of the blocks I_ℓ whereas $\mathbf{o}$ defines the number of overlapping variables between two consecutive blocks. More formally defining $c_1 := n_1$ and $c_\ell := c_{\ell-1} + n_\ell - \mathbf{o}$ we construct

$$I_\ell := \{c_\ell - n_\ell + 1, \ldots, c_\ell\}.$$

Note that the total number of variables in pattern I is c_p. We call those sparsity pattern banded, because the RIP (*see* Asumption 1) is satisfied by

$$\left(I_{\ell+1} \cap \bigcup_{r=1}^{\ell} I_r\right) \subseteq I_\ell.$$

Informally for $\mathbf{n}$ we use notations like (7×5) instead of $(5, 5, 5, 5, 5, 5, 5)$ or $(2 \times 17, 13)$ instead of $(17, 17, 13)$ without any misunderstanding.

[6] In this context we consider two numbers to be equal if there difference is less than 10^{-8}.

			BSOS			Sparse-BSOS		
problem	(d,k)	$d_{\max}$	solution	rk	time	solution	rk	time
P4_2	(1,1)	2	**-5.7491e-01**	1	0.8s	**-5.7491e-01**	1	1.6s
P4_4	(1,2)	4	-6.5919e-01	7	0.3s	-6.5919e-01	3	0.4s
	(2,2)	8	**-4.3603e-01**	1	0.7s	**-4.3603e-01**	1	0.5s
P4_6	(1,3)	6	-6.2500e-02*	27	1.0s	-6.2500e-02	15	0.5s
	(2,3)	12	-6.0937e-02	7	0.7s	-6.0937e-02*	6	0.6s
	(3,3)	18	-6.0693e-02	4	2.6s	-6.0693e-02*	4	4.7s
P4_8	(1,4)	8	-9.3381e-02*	39	9.2s	-9.3355e-02	15	1.7s
	(2,4)	16	-8.5813e-02*	9	3.0s	-8.5813e-02	4	1.3s
	(3,4)	24	-8.5813e-02	4	4.3s	-8.5814e-02*	4	4.1s
P6_2	(1,1)	2	**-5.7491e-01**	1	0.2s	**-5.7491e-01**	1	0.3s
P6_4	(1,2)	4	-5.7716e-01	13	0.7s	-5.7716e-01	4	0.4s
	(2,2)	8	-5.7696e-01	4	4.4s	-5.7696e-01	3	0.7s
	(3,2)	12	-5.7696e-01	3	25.0s	-5.7765e-01*	3	16.6s
P6_6	(1,3)	6	-6.5972e-01*	35	6.6s	-6.5972e-01	7	2.7s
	(2,3)	12	-6.5972e-01*	32	21.5s	-6.5972e-01	4	4.4s
	(3,3)	18	**-4.1288e-01***	1	44.5s	**-4.1288e-01***	1	82.1s
P8_2	(1,1)	2	**-5.7491e-01**	1	0.2s	**-5.7491e-01**	1	0.3s
P8_4	(1,2)	4	-6.5946e-01	21	1.5s	-6.5946e-01	5	0.8s
	(2,2)	8	**-4.3603e-01***	1	17.7s	**-4.3603e-01***	1	2.7s
P10_2	(1,1)	2	**-5.7491e-01**	1	0.3s	**-5.7491e-01**	1	0.3s
P10_4	(1,2)	4	-6.5951e-01	31	5.5s	-6.5951e-01	6	2.2s
	(2,2)	8	**-4.3603e-01***	1	23.4s	**-4.3603e-01***	1	8.4s
P20_2	(1,1)	4	**-5.7492e-01***	1	0.8s	**-5.7491e-01**	1	0.4s

Table 1: Comparison BSOS vs. Sparse-BSOS on non sparse examples

$\mathbf{o}/n$		# n-neg. var.	# free var.	# psd var. (size)	# constr.	solution	time
40/60	BSOS	125	1	1(61)	1891	-1.1123e+01	13.8s
	Sparse-BSOS	206	1723	2(51)	3513	-1.1123e+01	14.0s
30/70	BSOS	145	1	1(71)	2556	-1.2753e+01	24.3s
	Sparse-BSOS	206	993	2(51)	3148	-1.2753e+01	11.2s
20/80	BSOS	165	1	1(81)	3321	-1.3376e+01	48.1s
	Sparse-BSOS	206	463	2(51)	2883	-1.3376e+01	10.5s
10/90	BSOS	185	1	1(91)	4186	-1.5406e+01	73.6s
	Sparse-BSOS	206	133	2(51)	2718	-1.5406e+01	9.3s
5/95	BSOS	195	1	1(96)	4656	-1.5665e+01	89.5s
	Sparse-BSOS	206	43	2(51)	2673	-1.5665e+01	9.2s
1/99	BSOS	203	1	1(100)	5050	-1.5658e+01 *	152.2s
	Sparse-BSOS	206	7	2(51)	2655	-1.5658e+01	10.8s

Table 2: QPI $\mathbf{n} = (50, 50)$, quadratic constraints: $s = 2$, maximal degree of the certificate $d_{\max} = 2$, time to compute the first relaxation $d = 1$ with $k = 1$

We also analyze the impact of different sparsity pattern on instances of the following sparse quadratic optimization problem. Given a sparsity pattern $I = \{I_1, \dots, I_p\}$ we consider

$$\min_{\mathbf{x}} \left\{ x^T A x + b^T x \ : \ 1 - \sum_{i \in I_\ell} x_i^s \geq 0 \quad \ell = 1, \dots, p, \quad x_i \geq 0 \quad i = 1, \dots, n, \right\}, \tag{QP}$$

where b is a random vector and the symmetric matrix A is randomly generated according to I[7]. We verify that A has positive and negative eigenvalues to make sure, that our problem is non-convex. Depending on the choice of $s \in \{1, 2\}$ we call the constraints *linear* or *quadratic*, although in the latter case we still have the linear constraints $x_i \geq 0$. Note that the second constraint implies that the first constraint is at most 1 and vice versa.[8]

Table 2: To compare the respective SDPs arising from BSOS and Sparse-BSOS we fix $\mathbf{n} = (2 \times 50)$ and create different sparsity patterns by varying $\mathbf{o}$. From these patterns we generate instances of (QP) with $s = 2$. Choosing parameter $k = 1$ for the size of the sum of squares we compute and solve the first relaxation $d = 1$ of BSOS and Sparse-BSOS. As $\mathbf{n}$ is fixed for all examples the number of variables n grows when the overlap $\mathbf{o}$ decreases.

Both BSOS and Sparse-BSOS are able to solve all instances of this problem and provide the same lower bounds. In contrast to the previous example the certificates and the corresponding SDP handed over to the solver are different: Consider the example with $\mathbf{o} = 40$ and $n = 60$ variables. As $k = 1$ the psd variable in BSOS is of size $\binom{n+k}{k} = 61$ and grows with the number of variables. As the sparsity pattern in all examples consists of two blocks of 50 variables, Sparse-BSOS always has 2 psd variables of size $\binom{n_\ell+k}{k} = 51$, independently of the total number of

[7]By this we means that a random value between -1 and 1 is assigned to an entry a_{ij} of A if and only if both i and j are contained in the same I_ℓ for some ℓ. Otherwise $a_{ij} = 0$. The values of b are randomly generated between -1 and 1, too.

[8]Also note that Assumption 2 is fulfilled only when $s = 2$. In the case of $s = 1$ with the same arguments as in the first part of the proof of Theorem 3, one can show that in this case with $d \geq 2$, the feasible set of the SDP (9) is compact and an optimal solution is attained. Furthermore, since the set $\mathbf{K}$ is a polyhedron, the quadratic modules associated to the K_ℓ are archimedean. One can adapt the proofs of Theorem 1 and 2 so that the convergence result is still true in the case $s = 1$.

	n	# n-neg. var.	# free var.	# psd var. (size)	# constr.	time
BSOS	none	20706	1	1(91)	4186	882.4s
Sparse-BSOS	(90)	20706	2	1(91)	4187	49.0s
	$(50, 42)$	11001	13	1(51)/1(43)	2278	10.5s
	$(50, 26, 18)$	9072	24	1(51)/1(27)/1(19)	1905	7.8s
	$(50, 2 \times 18, 10)$	8439	35	1(51)/2(19)/1(11)	1788	6.7s
	$(50, 5 \times 10)$	7821	57	1(51)/5(11)	1682	6.6s
	$(2 \times 34, 26)$	7776	24	2(35)/1(27)	1649	5.3s
	$(3 \times 26, 18)$	6171	35	3(27)/1(19)	1340	3.2s
	$(34, 3 \times 18, 10)$	5862	46	1(35)/3(19)/1(11)	1287	3.4s
	$(2 \times 26, 2 \times 18, 10)$	5538	46	2(27)/2(19)/1(11)	1223	2.8s
	$(5 \times 18, 10)$	4581	57	5(19)/1(11)	1042	1.7s
	(11×10)	3036	112	11(11)	777	0.8s

Table 3: QPII $n = 90$, overlap 2, linear constraints: $s = 1$, $(d, k) = (2, 1)$, maximal degree of the certificate $d_{\max} = 2$, same optimal solution verified by rank one condition in all cases

variables. The unrestricted variable in BSOS corresponds to the optimizing variable t. Sparse-BSOS also has this optimizing variable. The other unrestricted variables correspond to the non-fixed coefficients of the polynomials f^ℓ. This is easy to see in the case of $\mathbf{o} = 1$: The maximal degree of the certificate is $d_{\max} = 2$. Hence, the non-fixed coefficients are the coefficients of the monomials $1, x_{50}$ and x_{50}^2. Consequently after removing the unrestricted variables for the fix coefficients, we have 3 unrestricted variables for f^1 and 3 for f^2. Together with the optimizing variable, we end up with 7 unrestricted variables as presented in the table. The non-negative variables in Table 2 correspond to the $\lambda_{\alpha\beta}$ in the description of BSOS and Sparse-BSOS. Note that the number of constraints for each block is $m_1 = m_2 = 51$ for Sparse-BSOS and $m = n + 2$ for BSOS (the linear constraints plus the two quadratic constraints). Consequently there are $206 = \binom{2m_1+d}{d} + \binom{2m_2+d}{d}$ non-negative variables for Sparse-BSOS and $\binom{2(n+2)+d}{d}$ non-negative variables for BSOS depending on the number of variables n. The number of constraints in BSOS corresponds to the number of point evaluations needed to guarantee the equality constraint in the BSOS formulation, i.e $|\mathbb{N}^n_{d_{\max}}| = \binom{n+d_{\max}}{d_{\max}}$, and hence increases with n. For Sparse-BSOS three equalities have to be considered. As they are implemented by comparing coefficients we expect $|\mathcal{I}_1^{d_{\max}}| + |\mathcal{I}_2^{d_{\max}}| + |\Gamma_{d_{\max}}| = 2 \times \binom{50+2}{2} + (2 \times \binom{50+2}{2} - \binom{\mathbf{o}+2}{2})$ many constraints. The difference comes from the fact that with every removed unrestricted variable, we also remove a constraint.

Summarizing, when the overlap $\mathbf{o}$ decreases Sparse-BSOS benefits from having less unrestricted variables and constraints while the size of the psd variables remains the same; the SDP becomes easier. In contrast to this in BSOS the size of the psd variable and the number of non-negative variables and constraints increases; the SDP becomes harder. The solving time reported in the last column of the table reflects this nearly perfectly.

Table 3: We create the sparsity pattern I with $\mathbf{n} = (11 \times 10)$ and $\mathbf{o} = 2$ and consider one instance of Problem (QP) with $s = 1$. According to the footnote in Section 5.1.2 we can guarantee the existence of a dual solution of Sparse-BSOS by choosing $d \geq 2$. Again, we choose parameter $k = 1$ and compute the second BSOS and Sparse-BSOS relaxation $d = 2$. For Sparse-BSOS we use different sparsity patterns and observe the effect on the SDP and the solving time.

The dense and the sparse hierarchy are able to solve this sparse problem and certify optimal-

n	**o**	n	# n-neg.var.	# unrest.var.	# psd var.(size)	# const	rnk	time
100x4	1	301	6 600	497	100(5)	1 699	1	1.8s
400x4	1	1201	26 400	1 997	400(5)	6 799	1.03	11.8s
700x4	1	2102	46 200	3 497	700(5)	11 899	1.02	28.2s
1000x4	1	3001	66 000	4 997	1000(5)	16 999	1.03	49.1s
100x5	2	302	9 100	1 091	100(6)	2 596	1.06	2.4s
400x5	2	1202	36 400	4 391	400(6)	10 396	1.04	16.2s
700x5	2	2102	63 700	7 691	700(6)	18 196	1.10	35.1s
1000x5	2	3002	91 000	10 991	1000(6)	25 996	1.08	74.8s
50x8	2	302	9 500	541	50(9)	2 496	1	4.3s
200x8	2	1202	38 000	2 191	200(9)	9 996	1.08	14.9s
350x8	2	2102	66 500	3 841	350(9)	17 496	1.01	35.3s
500x8	2	3002	95 000	5 491	500(9)	24 996	1.02	68.7s
50x9	3	303	11 550	933	50(10)	3 192	1.08	3.2s
200x9	3	1203	46 200	3 783	200(10)	12 792	1.07	18.2s
350x9	3	2103	80 850	6 633	350(10)	22 392	1.06	44.6s
500x9	3	3003	115 500	9 483	500(10)	31 992	1.04	133.4s

Table 4: QPLS, linear constraints: $s = 1$, $(d, k) = (2, 1)$, maximal degree of the certificate $d_{\max} = 2$

ity by the rank one condition. We do not repeat the discussion on the number of variables and constraints in this second example. We only remark that the additional unrestricted variable and constraints in in Sparse-BSOS for $\mathbf{n} = (90)$ compared to the BSOS case without sparsity pattern comes from the equality $\mathrm{f}_{\mathbf{0}}^1 = \mathrm{f}_{\mathbf{0}} - t$. Because t is variable $\mathrm{f}_{\mathbf{0}}^1$ is not fixed and hence cannot be removed like all the other coefficients in this case (cf.§4.2).

The reason for the big difference of computing times between BSOS and Sparse-BSOS is double. On the one hand side, searching for linear dependent constraints in BSOS takes a lot of time. Generating the SDP with BSOS took over 70 seconds whereas the SDP for Sparse-BSOS was generated in less than 5 seconds. The main reason however is hidden in the constraints. Indeed the constraints in Sparse-BSOS are sparse whereas the constraints in BSOS are dense and therefore the SDP solver is much slower in the dense case.[9] With regard to the computing times for Sparse-BSOS one can see that the size of the biggest psd variable is a more important factor than the number of non-negative and free variables or the number of constraints.

Table 4: We next use the quadratic problem (QP) a third time to investigate the range of Sparse-BSOS on large scale examples. In this sample we generate sparsity patterns with 400 to 1000 blocks of size 3 to 9 and small overlap between 1 and 3. As in the previous example we chose $s = 1$ and $d = 2$. The size of those examples is by far too large to run BSOS and so we only display the results obtained by Sparse-BSOS. They show that Sparse-BSOS is able to compute lower bounds for sparse large scale problems in reasonable time.

Summarizing the computational results of this section, we saw that Sparse-BSOS is competitive with the dense version on dense examples. On sparse examples the Sparse-BSOS outperforms BSOS as it can use the additional information. The advantage becomes bigger, when the block size n_ℓ is small with respect to the total number of variables n. In addition, the number of

[9]Regarding this example the implementation of equalities using sampling might look questionable. Its positive effect comes into play when considering larger psd variables. In [6] the authors were able to handle problems with psd variables of size up to 861 ($n = 40$, $k = 2$) due to the special handling of the constraints associated to the psd variable in the case of sampling. This is by far out of the range of what can be done by comparing coefficients.

variables in the intersection of at least two blocks I_ℓ influences the performance of Sparse-BSOS. Although the certificates depend on the information, known about the sparsity pattern, we did not encounter that the value computed by BSOS or by Sparse-BSOS with a coarse sparsity pattern, was better than the one computed with the actual pattern.

5.2 Sparse-BSOS vs. Sparse-PUT

As already mentioned, the sparse version Sparse-PUT [24] of the standard hierarchy of SOS relaxations has been proved to be efficient in solving several large scale problems; see for instance its successful application to some Optimal Power Flow problems [15, 3]. However there are a number of cases where only the first relaxation of Sparse-PUT can be implemented because the second one is too costly to implement. Also if $t := \deg(f) > 2$ then the first SDP relaxation is already very expensive (or cannot be implemented) because some moment matrices of size $\binom{n^*+t}{t} \times \binom{n^*+t}{t}$ are constrained to be psd (where $n^* := \max_\ell n_\ell$).

5.2.1 Test problems from the literature

In this section we present experiments on some test problems considered to be challenging in non-linear optimization. All test functions are sums of squares and share the global minimum 0. Hence, it would be possible to compute the minimum in the unconstrained case. However, if not using constraints both Sparse-BSOS and Sparse-PUT reduce to searching for sums of squares. Hence, we restrict the problems to the set

$$\mathbf{K} = \{x \in \mathbb{R}^n \ : \ 1 - \sum_{i \in I_\ell} x_i \geq 0, \quad \ell = 1, \ldots, p; \quad x_i \geq 0, \quad i = 1, \ldots, n\},$$

which depends on the sparsity pattern of the specific function. Consequently, the optimal values of the considered functions are strictly greater than zero, when the minimizer of the unconstrained problem is not in $\mathbf{K}$. We consider the following test functions of degree 4:

- The *Chained Wood Function*:

$$f := \sum_{j \in H} \Big(100(x_{j+1} - x_j^2)^2 + (1 - x_j)^2 + 90(x_{j+3} - x_{j+2}^2)^2 \\ +(1 - x_{j+2})^2 + 10(x_{j+1} + x_{j+3} - 2)^2 + 0.1(x_{j+1} - x_{j+3})^2\Big)$$

  where $H := \{2i - 1 \ : \ i = 1, \ldots, n/2 - 1\}$ and $n \equiv 0 \mod 4$. The sparsity pattern is given by $\mathbf{n} = (p \times 4)$ and $\mathbf{o} = 2$.

- The *Chained Singular Function*:

$$f := \sum_{j \in H} \Big((x_j + 10x_{j+1})^2 + 5(x_{j+2} - x_{j+3})^2 + (x_{j+1} - 2x_{j+2})^4 + 10(x_j - x_{j+3})^4\Big)$$

  where $H := \{2i - 1 \ : \ i = 1, \ldots, n/2 - 1\}$ and $n \equiv 0 \mod 4$. The sparsity pattern is given by $\mathbf{n} = (p \times 4)$ and $\mathbf{o} = 2$.

- The *Generalized Rosenbrock Function*:

$$f := \sum_{i=2}^{n} \Big(100(x_i - x_{i-1}^2)^2 + (1 - x_i)^2\Big).$$

  The sparsity pattern is given by $\mathbf{n} = (p \times 2)$ and $\mathbf{o} = 1$.

ChainedWood	rel.	Sparse-BSOS solution	rk	time	Sparse-PUT solution	rk	time
$n = 500$	$d = 1$	**3.8394e+03***	1	16.7s	inf	-	-
	$d = 2$	**3.8394e+03**	1	10.4s	**3.8394e+03**	1	16.7s
$n = 600$	$d = 1$	**4.6104e+03***	1	20.6s	inf	-	-
	$d = 2$	**4.6104e+03**	1	13.2s	**4.6104e+03**	1	21.0s
$n = 700$	$d = 1$	**5.3813e+03***	1	24.1s	inf	-	-
	$d = 2$	**5.3813e+03**	1	15.8s	**5.3813e+03**	1	26.0s
$n = 800$	$d = 1$	**6.1523e+03***	1	27.3s	inf	-	-
	$d = 2$	**6.1523e+03**	1	19.4s	**6.1523e+03**	1	31.1s
$n = 900$	$d = 1$	**6.9232e+03***	1	30.8s	inf	-	-
	$d = 2$	**6.9232e+03**	1	22.3s	**6.9232e+03**	1	36.5s
$n = 1000$	$d = 1$	7.6942e+03*	3	28.6s	inf	-	-
	$d = 2$	**7.6942e+03**	1	26.1s	**7.6942e+03**	1	42.3s

Table 5: Comparison Sparse-BSOS ($k = 2$) and Sparse-PUT on the Chained Wood Function

ChainedSingular	rel.	Sparse-BSOS solution	rk	time	Sparse-PUT solution	rk	time
$n = 500$	$d = 1$	-1.4485e-02*	1.0	19.6s	inf	-	-
	$d = 2$	**-9.7833e-10**	1	17.8s	**-2.0271e-10**	1	22.6s
$n = 600$	$d = 1$	-2.7372e-03*	1.0	40.1s	inf	-	-
	$d = 2$	**-1.2640e-09**	1	21.4s	**-1.9613e-10**	1	27.8s
$n = 700$	$d = 1$	-1.7548e-03*	1.0	41.6s	inf	-	-
	$d = 2$	**-1.7613e-09**	1	25.3s	**-2.4628e-10**	1	34.1s
$n = 800$	$d = 1$	-1.9438e-03*	1.0	58.9s	inf	-	-
	$d = 2$	**2.1935e-09**	1	29.0s	**-2.3398e-10**	1	41.0s
$n = 900$	$d = 1$	-1.8924e-02*	1.0	43.5s	inf	-	-
	$d = 2$	**-2.6072e-09**	1	33.5s	**-3.5871e-10**	1	47.3s
$n = 1000$	$d = 1$	-4.4914e-02*	1.0	35.5s	inf	-	-
	$d = 2$	**-9.3508e-10**	1	39.5s	**-1.7329e-10**	1	54.9s

Table 6: Comparison Sparse-BSOS ($k = 2$) and Sparse-PUT on the Chained Singular Function

Table 5 & 6: We solve the Chained Wood and the Chained Singular Function for $n = 500, \ldots, 1000$ with Sparse-BSOS and Sparse-PUT. For Sparse-BSOS we fix $k = 2$ and compute the first and the second relaxation. For Sparse-PUT the relaxation $d = 1$ is infeasible, as the degree of the certificate is at most 2 but the functions are of degree 4. Hence the first feasible relaxation for Sparse-PUT is $d = 2$. When reading the tables note that the reported rank is the average of the rank of several matrices and hence is not necessarily an integer.

For $d = 2$ both Sparse-BSOS and Sparse-PUT are able to find and certify the optimal value (up to numerical errors) for both functions. In these examples Sparse-PUT is slower than Sparse-BSOS because for every constraint Sparse-PUT introduces a psd variable of size 5×5 corresponding to a sum of square of degree 2. Sparse-BSOS only introduces a non-negative variable for all products and squares of constraints. As the number of non-negative variables is not too big, Sparse-BSOS beats Sparse-PUT.

At this point we noticed a rather strange phenomenon. For the first Sparse-BSOS relaxation $d = 1$, the SDP solver runs into numerical problems. Yet, from the definition of the Chained Functions we know that they are sum of squares of degree 4, which in principle Sparse-BSOS

GeneralizedRosenbrock	rel.	Sparse-BSOS solution	rk	time	Sparse-PUT solution	rk	time
$n = 100$	$d = 1$	4.8496e+01	2.0	2.8s	inf	-	-
	$d = 2$	9.6145e+01	2.2	1.7s	**9.6197e+01**	1	2.4s
	$d = 3$	9.6184e+01	2.1	4.5s	-	-	-
	$d = 4$	9.6195e+01*	1.3	18.2s	-	-	-
$n = 200$	$d = 1$	9.7496e+01	2.0	2.7s	inf	-	-
	$d = 2$	1.9512e+02	2.1	3.2s	**1.9519e+02**	1	4.6s
	$d = 3$	1.9516e+02	2.1	5.9s	-	-	-
	$d = 4$	1.9395e+02*	3	565.6s	-	-	-
$n = 300$	$d = 1$	1.4650e+02	2.0	3.9s	inf	-	-
	$d = 2$	2.9410e+02	2.1	4.8s	**2.9418e+02**	1	6.9s
	$d = 3$	2.9414e+02	2.0	9.3s	-	-	-
	$d = 4$	2.9176e+02*	3	695.6s	-	-	-
$n = 400$	$d = 1$	1.9550e+02	2.0	5.2s	inf	-	-
	$d = 2$	3.9308e+02	2.1	6.5s	**3.9317e+02**	1	9.4s
	$d = 3$	3.9312e+02*	2.0	27.6s	-	-	-
	$d = 4$	-8.1403e+05*	3	801.9s	-	-	-
$n = 500$	$d = 1$	2.4450e+02	2.0	6.8s	inf	-	-
	$d = 2$	4.9206e+02	2.0	8.3s	**4.9216e+02**	1	12.4s
	$d = 3$	4.9210e+02*	2.0	31.8s	-	-	-
	$d = 4$	4.9215e+02*	3	1144.6s	-	-	-
$n = 600$	$d = 1$	2.9350e+02	2.0	8.1s	inf	-	-
	$d = 2$	5.9104e+02	2.0	10.4s	**5.9115e+02**	1	15.5s
	$d = 3$	5.9108e+02*	2.0	22.3s	-	-	-
	$d = 4$	5.9114e+02*	1.0	111.6s	-	-	-

Table 7: Comparison Sparse-BSOS ($k = 2$) and Sparse-PUT on the Generalized Rosenbrock Function

is able to represent with $k = 2$ for any d. One possible explanation is that the solver may be confused by the additional non-negative variables. However, we could reproduce the same behaviour when omitting the constraints and explicitly only searching for a sum of squares. This phenomena is not specific to our implementation or to SDPT3. It also occurs when searching for a sum of squares representation of the Chained Wood Function ($n = 4$) with Gloptipoly3 [2] using SeDuMi1.3[20] and with Yalmip[13] using Mosek7[17].

Table 7: Solving the Generalized Rosenbrock Function for $n = 100, \ldots, 600$ with Sparse-BSOS and Sparse-PUT. As in the previous examples for Sparse-BSOS we fix $k = 2$ and compute the first and the second relaxation. Again for Sparse-PUT the relaxation $d = 1$ is infeasible. Hence we start the Sparse-PUT relaxation with $d = 2$.

As in the previous examples we find a unique minimizer with Sparse-PUT, certified by the rank condition. This time Sparse-BSOS is not able to find the optimum even when going up to the relaxation $d = 4$. However, even though Sparse-BSOS does not obtain the optimal value at an early relaxation, its optimal value at step $d = 2$ is already in the right order of magnitude and can be computed faster than the optimal value provided by Sparse-PUT.

Note that for $n = 100$ the relaxation $d = 1$ is slower than the relaxation $d = 2$. The same happens in Table 9 for some values of n. This is an issue related to our configuration and could not be reproduced when using another SDP solver or another operating system, respectively.

	Sparse-BSOS				Sparse-PUT			
DiscreteBoundary	rel.	solution	rk	time	rel.	solution	rk	time
$n = 15$	$d = 1$	**9.8705e-04**	1	1.4s	$d = 3$	**9.8705e-04**	1	2.0s
$n = 20$	$d = 1$	**4.4893e-04**	1	1.8s	$d = 3$	**4.4893e-04**	1	2.6s
$n = 25$	$d = 1$	**2.4060e-04**	1	2.3s	$d = 3$	**2.4060e-04**	1	3.3s
$n = 30$	$d = 1$	1.4358e-04	2.1	2.7s	$d = 1$	inf	-	-
	$d = 2$	1.4359e-04	1.1	3.2s	$d = 2$	inf	-	-
	$d = 3$	**1.4358e-04**	1	4.0s	$d = 3$	**1.4359e-04**	1	3.9s
$n = 35$	$d = 1$	9.2438e-05	4	3.9s	$d = 1$	inf	-	-
	$d = 2$	9.2438e-05*	3.8	4.3s	$d = 2$	inf	-	-
	$d = 3$	**9.2439e-05**	1	4.8s	$d = 3$	**9.2441e-05**	1	4.5s

Table 8: Comparison Sparse-BSOS ($k = 3$) and Sparse-PUT on the Discrete Boundary Value Function

To close this section we consider the following test functions of degree 6:

- The *Discrete Boundary Value Function*:

$$f := \sum_{i=1}^{n} (2x_i - x_{i-1} - x_{i+1} + \frac{1}{2}h^2(x_i + ih + 1)^3)^2,$$

  where $h := \frac{1}{n+1}$,$x_0 := 9 =: x_{n+1}$. The sparsity pattern is $\mathbf{n} = (p \times 3)$ and $\mathbf{o} = 2$.

- The *Broyden Banded Function*:

$$f := \sum_{i=1}^{n} \left(x_i(2 + 10x_i^2) + 1 - \sum_{j \in H_i} (1 + x_j)x_j \right)^2,$$

  where $H_i := \{j : j \neq i, \max(1, i - 5) \leq j \leq \min(n, i + 1)\}$. The sparsity pattern is $\mathbf{n} = (p \times 7)$ and $\mathbf{o} = 6$.

Table 8: Solving the Discrete Boundary Value Function for $n = 15, \ldots, 35$. The first possible relaxation for Sparse-PUT is $d = 3$. For Sparse-BSOS we choose $k = 3$ and compute the first relaxations until we get the certified optimal value.

Note that Sparse-BSOS and Sparse-PUT certify different optimal values in the case $n = 30$ and $n = 35$ and that in contrast to the theory, the series of lower bounds computed by Sparse-BSOS for $n = 35$ is not monotonously increasing. The difference however is less than 10^{-8} and can be considered to be zero "numerically". As for the Chained Functions in Table 5 & 6 both Sparse-BSOS and Sparse-PUT are able to certify the minimum in all cases. When Sparse-BSOS succeeds to do so at an early step of the relaxation it is faster and if not then it takes approximately the same time.

Table 9: Solving the Broyden Banded Function for $n = 7, \ldots, 15$. The first possible relaxation for Sparse-PUT is $d = 3$. For Sparse-BSOS we let $k = 3$ and compute the first relaxations.

As for the Generalized Rosenbrock Function (Table 7) Sparse-PUT is able to find and certify the optimal solution at the first possible relaxation step, whereas Sparse-BSOS does not succeed to do so in the first three steps. However up to relaxation order $d = 3$ Sparse-BSOS is faster than Sparse-PUT and hence provides lower bounds for the objective function in less time and reasonably close to the optimal value.

BroydenBanded	rel.	Sparse-BSOS solution	rk	time	Sparse-PUT solution	rk	time
$n = 7$	$d = 1$	2.1371	2	11.5s	inf	-	-
	$d = 2$	2.7522	2	9.2s	inf	-	-
	$d = 3$	3.1161	2	11.0s	**3.4233**	1	15.2s
$n = 9$	$d = 1$	2.2171	3	77.0s	inf	-	-
	$d = 2$	2.8313	3	72.7s	inf	-	-
	$d = 3$	3.1354	3	87.1s	**3.3941**	1	105.6s
$n = 11$	$d = 1$	2.2968	3	160.3s	inf	-	-
	$d = 2$	3.0108	2	159.7s	inf	-	-
	$d = 3$	3.2638	4	190.7s	**3.3924**	1	215.1s
$n = 13$	$d = 1$	2.3353	3	282.9s	inf	-	-
	$d = 2$	3.0963	2.9	301.6s	inf	-	-
	$d = 3$	3.3268	4.4	367.5s	**3.4120**	1	357.7s
$n = 15$	$d = 1$	2.3555	3	445.2s	inf	-	-
	$d = 2$	3.1514	3.8	466.3s	inf	-	-
	$d = 3$	3.3617	3.8	509.1s	**3.4243**	1	545.2s

Table 9: Comparison Sparse-BSOS ($k = 3$) and Sparse-PUT on the Broyden Banded Function

5.2.2 Random medium scale quadratic and quartic test problems

So far we have compared Sparse-BSOS and Sparse-PUT on examples where the first possible relaxation of Sparse-PUT is exact. We now present examples where this is not the case or the first relaxation cannot even be computed because it is already to large. To this end we choose sparsity patterns with 40 to 80 variables in blocks of 10 to 40 and fixed overlap $\mathbf{o} = 5$. Note that the crucial parameter for the sparse hierarchies is not so much the total number of variables but rather the maximum block size of the sparsity pattern.

We change Problem (QP) slightly to generate the following sample of problems. Given a sparsity pattern $I = \{I_1, \dots, I_p\}$ we now consider:

$$\min_{\mathbf{x}} \left\{ \sum_{i=1}^{n} a_i x^4 + x^T A x + b^T x \ : \ 1 - x_i^2 \geq 0, \quad x_i \geq 0 \quad i = 1, \dots, n, \right\}, \tag{QP'}$$

where b is a random vector and the symmetric matrix A is randomly generate according to I. We verify that A has positive and negative eigenvalues to make sure that our problem is non-convex again. When $a = 0 \in \mathbb{R}^n$ we refer to (QP') as a *quadratic* problem. When mentioning the *quartic* problem (QP'), it means that we chose a randomly in $[-1, 1]^n$.

Table 10: We consider the quadratic instance of Problem (QP') and compute, whenever possible, the first two relaxations of BSOS, Sparse-BSOS and Sparse-PUT.

We were able to solve the first relaxation for all algorithms. However, as BSOS cannot benefit from the sparsity structure, it runs into numerical problems, in particular for the examples with $n = 120$ variables. Note that in [6] problems comparable to this one have been solved by BSOS. There the authors were able to solve the second relaxation for a quadratic problem with 100 variables. Indeed we were able to compute the second relaxation for some examples with 80 variables. However, this depends strongly on which SDP constraints BSOS deletes before handing over the system to the solver. We decided not to search for examples where we can compute the second relaxation.

When the solver does not run into numerical problems all solutions of the first relaxations coincide. This is because of the degree of the constraints and the objective function, the first

(QP')			BSOS			Sparse-BSOS			Sparse-PUT		
Quadratic	n	rel.	solution	rk	time	solution	rk	time	solution	rk	time
(7×10)	40	$d=1$	-1.2496e+02	4	3.5s	-1.2496e+02	4.0	1.8s	-1.2496e+02	4.0	0.8 s
		$d=2$	-4.4436e+01*	15	574.4s	-4.4457e+01	3.9	8.8s	**4.4326e+01**	1	45.8s
$(2 \times 20, 10)$	40	$d=1$	-1.6197e+02	5	3.4s	-1.6197e+02	5.0	0.7s	-1.6197e+02	5.0	0.7 s
		$d=2$	-5.9412e+01*	16	592.9s	-5.9447e+01	9.3	6.5s	-	-	-
(15×10)	80	$d=1$	-2.7552e+02*	8	71.7s	-2.7557e+02	4.0	0.8s	-2.7557e+02	4.0	0.8 s
		$d=2$	-	-	-	-1.0837e+02	2.4	2.7s	**1.0825e+02**	1	143.7s
(5×20)	80	$d=1$	-3.4782e+02*	5	86.6s	-3.4782e+02	5.0	1.6s	-3.4782e+02	5.0	1.6s
		$d=2$	-	-	-	-1.2536e+02	9.0	26.7s	-	-	-
$(2 \times 40, 10)$	80	$d=1$	-4.8983e+02*	5	69.3s	-4.8988e+02	5.0	3.9s	-4.8988e+02	5.0	4.2 s
		$d=2$	-	-	-	**-1.8564e+02**	1	66.8s	-	-	-
(23×10)	120	$d=1$	-3.8765e+02*	8	849.2s	-3.8765e+02	4.0	0.9s	-3.8765e+02	4.0	1.0 s
		$d=2$	-	-	-	-1.5884e+02	2.2	4.6s	-	-	-
$(7 \times 20, 15)$	120	$d=1$	-5.4921e+02*	5	772.0s	-5.4920e+02	4.6	3.6s	-5.4920e+02	4.6	3.8 s
		$d=2$	-	-	-	-2.3846e+02*	6.5	85.2s	-	-	-
$(3 \times 40, 15)$	120	$d=1$	-7.1721e+02 *	6	581.0s	-7.1720e+02	6.0	9.8s	-7.1720e+02	6.0	11.1s
		$d=2$	-	-	-	-2.3079e+02	12.2	143.8s	-	-	-

Table 10: Comparison BSOS($k=1$), Sparse-BSOS($k=1$), and Sparse-PUT on Quadratic Problem (QP'), $\mathbf{o}=5$

relaxations are more or less the same. In fact the sum of squares weights of the constraints in Sparse-BSOS are all of degree 0, i.e. they are non-negative scalar variables (and are implemented as such). BSOS and Sparse-BSOS use twice as many constraints because they not only consider the constraints $g_j(x) \geq 0$ but also the constraints $g_j(x) \leq 1$. This explains why Sparse-PUT is faster for the first relaxation.

In a number of cases the second relaxation of BSOS and Sparse-PUT could not be implemented because the psd variables become to big for the solver. Sparse-PUT is able to solve the second relaxation in two cases where the block size is 10 and we could actually certify optimality by the rank condition. The examples with same block size but $n = 120$ variables could not be solved because the solver runs out of memory. The same happened for examples with larger block size.

Only Sparse-BSOS was able to compute the second relaxation in all cases. Of course this second relaxation is weaker than the second relaxation of Sparse-PUT and Sparse-BSOS could only certify optimality in one case. However, when comparing the results with the certified values from Sparse-PUT, we see that they are actually quite close and much less time was spent to compute them. In all cases where Sparse-PUT could not solve the second relaxation, Sparse-BSOS could provide a lower bound that is much better that than the one provided by the the first relaxation of Sparse-PUT.

Table 11: The quartic version ($a \neq 0$) of Problem (QP'). As the degree of the objective function is now 4, the first relaxation of Sparse-PUT (i.e. with $d = 1$) is infeasible. Choosing $k = 1$ as fixed parameter, the respective relaxations with $d = 1$ of BSOS and Sparse-BSOS are not feasible either, and therefore one computes the second and the third relaxation, whenever possible.

Sparse-PUT could solve the second relaxation only for the patterns (7×10) and (15×10) in which case the optimal solution was attained and certified. With $k = 1$ (i.e. the SOS in (**??**) is of degree at most 2 and so less than the degree $(= 4)$ of the objective function) one still may solve higher relaxations with the Sparse-BSOS hierarchy. In contrast to the previous example the first relaxations of Sparse-BSOS and Sparse-PUT do not yield the same values. Indeed the resulting relaxations are actually different because the sum of squares in Sparse-PUT for $d = 2$ has degree 4 whereas in (Sparse-)BSOS it is of degree 2. Also in the sparse positivity certificate of Sparse-PUT, the SOS weights of the constraints are of degree 2 whereas in (S)BSOS the weights associated with the products of constraints are non-negative scalars. Note however that the values of the second relaxation of Sparse-BSOS for the patterns (7×10) and (15×10) are not so far from being optimal, which suggests that for the other test problems they are not so bad either. In any case Sparse-BSOS is able to provide lower bounds for larger block size, whereas the other hierarchies already overpassed their limit.

6 Conclusion

We have provided a sparse version of the BSOS hierarchy [6] so as to handle large scale polynomial optimization problems that satisfy a structured sparsity pattern. The positivity certificates used in the sparse BSOS hierarchy are coming from a sparse version of Krivine-Stengle Positivstellensatz, also proved in this paper.

We have tested the Sparse-BSOS hierarchy on a sample of non-convex problems randomly generated as well as on some typical examples from the literature. The results show that the hierarchy is able to solve small scale dense and large scale sparse polynomial optimization problems in reasonable computational time. In all our experiments where the problem size allowed to compare the dense and sparse versions, finite convergence in the latter took place

(QP')			BSOS			Sparse-BSOS			Sparse-PUT		
Quartic	n	rel.	solution	rk	time	solution	rk	time	solution	rk	time
(7×10)	40	$d = 2$	-5.4736e+01*	16	466.3s	-5.4802e+01	6.0	8.9s	**-5.2576e+01**	1	49.6s
		$d = 3$	-	-	-	-5.3047e+01*	3.1	319.0s	-	-	-
$(2 \times 20, 10)$	40	$d = 2$	-6.4481e+01*	17	606.3s	-6.4528e+01	8.7	5.9s	-	-	-
(15×10)	80	$d = 2$	-	-	-	-1.2037e+02	3.8	3.2s	**-1.1897e+02**	1	150.1s
		$d = 3$	-	-	-	-1.1950e+02	1.7	37.9s	-	-	-
(5×20)	80	$d = 2$	-	-	-	-1.2725e+02	8.4	23.5s	-	-	-
$(2 \times 40, 10)$	80	$d = 2$	-	-	-	-1.9476e+02	12.3	78.7s	-	-	-
(23×10)	120	$d = 2$	-	-	-	-1.6791e+02	4.0	8.7s	-	-	-
		$d = 3$	-	-	-	-1.6375e+02	1.2	103.0s	-	-	-
$(7 \times 20, 15)$	120	$d = 2$	-	-	-	2.1229e+02	9.5	54.7s	-	-	-
$(3 \times 40, 15)$	120	$d = 2$	-	-	-	-2.3994e+02	11.8	137.1s	-	-	-

Table 11: Comparison BSOS($k = 1$), Sparse-BSOS($k = 1$), and Sparse-PUT on Quartic Problem (QP'), $\mathbf{o} = 5$

whenever it took place for the former and moreover at the same relaxation order. This is remarkable, since in principle convergence of the dense version is at least faster than convergence for the sparse version.

In principle the sparse version [24] of the standard SOS hierarchy is hard to beat as it has proved to be efficient and successful in a number of cases. However in many cases the first relaxation of the Sparse-BSOS hierarchy is (provably) at least as good as the first relaxation of the former (e.g. for quadratic/quadratically constrained problems), and it can also provide better lower bounds in cases where the former cannot solve the first or second relaxation because the size the semidefinite constraint is till too large for the solver. (In this case we take full advantage of the fact that in Sparse-BSOS relaxations, the size of the semidefinite constraint is chosen and fixed.) We have seen some such examples. This is particularly interesting as it could help solve hard MINLP problems by Branch & Bound methods where for efficiency one needs to compute good quality lower bounds at each node of the search tree (and one does not need to provide the exact optimal value).

Crucial in our implementation is the comparison of coefficients to state that two polynomials are identical (instead of checking their values on a sample of generic points). This limits the application to problems with polynomials of small degree (say less than 4). In particular if some degree in the problem data is at least 6 and the block size is not small, the resulting SDP can become ill-conditioned when the relaxation order increases. Depending on the context in which one wants to use the sparse hierarchy, an alternative may be to implement polynomial identities by sampling.

Acknowledgement

The first author is very thankful to the National University of Singapore and Professor Kim-Chuan Toh for their hospitality and financial support during his stay in Singapore.

References

[1] Handelman D., *Representing polynomials by positive linear functions on compact convex polyhedra*, Pac. J. Math. **132**, pp. 35–62, 1988.

[2] Henrion D., Lasserre J.B., Lofberg J., *GloptiPoly 3: moments, optimization and semidefinite programming*, Optim. Methods and Softwares**24**, pp. 761–779, 2009.

[3] Josz C, Molzahn D.K., *Moment/Sum-of-Squares Hierarchy for Complex Polynomial Optimization*, preprint arxiv:1508.02068v2, 2016.

[4] Kojima M., Muramatsu M., *A note on sparse sos and sdp relaxations for polynomial optimization problems over symmetric cones*, Computational Optimization and Applications **42** pp. 31–41, 2009.

[5] Krivine J.L., *Anneaux préordonnés*, J. Anal. Math. **12**, pp. 307–326, 1964.

[6] Lasserre J.B., Toh K.C., Yang S., *A bounded degree SOS hierarchy for Polynomial Optimization*, EURO J. Comp. Optim. **5**, pp. 87–117, 2017.

[7] Lasserre J.B., *Global optimization with polynomials and the problem of moments*, SIAM J. Optim. **11**, pp. 796–817, 2001.

[8] Lasserre J.B., *Semidefinite programming vs. LP relaxations for polynomial programming*, Math. Oper. Res. **27**, pp. 347–360, 2002.

[9] Lasserre J.B., *A Lagrangian relaxation view of linear and semidefinite hierarchies*, SIAM J. Optim. **23**, pp. 1742–1756, 2013.

[10] Lasserre J.B., *Convergent SDP-relaxations in polynomial optimization with sparsity*, SIAM J. Optim. **17**, pp. 822–843, 2006.

[11] Lasserre J.B., *Moments, Positive Polynomials and Their Applications*, Imperial College Press, London, 2009.

[12] Lasserre J.B., *An Introduction to Polynomial and Semi-Algebraic Optimization*, Cambridge University Press, Cambridge, 2015.

[13] Löefberg, J, *YALMIP: A toolbox for modeling and optimization in MATLAB*, IEEE CCA/ISIS/CACSD, 2004.

[14] Marandi A., Dahl J., de Klerk E., *A numerical evaluation of the bounded degree sum-of-squares hierarchy of Lasserre, Toh, and Yang on the pooling problem*, Annals of Operations Research, pp. 1–26, 2017.

[15] Molzahn D.K., Hiskens I.A., *Sparsity-Exploiting moment-Based Relaxations of the Optimal Power Flow Problem*, IEEE Transactions on Power Systems **30**, pp. 3168–3180, 2015.

[16] Molzahn D.K., Baghsorkhi S.S., Hiskens I.A.,*Semidefinite Relaxations of Equivalent Optimal Power Flow Problems: An Illustrative Example*, IEEE International Symposium on Circuits and Systems (ISCAS), 2015.

[17] MOSEK ApS,*The MOSEK optimization toolbox for MATLAB manual*, 2015.

[18] Putinar M., *Positive polynomials on compact semi-algebraic sets*, Ind. Univ. Math. J. **42**, pp. 969–984, 1993.

[19] Stengle G., *A Nullstellensatz and a Positivstellensatz in semialgebraic geometry*, Math. Ann. **207**, pp. 87–97, 1974.

[20] Sturm, J.F., *Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones*, Optimization Methods and Software **11–12**,pp. 625–653, 1999.

[21] Toh K.C., Todd M.J., Tutuncu R.H., *SDPT3 — a Matlab software package for semidefinite programming*, Optimization Methods and Software **11**, pp. 545–581, 1999.

[22] Toh K.C., Todd M.J., Tutuncu R.H., *Solving semidefinite-quadratic-linear programs using SDPT3*, Mathematical Programming **95**, pp. 189–217, 2003.

[23] Vasilescu F.-H., *Spectral measures and moment problems*, in *Spectral Theory and Its Applications*, Theta Foundation, Bucharest, Romania, pp. 173–215, 2003.

[24] Waki S., Kim S., Kojima M., Maramatsu M., *Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity*, SIAM J. Optim. **17**, pp. 218–242, 2006.