SparseJSR: A Fast Algorithm to Compute Joint Spectral Radius via Sparse SOS Decompositions

Jie Wang, Martina Maggio and Victor Magron

***Abstract*— This paper focuses on the computation of the joint spectral radius (JSR), when the involved matrices are sparse. We provide a sparse variant of the procedure proposed by Parrilo and Jadbabaie to compute upper bounds of the JSR by means of sum-of-squares (SOS) programming. Our resulting iterative algorithm, called `SparseJSR`, is based on the *term sparsity* SOS (TSSOS) framework developed by Wang, Magron and Lasserre, which yields SOS decompositions of polynomials with arbitrary sparse supports. `SparseJSR` exploits the sparsity of the input matrices to significantly reduce the computational burden associated with the JSR computation. Our algorithmic framework is then successfully applied to compute upper bounds for JSR on randomly generated benchmarks as well as on problems arising from stability proofs of controllers, in relation with possible hardware and software faults.**

I. Introduction

Given a set of matrices $\mathcal{A} = \{A_1, \ldots, A_m\} \subseteq \mathbb{R}^{n\times n}$, the *joint spectral radius* (JSR) of $\mathcal{A}$ is defined by

$$\rho(\mathcal{A}) := \lim_{k\to\infty} \max_{\sigma\in\{1,\ldots,m\}^k} ||A_{\sigma_1}A_{\sigma_2}\cdots A_{\sigma_k}||^{\frac{1}{k}}, \quad (1)$$

which characterizes the maximal asymptotic growth rate of products of matrices from $\mathcal{A}$. Note that the value of $\rho(\mathcal{A})$ is independent of the choice of the norm used in (1). When $\mathcal{A}$ contains a single matrix, the JSR coincides with the usual spectral radius. Hence JSR can be viewed as a generalization of the usual spectral radius to the case of multiple matrices.

The concept of JSR was first introduced by Rota and Strang in [22] and since then has found applications in many areas such as the stability of switched linear dynamical systems, the continuity of wavelet functions, combinatorics and language theory, the capacity of some codes, the trackability of graphs. We refer the readers to [14] for a survey of the theory and applications of JSR.

Inspired by the various applications, there has been a lot of work on the computation of JSR; see e.g. [2], [5], [11], [12], [19], [20] to name a few. Unfortunately, it turns out that the exact computation and even the approximation of JSR are notoriously difficult [23]. It was proved in [6] that the problem of deciding whether $\rho(\mathcal{A}) \leq 1$ is undecidable even for $\mathcal{A}$ consisting of two matrices. Therefore, various methods focus on computing lower bounds and upper bounds for JSR [2], [5], [11], [19].

Parrilo and Jadbabaie proposed in [19] a sum-of-squares (SOS) approach which makes use of semidefinite programming (SDP) to compute a sequence of upper bounds $\{\rho_{SOS,2d}(\mathcal{A})\}_{d\geq 1}$ for $\rho(\mathcal{A})$. They proved that the sequence $\{\rho_{SOS,2d}(\mathcal{A})\}_{d\geq 1}$ converges to $\rho(\mathcal{A})$ when d increases. In practice, mostly often even small d (e.g., $d = 1, 2$) can provide upper bounds of good quality for $\rho(\mathcal{A})$. Once the upper bound coincides with a lower bound provided by other methods, then we obtain the exact value of the JSR. However, the computational burden of the SOS approach grows rapidly when the matrix size or d increases. Given the current state of SDP solvers, this approach can only handle matrices of modest sizes when $d \geq 2$.

For general polynomial optimization problems (POP), one way to reduce the computational cost of the associated SOS relaxations is to exploit the so-called *correlative sparsity pattern* relative to the variables of the POP [26]. To build these sparse SOS relaxations, one relies on the *correlative sparsity pattern (csp) graph* of the POP. The nodes of the csp graph are the variables and two nodes are connected via an edge when the corresponding variables appear in the same term of the objective function or in the same constraint involved in the POP. This approach was successfully used for several interesting applications, including certified roundoff error bounds [17], optimal powerflow problems [13], noncommutative optimization [15], Lipschitz constants of ReLU networks [9], robust geometric perception [32].

A complementary workaround is to take into account *term sparsity* (TS) of the input data to obtain sparse SOS relaxations, as recently studied in [28], [30], [29], yielding the so-called TSSOS framework. TSSOS relies on the *term sparsity pattern (tsp) graph* related to the input polynomials. To build the associated sparse SOS relaxations, one connects the nodes of this graph (corresponding to monomials from a monomial basis) whenever the product of the corresponding monomials either appears in the supports of input polynomials or is a monomial of even degree. Recent applications include learning and forecasting of linear systems [33], [34] via reformulation into noncommutative polynomial optimization and exploiting term sparsity to reduce the size of the associated relaxations. Note that term sparsity can be combined with correlative sparsity to reduce even further the size of the associated relaxations [18], [31].

The original underlying motivation of this paper was to apply term sparsity to improve the scalability of JSR computation arising from the study of deadline hit and deadline miss [16]. In this case, the computation of the control signal can fail due to a hardware and software fault, causing either no update or a delayed application of the control signal. The main application in this case is to determine how long the controller can operate in a faulty state (in which it does not complete the computation in due time, causing a deadline miss) before the stability of the system is compromised. The idea is to bound the JSR of products between state matrices associated to deadline hit and deadline miss by solving a POP

[2]. For such JSR problems, matrices of large sizes issued from applications reveal certain kinds of sparsity in many cases. A natural question is: can we exploit the sparsity of matrices to improve the scalability of the SOS approach and to compute upper bounds more efficiently? In this paper, we address this specific question.

Contributions and outline: In Section II, we recall preliminary background about SOS forms, chordal graphs and approximation of JSR via SOS programming. To make the current paper as self-contained as possible, Section III is dedicated to detailed explanation about sparse SOS decompositions via generation of smaller monomial bases and exploitation of the block structure of Gram matrices. Our main contribution is described in Section IV. We propose a so-called `SparseJSR` algorithm, which is based on the SOS approach and in coordination with the sparsity of matrices appearing within the JSR computation. The algorithm is implemented in the open-source Julia package, also called `SparseJSR`, and is freely available[1]. The performance of `SparseJSR` is then illustrated in Section V, first on randomly generated benchmarks, and then on benchmarks coming from the study of deadline hit/miss in [16]. Although our sparse version of the SOS approach is not guaranteed to produce upper bounds for JSR as good as the dense one with the same relaxation order, the numerical experiments in this paper demonstrate that our sparse approach is able to produce upper bounds of rather good quality but at a significantly cheaper computational cost compared to the dense approach.

II. Notation and Preliminaries

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a tuple of variables and $\mathbb{R}[\mathbf{x}] = \mathbb{R}[x_1, \ldots, x_n]$ be the ring of real n-variate polynomials. We use $\mathbb{R}[\mathbf{x}]_{2d}$ to denote the set of forms (i.e., homogeneous polynomials) of degree $2d$ for $d \in \mathbb{N}$. A polynomial $f \in \mathbb{R}[\mathbf{x}]$ can be written as $f(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathscr{A}} f_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$ with $f_{\boldsymbol{\alpha}} \in \mathbb{R}, \mathbf{x}^{\boldsymbol{\alpha}} = x_1^{\alpha_1} \cdots x_n^{\alpha_n}$. The *support* of f is defined by $\mathrm{supp}(f) := \{\boldsymbol{\alpha} \in \mathscr{A} \mid f_{\boldsymbol{\alpha}} \neq 0\}$. We use $|\cdot|$ to denote the cardinality of a set. For a nonempty finite set $\mathscr{A} \subseteq \mathbb{N}^n$, let $\mathbb{R}[\mathscr{A}]$ be the set of polynomials in $\mathbb{R}[\mathbf{x}]$ whose supports are contained in $\mathscr{A}$, i.e., $\mathbb{R}[\mathscr{A}] = \{f \in \mathbb{R}[\mathbf{x}] \mid \mathrm{supp}(f) \subseteq \mathscr{A}\}$ and let $\mathbf{x}^{\mathscr{A}}$ be the $|\mathscr{A}|$-dimensional column vector consisting of elements $\mathbf{x}^{\boldsymbol{\alpha}}, \boldsymbol{\alpha} \in \mathscr{A}$ (fix any ordering on $\mathbb{N}^n$). For convenience, we abuse notation a bit in this paper and use also $\mathscr{B} \subseteq \mathbb{N}^n$ (resp. $\boldsymbol{\beta} \in \mathbb{N}^n$) to denote a monomial set (resp. a monomial). For a positive integer r, let $\mathbf{S}^r$ be the set of $r \times r$ symmetric matrices and the set of $r \times r$ positive semidefinite (PSD) matrices is denoted by $\mathbf{S}^r_+$.

A. SOS forms

Given a form $f \in \mathbb{R}[\mathbf{x}]_{2d}$ with $d \in \mathbb{N}$, if there exist forms $f_1, \ldots, f_t \in \mathbb{R}[\mathbf{x}]_d$ such that $f = \sum_{i=1}^t f_i^2$, then we say that f is a *sum-of-squares* (SOS) form. The set of SOS forms in $\mathbb{R}[\mathbf{x}]_{2d}$ is denoted by $\Sigma_{n,2d}$. For $d \in \mathbb{N}$, let $\mathbb{N}^n_d := \{(\alpha_i)_{i=1}^n \in \mathbb{N}^n \mid \sum_{i=1}^n \alpha_i = d\}$ and assume that $f \in \mathbb{R}[\mathbf{x}]_{2d}$. Then deciding whether $f \in \Sigma_{n,2d}$ is equivalent to verifying the existence of a PSD matrix Q (which is called a *Gram matrix* for f) such that

$$f = (\mathbf{x}^{\mathbb{N}^n_d})^T Q \mathbf{x}^{\mathbb{N}^n_d}, \tag{2}$$

which can be formulated as a semidefinite program (SDP). The monomial basis $\mathbf{x}^{\mathbb{N}^n_d}$ used in (2) is called the *standard monomial basis*.

B. Chordal graphs and sparse matrices

An (undirected) *graph* $G(V, E)$ or simply G consists of a set of nodes V and a set of edges $E \subseteq \{\{v_i, v_j\} \mid (v_i, v_j) \in V \times V\}$. For a graph $G(V, E)$, a *cycle* of length k is a set of nodes $\{v_1, v_2, \ldots, v_k\} \subseteq V$ with $\{v_k, v_1\} \in E$ and $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, k-1$. A *chord* in a cycle $\{v_1, v_2, \ldots, v_k\}$ is an edge $\{v_i, v_j\}$ that joins two nonconsecutive nodes in the cycle. A graph is called a *chordal graph* if all its cycles of length at least four have a chord. Chordal graphs include some common classes of graphs, such as complete graphs, line graphs and trees, and have applications in sparse matrix theory [24]. Any non-chordal graph $G(V, E)$ can always be extended to a chordal graph $\overline{G}(V, \overline{E})$ by adding appropriate edges to E, which is called a *chordal extension* of $G(V, E)$. A *clique* $C \subseteq V$ of G is a subset of nodes where $\{v_i, v_j\} \in E$ for any $v_i, v_j \in C$. If a clique C is not a subset of any other clique, then it is called a *maximal clique*. It is known that maximal cliques of a chordal graph can be enumerated efficiently in linear time in the number of nodes and edges of the graph [4].

For a graph G, the chordal extension of G is usually not unique. We would prefer a chordal extension with the smallest clique number. Finding a chordal extension with the smallest clique number is an NP-complete problem in general. Fortunately, several heuristic algorithms are known to efficiently produce a good approximation [7].

Given a graph $G(V, E)$, a symmetric matrix Q with row and column indices labeled by V is said to have sparsity pattern G if $Q_{\boldsymbol{\beta}\boldsymbol{\gamma}} = Q_{\boldsymbol{\gamma}\boldsymbol{\beta}} = 0$ whenever $\boldsymbol{\beta} \neq \boldsymbol{\gamma}$ and $\{\boldsymbol{\beta}, \boldsymbol{\gamma}\} \notin E$. Let $\mathbf{S}_G$ be the set of symmetric matrices with sparsity pattern G. A matrix in $\mathbf{S}_G$ exhibits a block structure (after an appropriate permutation of rows and columns) as illustrated in Figure 1. Each block corresponds to a maximal clique of G. The maximal block size is the maximal size of maximal cliques of G, namely, the *clique number* of G. Note that there might be overlaps between blocks because different maximal cliques may share nodes.

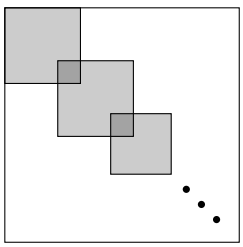

Fig. 1. A block structure of matrices in $\mathbf{S}_G$. The gray area indicates the positions of possible nonzero entries.

[1] https://github.com/wangjie212/SparseJSR

Given a maximal clique C of $G(V,E)$, we define an *indexing matrix* $P_C \in \mathbb{R}^{|C|\times|V|}$ as

$$[P_C]_{i\boldsymbol{\beta}} = \begin{cases} 1, & \text{if } C(i) = \boldsymbol{\beta}, \\ 0, & \text{otherwise}, \end{cases} \tag{3}$$

where $C(i)$ denotes the i-th node in C, sorted in the ordering compatible with V. Note that $Q_C = P_C Q P_C^T \in \mathbf{S}^{|C|}$ extracts a principal submatrix Q_C defined by the indices in the clique C from a symmetric matrix Q, and $Q = P_C^T Q_C P_C$ inflates a $|C|\times|C|$ matrix Q_C into a sparse $|V|\times|V|$ matrix Q.

PSD matrices with sparsity pattern G form a convex cone

$$\mathbf{S}_+^{|V|} \cap \mathbf{S}_G = \{Q \in \mathbf{S}_G \mid Q \succeq 0\}. \tag{4}$$

When the sparsity pattern graph G is chordal, the cone $\mathbf{S}_+^{|V|} \cap \mathbf{S}_G$ can be decomposed as a sum of simple convex cones, as stated in the following theorem.

Theorem 2.1 ([1]): Let $G(V,E)$ be a chordal graph and assume that $C_1,\dots,C_t$ are the list of maximal cliques of $G(V,E)$. Then a matrix $Q \in \mathbf{S}_+^{|V|} \cap \mathbf{S}_G$ if and only if there exists $Q_k \in \mathbf{S}_+^{|C_k|}$ for $k = 1,\dots,t$ such that $Q = \sum_{k=1}^t P_{C_k}^T Q_k P_{C_k}$.

For more details about sparse matrices and chordal graphs, the reader may refer to [24].

C. Approximating the joint spectral radius via SOS relaxations

The joint spectral radius (JSR) for a set of matrices $\mathcal{A} = \{A_1,\dots,A_m\} \subseteq \mathbb{R}^{n\times n}$ is given by

$$\rho(\mathcal{A}) := \lim_{k\to\infty} \max_{\sigma\in\{1,\dots,m\}^k} ||A_{\sigma_1}A_{\sigma_2}\cdots A_{\sigma_k}||^{\frac{1}{k}}. \tag{5}$$

Parrilo and Jadbabaie proposed to compute a sequence of upper bounds for $\rho(\mathcal{A})$ via SOS relaxations. The core idea is based on the following theorem.

Theorem 2.2 ([19], Theorem 2.2): Given a set of matrices $\mathcal{A} = \{A_1,\dots,A_m\} \subseteq \mathbb{R}^{n\times n}$, let p be a strictly positive form of degree $2d$ that satisfies

$$p(A_i\mathbf{x}) \le \gamma^{2d} p(\mathbf{x}), \quad \forall \mathbf{x} \in \mathbb{R}^n, \quad i = 1,\dots,m.$$

Then, $\rho(\mathcal{A}) \le \gamma$.

Replacing positive forms by more tractable SOS forms, Theorem 2.2 immediately suggests the following SOS relaxations indexed by $d \in \mathbb{N}\backslash\{0\}$ to compute a sequence of upper bounds for $\rho(\mathcal{A})$:

$$\rho_{\mathrm{SOS},2d}(\mathcal{A}) := \inf_{p\in\mathbb{R}[\mathbf{x}]_{2d},\gamma} \gamma \tag{6}$$
$$\text{s.t.} \begin{cases} p(\mathbf{x}) - ||\mathbf{x}||_2^{2d} \in \Sigma_{n,2d}, \\ \gamma^{2d}p(\mathbf{x}) - p(A_i\mathbf{x}) \in \Sigma_{n,2d}, \ 1 \le i \le m. \end{cases}$$

The terms "$||\mathbf{x}||_2^{2d}$" is added to make sure p is strictly positive. The optimization problem (6) can be solved via SDP by bisection on γ. It was shown in [19] that the upper bound $\rho_{\mathrm{SOS},2d}(\mathcal{A})$ satisfies the following theorem.

Theorem 2.3 ([19]): Let $\mathcal{A} = \{A_1,\dots,A_m\} \subseteq \mathbb{R}^{n\times n}$. For any integer $d \ge 1$, one has $m^{-\frac{1}{2d}}\rho_{\mathrm{SOS},2d}(\mathcal{A}) \le \rho(\mathcal{A}) \le \rho_{\mathrm{SOS},2d}(\mathcal{A})$.

It is immediate from Theorem 2.3 that $\{\rho_{\mathrm{SOS},2d}(\mathcal{A})\}_{d\ge1}$ converges to $\rho(\mathcal{A})$ when d increases.

III. Sparse SOS Decompositions

Deciding whether a form f is SOS involves solving an SDP whose size scales combinatorially with the number of variables and the degree of f. When f is sparse, it is possible to exploit the sparsity to construct an SDP of smaller size in order to reduce the computational burden. This includes two aspects: generating a smaller monomial basis and exploiting block structures for Gram matrices.

A. Generating a smaller monomial basis

Given a polynomial $f \in \mathbb{R}[\mathbf{x}]$, the *Newton polytope* of f is the convex hull of the support of f. It is known that the standard monomial basis $\mathbb{N}_d^n$ used in (2) can be replaced by the integer points in half of the Newton polytope of f, i.e., by

$$\mathscr{B} = \frac{1}{2}\mathrm{New}(f) \cap \mathbb{N}^n \subseteq \mathbb{N}_d^n. \tag{7}$$

See, e.g., [21] for a proof.

In [29], an algorithm named `GenerateBasis` was proposed to generate a smaller monomial basis for (2) than the one provided by the Newton polytope. Given the support of f, the output of `GenerateBasis` is an increasing chain of monomial sets:

$$\mathscr{B}_1 \subseteq \mathscr{B}_2 \subseteq \mathscr{B}_3 \subseteq \cdots \subseteq \mathbb{N}_d^n.$$

Each $\mathscr{B}_p$ can serve as a candidate monomial basis. In practice, if indexing the unknown Gram matrix from (2) by $\mathscr{B}_p$ leads to an infeasible SDP, then we turn to $\mathscr{B}_{p+1}$ until a feasible SDP is retrieved. In many cases, the algorithm `GenerateBasis` can provide a monomial basis smaller than the one given by (7); see [29] for such examples.

Remark 3.1: For all tested examples, $\mathscr{B}_1$ is a suitable monomial basis, but we do not know if this is true in general.

B. Term sparsity patterns

To derive a block structure for Gram matrices, we recall the concept of term sparsity patterns [28], [30], [29].

Definition 3.2: Let $f(\mathbf{x}) \in \mathbb{R}[\mathbf{x}]$ with $\mathrm{supp}(f) = \mathscr{A}$. Assume that $\mathscr{B}$ is a monomial basis. The *term sparsity pattern graph* $G(V,E)$ of f is defined by $V = \mathscr{B}$ and

$$E = \{\{\boldsymbol{\beta},\boldsymbol{\gamma}\} \mid \boldsymbol{\beta},\boldsymbol{\gamma} \in V,\ \boldsymbol{\beta} \ne \boldsymbol{\gamma},\ \boldsymbol{\beta}+\boldsymbol{\gamma} \in \mathscr{A} \cup 2\mathscr{B}\}, \tag{8}$$

where $2\mathscr{B} = \{2\boldsymbol{\beta} \mid \boldsymbol{\beta} \in \mathscr{B}\}$.

For a term sparsity pattern graph $G(V,E)$, we denote a chordal extension of G by $\overline{G}(V,\overline{E})$.

Example 3.3: Consider the polynomial $f = x_1^4 + x_2^4 + x_3^4 + x_1x_2x_3^2 + x_1x_2^2x_3$. A monomial basis for f is $\{x_1^2, x_2^2, x_3^2, x_1x_2, x_1x_3, x_2x_3\}$. See Figure 2 for the term sparsity pattern graph G of f and a chordal extension $\overline{G}$ of G.

Given a sparse SOS form $f(\mathbf{x}) \in \mathbb{R}[\mathscr{A}]$ and a monomial basis $\mathscr{B}$, generally a Gram matrix for f is not necessarily sparse. Let G be the term sparsity pattern graph of f and $\overline{G}$ a chordal extension of G. To get a sparse SOS decomposition of f, we then impose the sparsity pattern $\overline{G}$ to the Gram

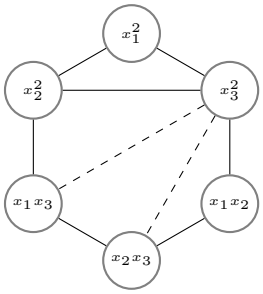

Fig. 2. The term sparsity pattern graph and a chordal extension for Example 3.3. The dashed edges are added after a chordal extension.

matrix for f, i.e., we consider the following subset of SOS forms in $\Sigma_{n,2d}$:

$$\Sigma_{\mathscr{A}} := \{f \in \mathbb{R}[\mathscr{A}] \mid \exists Q \in \mathbf{S}_+^{|\mathscr{B}|} \cap \mathbf{S}_{\overline{G}} \text{ s.t. } f = (\mathbf{x}^{\mathscr{B}})^T Q \mathbf{x}^{\mathscr{B}}\}.$$

Theorem 2.1 enables us to give the following sparse SOS decompositions for polynomials in $\Sigma_{\mathscr{A}}$.

Theorem 3.4 ([28], Theorem 3.3): Given $\mathscr{A} \subseteq \mathbb{N}^n$, assume that $\mathscr{B} = \{\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_r\}$ is a monomial basis and G is the term sparsity pattern graph. Let $C_1, C_2, \ldots, C_t \subseteq V$ denote the list of maximal cliques of $\overline{G}$ (a chordal extension of G) and $\mathscr{B}_k = \{\boldsymbol{\beta}_i \in \mathscr{B} \mid i \in C_k\}, k = 1, 2, \ldots, t$. Then, $f(\mathbf{x}) \in \Sigma_{\mathscr{A}}$ if and only if there exist $f_k(\mathbf{x}) = (\mathbf{x}^{\mathscr{B}_k})^T Q_k \mathbf{x}^{\mathscr{B}_k}$ with $Q_k \in \mathbf{S}_+^{|C_k|}$ for $k = 1, \ldots, t$ such that

$$f(\mathbf{x}) = \sum_{k=1}^{t} f_k(\mathbf{x}). \tag{9}$$

By virtue of Theorem 3.4, checking membership in $\Sigma_{\mathscr{A}}$ boils down to solving an SDP problem involving PSD matrices of small sizes if each maximal clique of $\overline{G}$ has a small size relative to the original matrix. This might significantly reduce the overall computational cost.

IV. The SparseJSR Algorithm

In this section, we propose an algorithm for bounding JSR based on the sparse SOS decomposition discussed in the previous section. To this end, we first establish a hierarchy of sparse supports for the auxiliary form $p(\mathbf{x})$ used in the SOS program (6).

Let $\mathcal{A} = \{A_1, \ldots, A_m\} \subseteq \mathbb{R}^{n \times n}$ be a tuple of matrices. Fixing a relaxation order d, let $p_0(\mathbf{x}) = \sum_{j=1}^n c_j x_j^{2d}$ with random coefficients $c_j \in (0, 1)$ and let $\mathscr{A}^{(0)} = \operatorname{supp}(p_0)$. Then for $s \in \mathbb{N}\backslash\{0\}$, we iteratively define

$$\mathscr{A}^{(s)} := \mathscr{A}^{(s-1)} \cup \bigcup_{i=1}^{m} \operatorname{supp}(p_{s-1}(A_i\mathbf{x})), \tag{10}$$

where $p_{s-1}(\mathbf{x}) = \sum_{\boldsymbol{\alpha} \in \mathscr{A}^{(s-1)}} c_{\boldsymbol{\alpha}} \mathbf{x}^{\boldsymbol{\alpha}}$ with random coefficients $c_{\boldsymbol{\alpha}} \in (0, 1)$. Note that the particular form of $p_0(\mathbf{x})$ is chosen such that $\mathscr{A}^{(s)}$ contains all possible homogeneous monomials of degree $2d$ that are "compatible" with the couplings between variables $x_1, \ldots, x_n$ introduced by the mappings $\mathbf{x} \mapsto A_i\mathbf{x}$ for all i. It is clear that

$$\mathscr{A}^{(1)} \subseteq \cdots \subseteq \mathscr{A}^{(s)} \subseteq \mathscr{A}^{(s+1)} \subseteq \cdots \subseteq \mathbb{N}_{2d}^n \tag{11}$$

and the sequence $\{\mathscr{A}^{(s)}\}_{s \geq 1}$ stabilizes in finitely many steps. We point out that it is not guaranteed a hierarchy of sparse supports is always retrieved in (11) even if all A_i are sparse. For instance, if some matrix $A_i \in \mathcal{A}$ has a fully dense row, then by definition, one immediately has $\mathscr{A}^{(1)} = \mathbb{N}_{2d}^n$. In this case, the sparsity of $\mathcal{A}$ cannot be exploited by the present method. This obstacle might be overcome if a more suitable $p_0(\mathbf{x})$ is chosen taking into account the sparsity pattern of $\mathcal{A}$, which we leave for future investigation.

On the other hand, if the matrices in $\mathcal{A}$ have some common zero columns, then a hierarchy of sparse supports must be retrieved.

Proposition 4.1: Let $\mathcal{A} = \{A_1, \ldots, A_m\} \subseteq \mathbb{R}^{n \times n}$ and assume that the matrices in $\mathcal{A}$ have common zero columns indexed by $J \subseteq [n] := \{1, 2, \ldots, n\}$. Let $\tilde{\mathbb{N}}_{2d}^{n-|J|} := \{(\alpha_i)_{i \in [n]} \in \mathbb{N}^n \mid (\alpha_i)_{i \in [n]\backslash J} \in \mathbb{N}_{2d}^{n-|J|}, \alpha_i = 0 \text{ for } i \in J\}$ and $\mathbf{b}_j := \{(\alpha_i)_{i \in [n]} \in \mathbb{N}^n \mid \alpha_j = 2d, \alpha_i = 0 \text{ for } i \neq j\}$ for $j \in [n]$. Then $\mathscr{A}^{(s)} \subseteq \tilde{\mathbb{N}}_{2d}^{n-|J|} \cup \{\mathbf{b}_j\}_{j \in J}$ for all $s \geq 1$.

Proof: Let us do induction on s. It is obvious that $\mathscr{A}^{(0)} \subseteq \tilde{\mathbb{N}}_{2d}^{n-|J|} \cup \{\mathbf{b}_j\}_{j \in J}$. Now assume $\mathscr{A}^{(s)} \subseteq \tilde{\mathbb{N}}_{2d}^{n-|J|} \cup \{\mathbf{b}_j\}_{j \in J}$ for some $s \geq 0$. Since the variables effectively involved in $p_s(A_j\mathbf{x})$ are contained in $\{x_i\}_{i \in [n]\backslash J}$, we have $\operatorname{supp}(p_s(A_j\mathbf{x})) \subseteq \tilde{\mathbb{N}}_{2d}^{n-|J|}$ for $j = 1, \ldots, m$. This combined with the induction hypothesis yields $\mathscr{A}^{(s+1)} \subseteq \tilde{\mathbb{N}}_{2d}^{n-|J|} \cup \{\mathbf{b}_j\}_{j \in J}$ as desired. ∎

For each $s \geq 1$, by restricting $p(\mathbf{x})$ to forms with the sparse support $\mathscr{A}^{(s)}$, (6) now reads as

$$\begin{aligned} &\inf_{p \in \mathbb{R}[\mathscr{A}^{(s)}], \gamma} \gamma \\ &\text{s.t.} \begin{cases} p(\mathbf{x}) - ||\mathbf{x}||_2^{2d} \in \Sigma_{n,2d}, \\ \gamma^{2d} p(\mathbf{x}) - p(A_i\mathbf{x}) \in \Sigma_{n,2d}, \ 1 \leq i \leq m. \end{cases} \end{aligned} \tag{12}$$

Let $\mathscr{A}_i^{(s)} = \mathscr{A}^{(s)} \cup \operatorname{supp}(p_s(A_i\mathbf{x})$ for $i = 1, \ldots, m$. In order to exploit the sparsity present in (12), we then replace $\Sigma_{n,2d}$ with $\Sigma_{\mathscr{A}^{(s)}}$ or $\Sigma_{\mathscr{A}_i^{(s)}}$ in (12). Consequently we obtain a hierarchy of SOS relaxations indexed by s for a fixed d:

$$\begin{aligned} \rho_{s,2d}(\mathcal{A}) := &\inf_{p \in \mathbb{R}[\mathscr{A}^{(s)}], \gamma} \gamma \\ &\text{s.t.} \begin{cases} p(\mathbf{x}) - ||\mathbf{x}||_2^{2d} \in \Sigma_{\mathscr{A}^{(s)}}, \\ \gamma^{2d} p(\mathbf{x}) - p(A_i\mathbf{x}) \in \Sigma_{\mathscr{A}_i^{(s)}}, \ 1 \leq i \leq m. \end{cases} \end{aligned} \tag{13}$$

We call the index s the *sparse order* of (13). As in the dense case, the optimization problem (13) can be solved via SDP by bisection on γ. Moreover, we have the following theorem.

Theorem 4.2: Let $\mathcal{A} = \{A_1, \ldots, A_m\} \subseteq \mathbb{R}^{n \times n}$. For any integer $d \geq 1$, one has $\rho_{\mathrm{SOS},2d}(\mathcal{A}) \leq \cdots \leq \rho_{s,2d}(\mathcal{A}) \leq \cdots \leq \rho_{2,2d}(\mathcal{A}) \leq \rho_{1,2d}(\mathcal{A})$.

Proof: For any fixed $d \in \mathbb{N}\backslash\{0\}$, because of (11), it is clear that the feasible set of (13) with the sparse order s is contained in the feasible set of (13) with the sparse order $s+1$, which is in turn contained in the feasible set of (6). This yields the desired conclusion. ∎

So we can propose the algorithm `SparseJSR` that computes a non-increasing sequence of upper bounds for the JSR of a tuple of matrices via solving (13) for any fixed d. By varying the relaxation order d and the sparse order s, `SparseJSR` offers a trade-off between the computational cost and the quality of the obtained upper bound. The

correctness of SparseJSR is guaranteed by Theorem 2.2 and Theorem 4.2.

V. Numerical Experiments

In this section, we present numerical experiments for the proposed algorithm SparseJSR, which is implemented in the Julia package also named SparseJSR and based on the TSSOS package used in [30], [29], [31]. SparseJSR utilizes the Julia packages LightGraphs [8] to handle graphs, ChordalGraph [27] to generate chordal extensions and JuMP [10] to model SDP. Finally, SparseJSR relies on the SDP solver MOSEK [3] to solve SDP. For the comparison purpose, we also implement the dense SOS relaxation (6) in SparseJSR using the same SDP solver MOSEK. For all examples, the sparse order s is set as 1, the tolerance for bisection is set as $\epsilon = 1 \times 10^{-5}$, and the initial interval for bisection is set as $[0, 2]$. To measure the quality of upper bounds that we obtain, a lower bound for JSR is also computed using the MATLAB JSR toolbox [25]. All examples were computed on an Intel Core i5-8265U@1.60GHz CPU with 8GB RAM memory. The notations that we use are listed in Table I.

TABLE I
THE NOTATIONS

m	the number of matrices in $\mathcal{A}$
n	the size of matrices in $\mathcal{A}$
lb	lower bounds for JSR given by the JSR toolbox
ub	upper bounds for JSR given by SparseJSR
d	the relaxation order
mb	the maximal size of PSD blocks
time	running time in seconds
-	> 3600 s
$*$	an out of memory error

We consider randomly generated examples and examples arising from the study of deadline hit/miss in [16].

A. Randomly generated examples

We generate random sparse matrices as follows[2]: first call the function "erdos_renyi" in the Julia packages LightGraphs to generate a random directed graph G with n nodes and $n + 10$ edges; for each edge (i, j) of G, put a random number in $[-1, 1]$ on the position (i, j) of the matrix and put zeros for other positions. We compute an upper bound of the JSR for pairs of such matrices with different sparsity patterns using the first-order SOS relaxations. The results are displayed in Table II. It is evident that the sparse approach is much more efficient than the dense approach. For instance, the dense approach takes over 3600 s when the size of matrices is greater than 100 while the sparse approach can easily handle matrices of size 120 within 12 s. Both the dense approach and the sparse approach produce upper bounds which are within 0.05 greater than the corresponding lower bounds.

[2]Available at https://wangjie212.github.io/jiewang/code.html.

TABLE II
RANDOMLY GENERATED EXAMPLES WITH $d = 1$ AND $m = 2$

n	lb	Sparse ($d = 1$)			Dense ($d = 1$)		
		time	ub	mb	time	ub	mb
20	0.7894	0.74	0.8192	10	1.88	0.7967	20
30	0.8502	1.65	0.8666	10	7.79	0.8523	30
40	0.9446	2.68	0.9446	14	25.6	0.9446	40
50	0.8838	2.97	0.9102	14	55.9	0.8838	50
60	0.7612	3.64	0.7843	13	171	0.7612	60
70	0.9629	4.35	0.9629	11	308	0.9629	70
80	0.9345	5.95	0.9399	15	743	0.9345	80
90	0.8020	6.27	0.8465	14	1282	0.8020	90
100	0.8642	8.15	0.9132	13	2568	0.8659	100
110	0.8355	9.59	0.8839	15	-	-	-
120	0.7483	11.7	0.7735	16	-	-	-

B. Examples from control systems

Here we consider examples from [16], where the dynamics of closed-loop systems are given by the combination of a plant and a one-step delay controller that stabilizes the plant. The closed-loop system evolves according to either a completed or a missed computation. In the case of a deadline hit, the closed-loop state matrix is A_H. In the case of a deadline miss, the associated closed-loop state matrix is A_M. The computational platform (hardware and software) ensures that no more than $m-1$ deadlines are missed consecutively. The set of possible realisations $\mathcal{A}$ of such a system contains either a single hit or at most $m-1$ misses followed by a hit, namely $\mathcal{A} := \{A_H A_M^i \mid 0 \le i \le m-1\}$. Then, the closed-loop system that can switch between the realisations included in $\mathcal{A}$ is asymptotically stable if and only if $\rho(\mathcal{A}) < 1$. This gives an indication for scheduling and control co-design, in which the hardware and software platform must guarantee that the maximum number of deadlines missed consecutively does not interfere with stability requirements.

In Table III and Table IV, we report the results obtained for various control systems with n states, under $m - 1$ deadline misses, by applying the dense and sparse relaxations with relaxation orders $d = 1$ and $d = 2$, respectively. The examples are randomly generated, i.e., our script generates a random system and then tries to control it[3].

In Table III, we fix $m = 5$ and vary n from 20 to 110. For these examples, surprisingly the dense and sparse approaches with the relaxation order $d = 1$ always produce the same upper bounds. As we can see from the table, the sparse approach is more scalable and efficient than the dense one.

In Table IV, we vary m from 2 to 11 and vary n from 6 to 24. For each instance, one has $mb = 10$ for the sparse approach. The column "ub" indicates the upper bound given by the dense approach with the relaxation order $d = 1$. For these examples, with the relaxation order $d = 2$, the sparse approach produces upper bounds that are very close to those given by the dense approach. And again the sparse approach is more scalable and more efficient than the dense one.

[3]Available at https://wangjie212.github.io/jiewang/code.html.

TABLE III

RESULTS FOR CONTROL SYSTEMS WITH $d = 1$ AND $m = 5$

n	lb	Sparse ($d = 1$)			Dense ($d = 1$)		
		time	ub	mb	time	ub	mb
20	0.9058	1.78	0.9316	12	9.92	0.9316	20
20	0.8142	1.62	0.8142	12	9.08	0.8142	20
30	1.4682	4.30	1.5132	14	57.8	1.5131	30
30	1.0924	4.42	1.0961	14	65.4	1.0961	30
40	1.1648	9.29	1.1977	16	249	1.1977	30
40	0.9772	9.69	0.9804	16	259	0.9804	30
50	1.3153	17.3	1.3248	18	660	1.3248	50
50	1.1884	17.5	1.1884	18	680	1.1884	50
60	1.8366	29.7	1.8820	20	2049	1.8820	60
60	1.3259	30.7	1.3259	20	1776	1.3259	60
70	1.8135	54.2	1.8578	22	-	-	-
70	1.2727	53.9	1.2727	22	-	-	-
80	2.3005	85.3	2.3445	24	-	-	-
80	1.4262	85.6	1.4262	24	-	-	-
90	1.8745	133	1.9020	26	-	-	-
90	1.4452	132	1.4452	26	-	-	-
100	2.2316	196	2.2733	28	*	*	*
100	1.5267	195	1.5267	28	*	*	*
110	2.3597	280	2.3943	30	*	*	*
110	1.5753	287	1.5753	30	*	*	*

TABLE IV

RESULTS FOR CONTROL SYSTEMS WITH $d = 2$

m	n	lb	ub	Sparse ($d = 2$)		Dense ($d = 2$)		
				time	ub	time	ub	mb
2	6	0.9464	0.9782	0.42	0.9547	1.87	0.9539	21
3	8	0.7218	0.7467	0.60	0.7310	13.4	0.7305	36
4	10	0.7458	0.7738	0.75	0.7564	107	0.7554	55
5	12	0.8601	0.8937	1.08	0.8706	1157	0.8699	78
6	14	0.7875	0.8107	1.32	0.7958	-	-	-
7	16	1.1110	1.1531	1.81	1.1182	*	*	*
8	18	1.0487	1.0881	2.05	1.0569	*	*	*
9	20	0.7570	0.7808	2.52	0.7660	*	*	*
10	22	0.9911	1.0315	2.70	1.0002	*	*	*
11	24	0.7339	0.7530	3.67	0.7418	*	*	*

REFERENCES

[1] J. AGLER, W. HELTON, S. MCCULLOUGH, AND L. RODMAN, *Positive semidefinite matrices with a given sparsity pattern*, Linear algebra and its applications, 107 (1988), pp. 101–149.

[2] A. A. AHMADI, R. M. JUNGERS, P. A. PARRILO, AND M. ROOZBEHANI, *Joint spectral radius and path-complete graph lyapunov functions*, SIAM Journal on Control and Optimization, 52 (2014).

[3] M. APS, *The MOSEK optimization toolbox. Version 8.1.*, 2017.

[4] J. R. BLAIR AND B. PEYTON, *An introduction to chordal graphs and clique trees*, in Graph theory and sparse matrix computation, Springer, 1993, pp. 1–29.

[5] V. D. BLONDEL AND Y. NESTEROV, *Polynomial-time computation of the joint spectral radius for some sets of nonnegative matrices*, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 865–876.

[6] V. D. BLONDEL AND J. N. TSITSIKLIS, *The boundedness of all products of a pair of matrices is undecidable*, Systems & Control Letters, 41 (2000), pp. 135–140.

[7] H. L. BODLAENDER AND A. M. KOSTER, *Treewidth computations i. upper bounds*, Information and Computation, 208 (2010).

[8] S. BROMBERGER, J. FAIRBANKS, AND OTHER CONTRIBUTORS, *Juliagraphs/lightgraphs.jl: an optimized graphs package for the julia programming language*, 2017.

[9] T. CHEN, J.-B. LASSERRE, V. MAGRON, AND E. PAUWELS, *Semialgebraic Optimization for bounding Lipschitz constants of ReLU networks*, Proceeding of Advances in Neural Information Processing Systems, 33 (2020).

[10] I. DUNNING, J. HUCHETTE, AND M. LUBIN, *JuMP: A modeling language for mathematical optimization*, SIAM Review, 59 (2017), pp. 295–320.

[11] G. GRIPENBERG, *Computing the joint spectral radius*, Linear Algebra and its Applications, 234 (1996), pp. 43–60.

[12] N. GUGLIELMI AND M. ZENNARO, *Finding extremal complex polytope norms for families of real matrices*, SIAM Journal on Matrix Analysis and Applications, 31 (2009), pp. 602–620.

[13] C. JOSZ AND D. K. MOLZAHN, *Lasserre hierarchy for large scale polynomial optimization in real and complex variables*, SIAM Journal on Optimization, 28 (2018), pp. 1017–1048.

[14] R. JUNGERS, *The joint spectral radius: theory and applications*, vol. 385, Springer Science & Business Media, 2009.

[15] I. KLEP, V. MAGRON, AND J. POVH, *Sparse noncommutative polynomial optimization*, arXiv:1909.00569, (2019).

[16] M. MAGGIO, A. HAMANN, E. MAYER-JOHN, AND D. ZIEGENBEIN, *Control-system stability under consecutive deadline misses constraints*, in 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[17] V. MAGRON, G. CONSTANTINIDES, AND A. DONALDSON, *Certified Roundoff Error Bounds Using Semidefinite Programming*, ACM Trans. Math. Softw., 43 (2017), pp. 1–34.

[18] J. MILLER, Y. ZHENG, M. SZNAIER, AND A. PAPACHRISTODOULOU, *Decomposed structured subsets for semidefinite and sum-of-squares optimization*, arXiv:1911.12859, (2019).

[19] P. A. PARRILO AND A. JADBABAIE, *Approximation of the joint spectral radius using sum of squares*, Linear Algebra and its Applications, 428 (2008), pp. 2385–2402.

[20] V. Y. PROTASOV, R. M. JUNGERS, AND V. D. BLONDEL, *Joint spectral characteristics of matrices: a conic programming approach*, SIAM Journal on Matrix Analysis and Applications, 31 (2010), pp. 2146–2162.

[21] B. REZNICK, *Extremal psd forms with few terms*, Duke mathematical journal, 45 (1978), pp. 363–374.

[22] G.-C. ROTA AND W. STRANG, *A note on the joint spectral radius*, in Gian-Carlo Rota on Analysis and Probability: Selected Papers and Commentaries, 1960.

[23] J. N. TSITSIKLIS AND V. D. BLONDEL, *The Lyapunov exponent and joint spectral radius of pairs of matrices are hard—when not impossible—to compute and to approximate*, Mathematics of Control, Signals and Systems, 10 (1997), pp. 31–40.

[24] L. VANDENBERGHE, M. S. ANDERSEN, ET AL., *Chordal graphs and semidefinite optimization*, Foundations and Trends® in Optimization, 1 (2015), pp. 241–433.

[25] G. VANKEERBERGHEN, J. HENDRICKX, AND R. M. JUNGERS, *JSR: A toolbox to compute the joint spectral radius*, in Proceedings of the 17th international conference on Hybrid systems: computation and control, 2014, pp. 151–156.

[26] H. WAKI, S. KIM, M. KOJIMA, AND M. MURAMATSU, *Sums of Squares and Semidefinite Programming Relaxations for Polynomial Optimization Problems with Structured Sparsity*, SIAM Journal on Optimization, 17 (2006), pp. 218–242.

[27] J. WANG, *ChordalGraph: A Julia Package to Handle Chordal Graphs*, 2020.

[28] J. WANG, H. LI, AND B. XIA, *A new sparse SOS decomposition algorithm based on term sparsity*, in Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation, 2019, pp. 347–354.

[29] J. WANG, V. MAGRON, AND J.-B. LASSERRE, *Chordal-TSSOS: a moment-SOS hierarchy that exploits term sparsity with chordal extension*, SIAM Journal on Optimization, 31 (2021), pp. 114–141.

[30] ———, *TSSOS: A Moment-SOS hierarchy that exploits term sparsity*, SIAM Journal on Optimization, 31 (2021), pp. 30–58.

[31] J. WANG, V. MAGRON, J.-B. LASSERRE, AND N. H. A. MAI, *CS-TSSOS: Correlative and term sparsity for large-scale polynomial optimization*, arXiv:2005.02828, (2020).

[32] H. YANG AND L. CARLONE, *One ring to rule them all: Certifiably robust geometric perception with outliers*, arXiv:2006.06769, (2020).

[33] Q. ZHOU AND J. MARECEK, *Proper learning of linear dynamical systems as a non-commutative polynomial optimisation problem*, arXiv:2002.01444, (2020).

[34] Q. ZHOU, J. MARECEK, AND R. N. SHORTEN, *Fairness in forecasting and learning linear dynamical systems*, arXiv:2006.07315, (2020).