Solving Large-Scale Robust Stability Problems by Exploiting the Parallel Structure of Polya's Theorem

Reza Kamyar, Matthew M. Peet, and Yulia Peet

***Abstract*—In this paper, we propose a distributed computing approach to solving large-scale robust stability problems on the simplex. Our approach is to formulate the robust stability problem as an optimization problem with polynomial variables and polynomial inequality constraints. We use Polya's theorem to convert the polynomial optimization problem to a set of highly structured linear matrix inequalities (LMIs). We then use a slight modification of a common interior-point primal-dual algorithm to solve the structured LMI constraints. This yields a set of extremely large yet structured computations. We then map the structure of the computations to a decentralized computing environment consisting of independent processing nodes with a structured adjacency matrix. The result is an algorithm which can solve the robust stability problem with the same per-core complexity as the deterministic stability problem with a conservatism which is only a function of the number of processors available. Numerical tests on cluster computers and supercomputers demonstrate the ability of the algorithm to efficiently utilize hundreds and potentially thousands of processors and analyze systems with $100+$ dimensional state-space. The proposed algorithms can be extended to perform stability analysis of nonlinear systems and robust controller synthesis.**

***Index Terms*—Decentralized computing, large-scale systems, polynomial optimization, robust stability.**

I. Introduction

THIS PAPER addresses the problem of stability of large-scale systems with several unknown parameters. Control system theory, when applied in practical situations, often involves the use of large state-space models, typically due to the inherent complexity of the system, the interconnection of subsystems, or the reduction of an infinite-dimensional or PDE model to a finite-dimensional approximation. One approach to dealing with such large-scale models has been to use model reduction techniques, such as balanced truncation [1]. However, the use of model reduction techniques is not necessarily robust and can result in arbitrarily large errors. In addition to large state space, practical problems often contain uncertainty in the model due to modeling errors, linearization, or fluctuation in the operating conditions. The problem of stability and control of systems with uncertainty has been widely studied. See, for example, the texts [2]–[5]. However, a limitation of existing computational methods for analysis and control of systems with uncertainty is high complexity. This is a consequence of the fact that the problem of robust stability of systems with parametric uncertainty is known to be NP-hard [6], [7]. The result is that for systems with parametric uncertainty and with hundreds of states, existing algorithms will fail with the primary point of failure usually being a lack of unallocated memory.

In this paper, we seek to distribute the computation laterally over an array of processors within the context of existing computational resources. Specifically, we seek to utilize cluster-computing, supercomputing, and graphics-processing unit (GPU)-computing architectures. When designing algorithms to run in a parallel computing environment, one must both synchronize computational tasks among the processors while minimizing communication overhead among the processors. This can be difficult, since each architecture has a specific communication graph. We account for communication by explicitly modeling the required communication graph between processors. This communication graph is then mapped to the processor architecture using the message-passing interface (MPI) [8]. While there are many algorithms for robust stability analysis and control of linear systems, ours is the first which explicitly accounts for the processing architecture in the emerging multicore computing environment.

Our approach to robust stability is based on the well-established use of parameter-dependent quadratic-in-the-state (QITS) Lyapunov functions. The use of parameter-dependent Lyapunov QITS functions eliminates the conservativity associated with, for example, quadratic stability [9], [10], at the cost of requiring some restriction on the rate of parameter variation. Specifically, our QITS Lyapunov variables are polynomials in the vector of uncertain parameters. This is a generalization of the use of QITS Lyapunov functions with affine parameter dependence as in [11] and expanded in, for example, [11]–[14]. The use of polynomial QITS Lyapunov variables can be motivated by [15], wherein it is shown that any feasible parameter-dependent LMI with parameters inside a compact set has a polynomial solution or [16] wherein it is shown that local stability of a nonlinear vector field implies the existence of a polynomial Lyapunov function.

There are several results which use polynomial QITS Lyapunov functions to prove robust stability. In most cases, the stability problem is reduced to the general problem of optimization of polynomial variables subject to linear matrix in-

Manuscript received November 21, 2011; revised September 24, 2012; accepted February 26, 2013. Date of publication March 19, 2013; date of current version July 19, 2013. This work was supported entirely by funding from the National Science Foundation under Award CMMI-1100376. Recommended by Fabrizio Dabbene.

R. Kamyar is with the Department of Mechanical Engineering, Cybernetic Systems and Controls Laboratory, Arizona State University, Tempe, AZ 85281 USA (e-mail: rkamyar@asu.edu).

M. M. Peet and Y. Peet are with the School for Engineering of Matter, Transport, and Energy, Engineering Research Center, Arizona State University, Tempe, AZ 85281 USA (e-mail: mpeet@asu.edu; ypeet@asu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAC.2013.2253253

equality (LMI) constraints—an NP-hard problem [17]. To avoid NP-hardness, the polynomial optimization problem is usually solved in an asymptotic manner by posing a sequence of sufficient conditions of increasing accuracy and decreasing conservatism. For example, building on the result in [15], [18] provides a sequence of increasingly precise LMI conditions for robust stability analysis of linear systems with affine dependency on uncertain parameters on the complex unit ball. Necessary and sufficient stability conditions for linear systems with one uncertain parameter are derived in [19], providing an explicit bound on the degree of the polynomial-type Lyapunov function. The result is extended to multiparameter-dependent linear systems in [20]. Another important approach to the optimization of polynomials is the sum of squares (SOS) methodology which replaces the polynomial positivity constraint with the constraint that the polynomial admits a representation as a sum of squares of polynomials [21]–[24]. A version of this theorem for polynomials with matrix coefficients can be found in [23]. While we have worked extensively with the SOS methodology, we have not, as of yet, been able to adapt algorithms for solving the resulting LMI conditions to a parallel computing environment. Finally, there have been several results in recent years on the use of Polya's theorem to solve polynomial optimization problems [25] on the simplex. An extension of the Polya's theorem for uncertain parameters on the multisimplex or hypercube can be found in [26]. The approach presented in this paper is an extension of the use of Polya's theorem for solving polynomial optimization problems in a parallel computing environment.

The goal of this project is to create algorithms which explicitly map computation, communication, and storage to existing parallel processing architectures. This goal is motivated by the failure of existing general-purpose semidefinite programming (SDP) solvers to efficiently utilize platforms for large-scale computation. Specifically, it is well established that linear programming and semi-definite programming both belong to the complexity class P-Complete, also known as the class of inherently sequential problems. Although there have been several attempts to map certain SDP solvers to a parallel computing environment [27], [28], certain critical steps cannot be distributed. The result is that as the number of processors increases, certain bottleneck computations dominate, leading a saturation in computational speed of these solvers (Amdahl's law [29]). We avoid these bottleneck computations and communications by exploiting the particular structure of the LMI conditions associated with Polya's theorem. Note that, in principle, a perfectly designed general-purpose SDP algorithm could identify the structure of the SDP, as we have, and map the communication, computation, and memory constraints to the parallel architecture. Indeed, there has been a great deal of research on creating programming languages which attempt to do just this [30], [31]. Presently, however, such languages are mostly theoretical and have certainly not been incorporated into existing SDP solvers.

In addition to parallel SDP solvers, there have been some efforts to exploit structure in certain polynomial optimization algorithms to reduce the size and complexity of the resulting LMIs. For example, in [32], symmetry was used to reduce the size of the SDP variables. A specific sparsity structure was used in [33]–[35] to reduce the complexity of the linear algebra calculations. Generalized approaches to the use of sparsity in SDP algorithms can be found in [34]. Groebner basis techniques [36], [37] have been used by [33] to simplify the formulation of the SDPs associated with the SOS decomposition problems.

This paper is organized around two independent problems: setting up the sequence of structured SDPs associated with Polya's theorem and solving them. Note that the problem of decentralizing the setup algorithm is significant in that for large-scale systems, the instantiation of the problem may be beyond the memory and computational capacity of a single processing node. For the setup problem, the algorithm that we propose has no centralized memory or computational requirements whatsoever. Furthermore, if a sufficient number of processors are available, the number of messages does not change with the size of the state space or the number of Polya's iterations. In addition, the ideal communication architecture for the set-up algorithm does not correspond to the communication structure of GPU computing or supercomputing. In the second problem, we propose a variant of a standard SDP primal-dual algorithm and map the computational, memory, and communication requirements to a parallel computing environment. Unlike the setup algorithm, the primal-dual algorithm does have a small centralized component corresponding to the update of the set of dual variables. However, we have structured the algorithm so that the size of this dual computation is solely a function of the degree of the polynomial QITS Lyapunov function and does not depend on the number of Polya's iterations, meaning that the sequence of algorithms has fixed centralized computational and communication complexity. In addition, there is no communication between processors, which means that the algorithm is well suited to most parallel computing architectures. A graph representation of the communication architecture of the setup and SDP algorithms has also been provided in the relevant sections.

Combining the setup and SDP components and testing the results of both in cluster computing environments, we demonstrate the capability of robust analysis and control of systems with $100+$ states and several uncertain parameters. Specifically, we ran a series of numerical experiments using a local Linux cluster and the Blue Gene supercomputer (with 200 processor allocation). First, we applied the algorithm to a current problem in robust stability analysis of magnetic confinement fusion using a discretized PDE model. Next, we examine the accuracy of the algorithm as Polya's iterations progress and compare this accuracy with the SOS approach. We show that unlike the general-purpose parallel SDP solver SDPARA [28], the speed-up—the increase in processing speed per additional processor—of our algorithm shows no evidence of saturation. Finally, we calculate the envelope of the algorithm on the Linux cluster in terms of the maximum state-space dimension, number of processors, and Polya's iterations.

Notation

We represent l-variate monomials as $\alpha^\gamma = \prod_{i=1}^{l} \alpha_i^{\gamma_i}$, where $\alpha \in \mathbb{R}^l$ is the vector of variables and $\gamma \in \mathbb{N}^l$ is the vector of exponents and $\sum_{i=1}^{l} \gamma_i = d$ is the degree of the monomial. We

define $W_d := \left\{\gamma \in \mathbb{N}^l : \sum_{i=1}^{l} \gamma_i = d\right\}$ as the totally ordered set of the exponents of l-variate monomials of degree d, where the ordering is lexicographic. In lexicographical ordering $\gamma \in W_d$ precedes $\eta \in W_d$, if the leftmost nonzero entry of $\gamma - \eta$ is positive. The lexicographical index of every $\gamma \in W_d$ can be calculated using the map $\langle\cdot\rangle : \mathbb{N}^l \to \mathbb{N}$ defined as [38]

$$\langle\gamma\rangle = \sum_{j=1}^{l-1}\sum_{i=1}^{\gamma_i} f\Big(l-j, d+1-\sum_{k=1}^{j-1}\gamma_k - i\Big) + 1, \quad (1)$$

whereas in [39]

$$f(l,d) := \begin{cases} 0 & \text{for } l = 0 \\ \dbinom{l+d-1}{l-1} = \dfrac{(d+l-1)!}{d!(l-1)!} & \text{for } l > 0, \end{cases} \quad (2)$$

is the cardinality of W_d, that is, the number of l-variate monomials of degree d. For convenience, we also define the index of a monomial α^γ to be $\langle\gamma\rangle$. We represent l-variate homogeneous polynomials of degree d_p as

$$P(\alpha) = \sum_{\gamma \in W_{d_p}} P_{\langle\gamma\rangle}\alpha^\gamma, \quad (3)$$

where $P_{\langle\gamma\rangle} \in \mathbb{R}^{n\times n}$ is the matrix coefficient of the monomial α^γ. We denote the element corresponding to the ith row and jth column of matrix A as $[A]_{i,j}$. The subspace of symmetric matrices in $\mathbb{R}^{n\times n}$ is denoted by $\mathbb{S}^n$. We define a basis for $\mathbb{S}^n$ as

$$[E_k]_{i,j} := \begin{cases} 1 & \text{if } i = j = k \\ 0 & \text{otherwise} \end{cases}, \quad \text{for } k \le n \quad \text{and}$$
$$[E_k]_{i,j} := [F_k]_{i,j} + [F_k]_{i,j}^T, \quad \text{for } k > n, \quad (4)$$

where

$$[F_k]_{i,j} := \begin{cases} 1 & \text{if } i = j-1 = k-n \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Note that this choice of basis is arbitrary—any other basis could be used. However, any change in basis would require modifications to the formulae defined in this paper. The canonical basis for $\mathbb{R}^n$ is denoted by e_i for $i = 1, \ldots, n$, where $e_i = [0 \cdots 0 \overbrace{1}^{i\text{th}} 0 \cdots 0]$. The vector with all entries equal to one is denoted by $\vec{1}$. The trace of $A \in \mathbb{R}^{n\times n}$ is denoted by $tr(A) = \sum_{i=1}^{n}[A]_{i,i}$. The block-diagonal matrix with diagonal blocks $X_1, \ldots, X_m \in \mathbb{R}^{n\times n}$ is denoted as $\mathrm{diag}(X_1, \ldots, X_m) \in \mathbb{R}^{mn\times mn}$ or occasionally as $\mathrm{diag}(X_i|_{i=1}^m) \in \mathbb{R}^{mn\times mn}$. The identity and zero matrices are denoted by $I_n \in \mathbb{R}^{n\times n}$ and $0_n \in \mathbb{R}^{n\times n}$.

II. Preliminaries

Consider the linear system

$$\dot{x}(t) = A(\alpha)x(t), \quad (6)$$

where $A(\alpha) \in \mathbb{R}^{n\times n}$ and $\alpha \in Q \subset \mathbb{R}^l$ is a vector of uncertain parameters. In this paper, we consider the case where $A(\alpha)$ is a homogeneous polynomial and $Q = \Delta_l \subset \mathbb{R}^l$, where Δ_l is the unit simplex, i.e.,

$$\Delta_l = \left\{\alpha \in \mathbb{R}^l, \sum_{i=1}^{l}\alpha_i = 1, \alpha_i \geqslant 0\right\}. \quad (7)$$

If $A(\alpha)$ is not homogeneous, we can obtain an equivalent homogeneous representation in the following manner. Suppose $A(\alpha)$ is a nonhomogeneous polynomial with $\alpha \in \Delta_l$, is of degree d_a, and has N_a monomials with nonzero coefficients. Define $D = \left(d_{a_1}, \ldots, d_{a_{N_a}}\right)$, where d_{a_i} is the degree of the ith monomial of $A(\alpha)$ according to lexicographical ordering. Now define the polynomial $B(\alpha)$ as follows.

1) Let $B = A$.
2) For $i = 1, \ldots, N_a$, multiply the ith monomial of $B(\alpha)$, according to lexicographical ordering, by $\left(\sum_{j=1}^{l}\alpha_j\right)^{d_a - d_{a_i}}$.

Then, since $\sum_{j=1}^{l}\alpha_j = 1$, $B(\alpha) = A(\alpha)$ for all $\alpha \in \Delta_l$ and, hence, all properties of $\dot{x}(t) = A(\alpha)x(t)$ are retained by the homogeneous system $\dot{x}(t) = B(\alpha)x(t)$.

1) Example: Construction of the Homogeneous System $\dot{x}(t) = B(\alpha)x(t)$: Consider the nonhomogeneous polynomial $A(\alpha) = C\alpha_1^2 + D\alpha_2 + E\alpha_3 + F$ of degree $d_a = 2$, where $[\alpha_1, \alpha_2, \alpha_3] \in \Delta_3$. Using the aforementioned procedure, the homogeneous polynomial $B(\alpha)$ can be constructed as

$$\begin{aligned} B(\alpha) &= C\alpha_1^2 + D\alpha_2(\alpha_1+\alpha_2+\alpha_3) + E\alpha_3(\alpha_1+\alpha_2+\alpha_3) \\ &\quad + F(\alpha_1+\alpha_2+\alpha_3)^2 = \underbrace{(C+F)}_{B_1}\alpha_1^2 + \underbrace{(D+2F)}_{B_2}\alpha_1\alpha_2 \\ &\quad + \underbrace{(E+2F)}_{B_3}\alpha_1\alpha_3 + \underbrace{(D+F)}_{B_4}\alpha_2^2 + \underbrace{(D+E+2F)}_{B_5}\alpha_2\alpha_3 \\ &\quad + \underbrace{(E+F)}_{B_6}\alpha_3^2 = \sum_{\gamma\in W_2} B_{\langle\gamma\rangle}\alpha^\gamma. \end{aligned} \quad (8)$$

The following is a stability condition [25] for system (6).

Theorem 1: System (6) is stable if and only if there exists a polynomial matrix $P(\alpha)$ such that $P(\alpha) \succ 0$ and

$$A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0 \quad (9)$$

for all $\alpha \in \Delta_l$.

A similar condition also holds for discrete-time linear systems. The conditions associated with Theorem 1 are infinite-dimensional LMIs, meaning they must hold at an infinite number of points. Such problems are known to be NP-hard [17]. In this paper, we derive a sequence of polynomial-time algorithms such that their outputs converge to the solution of the infinite-dimensional LMI. Key to this result is Polya's Theorem [40]. A variation of this theorem for matrices is given as follows.

Theorem 2: (Polya's Theorem): The homogeneous polynomial $F(\alpha) \succ 0$ for all $\alpha \in \Delta_l$ if and only if for all sufficiently large d

$$\left(\sum_{i=1}^{l}\alpha_i\right)^d F(\alpha) \quad (10)$$

has all positive definite coefficients.

Upper bounds for Polya's exponent d can be found as in [41]. However, these bounds are based on the properties of F and are difficult to determine *apriori*. In this paper, we show that applying Polya's theorem to the robust stability problem (i.e., the inequalities in Theorem 1) yields a semi-definite programming condition with an efficiently distributable structure. This is discussed in Section IV.

III. Problem Setup

In this section, we show how Polya's theorem can be used to determine the robust stability of an uncertain system using linear matrix inequalities with a distributable structure.

A. Polya's Algorithm

We consider the stability of the system described by (6). We are interested in finding a $P(\alpha)$ which satisfies the conditions of Theorem 1 . According to Polya's theorem, the constraints of Theorem 1 are satisfied if for some sufficiently large d_1 and d_2, the polynomials

$$\left(\sum_{i=1}^{l}\alpha_i\right)^{d_1} P(\alpha) \qquad \text{and} \tag{11}$$

$$-\left(\sum_{i=1}^{l}\alpha_i\right)^{d_2} \left(A^T(\alpha)P(\alpha)+P(\alpha)A(\alpha)\right) \tag{12}$$

have all positive definite coefficients.

Let $P(\alpha)$ be a homogeneous polynomial of degree d_p which can be represented as

$$P(\alpha)=\sum_{\gamma\in W_{d_p}} P_{\langle\gamma\rangle}\alpha^{\gamma}, \tag{13}$$

where the coefficients $P_{\langle\gamma\rangle}\in\mathbb{S}^n$ and where we recall that $W_{d_p}:=\left\{\gamma\in\mathbb{N}^l:\sum_{i=1}^{l}\gamma_i=d_p\right\}$ is the set of the exponents of all l-variate monomials of degree d_p. Since $A(\alpha)$ is a homogeneous polynomial of degree d_a, we can write it as

$$A(\alpha)=\sum_{\gamma\in W_{d_a}} A_{\langle\gamma\rangle}\alpha^{\gamma}, \tag{14}$$

where the coefficients $A_{\langle\gamma\rangle}\in\mathbb{R}^{n\times n}$. By substituting (13) and (14) into (11) and (12) and defining d_{pa} as the degree of $P(\alpha)A(\alpha)$, the conditions of Theorem 2 can be represented in the form

$$\sum_{h\in W_{d_p}} \beta_{\langle h\rangle,\langle\gamma\rangle}P_{\langle h\rangle}\succ 0;\quad \gamma\in W_{d_p+d_1} \text{ and} \tag{15}$$

$$\sum_{h\in W_{d_p}} \left(H^T_{\langle h\rangle,\langle\gamma\rangle}P_{\langle h\rangle}+P_{\langle h\rangle}H_{\langle h\rangle,\langle\gamma\rangle}\right)\prec 0;\ \gamma\in W_{d_{pa}+d_2}. \tag{16}$$

Here, $\beta_{\langle h\rangle,\langle\gamma\rangle}$ is defined to be the scalar coefficient which multiplies $P_{\langle h\rangle}$ in the $\langle\gamma\rangle$th monomial of the homogeneous polynomial $\left(\sum_{i=1}^{l}\alpha_i\right)^{d_1}P(\alpha)$ using the lexicographical ordering. Likewise, $H_{\langle h\rangle,\langle\gamma\rangle}\in\mathbb{R}^{n\times n}$ is the term whose left or right multiplies $P_{\langle h\rangle}$ in the $\langle\gamma\rangle$th monomial of $\left(\sum_{i=1}^{l}\alpha_i\right)^{d_2}\left(A^T(\alpha)P(\alpha)+P(\alpha)A(\alpha)\right)$ using lexicographical ordering. For an intuitive explanation as to how these β and H terms are calculated, we consider a simple example. Precise formulae for these terms will follow the example.

1) Example: Calculating the β and H Coefficients: Consider $A(\alpha)=A_1\alpha_1+A_2\alpha_2$ and $P(\alpha)=P_1\alpha_1+P_2\alpha_2$. By expanding (11) for $d_1=1$, we have

$$(\alpha_1+\alpha_2)P(\alpha)=P_1\alpha_1^2+(P_1+P_2)\alpha_1\alpha_2+P_2\alpha_2^2. \tag{17}$$

The $\beta_{\langle h\rangle,\langle\gamma\rangle}$ terms are then extracted as

$$\beta_{1,1}=1,\ \beta_{2,1}=0,\ \beta_{1,2}=1,\ \beta_{2,2}=1,\ \beta_{1,3}=0,\ \beta_{2,3}=1. \tag{18}$$

Next, by expanding (12) for $d_2=1$, we have

$$\begin{aligned}&(\alpha_1+\alpha_2)\left(A^T(\alpha)P(\alpha)+P(\alpha)A(\alpha)\right)\\&=\left(A_1^TP_1+P_1A_1\right)\alpha_1^3\\&+\left(A_1^TP_1+P_1A_1+A_2^TP_1+P_1A_2+A_1^TP_2+P_2A_1\right)\alpha_1^2\alpha_2\\&+\left(A_2^TP_1+P_1A_2+A_1^TP_2+P_2A_1+A_2^TP_2+P_2A_2\right)\alpha_1\alpha_2^2\\&+\left(A_2^TP_2+P_2A_2\right)\alpha_2^3.\end{aligned} \tag{19}$$

The $H_{\langle h\rangle,\langle\gamma\rangle}$ terms are then extracted as

$$\begin{aligned}&H_{1,1}=A_1,\quad H_{2,1}=\mathbf{0},\quad H_{1,2}=A_1+A_2,\quad H_{2,2}=A_1,\\&H_{1,3}=A_2,\quad H_{2,3}=A_1+A_2,\quad H_{1,4}=\mathbf{0},\quad H_{2,4}=A_2.\end{aligned} \tag{20}$$

2) General Formula: The $\{\beta_{\langle h\rangle,\langle\gamma\rangle}\}$ can be formally defined recursively as follows. Let the initial values for $\beta_{\langle h\rangle,\langle\gamma\rangle}$ be defined as

$$\beta^{(0)}_{\langle h\rangle,\langle\gamma\rangle}=\begin{cases}1 & \text{if } h=\gamma\\ 0 & \text{otherwise}\end{cases}\qquad \text{for } \gamma\in W_{d_p} \text{ and } h\in W_{d_p}. \tag{21}$$

Then, iterating for $i=1,\dots d_1$, we let

$$\beta^{(i)}_{\langle h\rangle,\langle\gamma\rangle}=\sum_{\lambda\in W_1}\beta^{(i-1)}_{\langle h\rangle,\langle\gamma-\lambda\rangle}\quad \text{for } \gamma\in W_{d_p+i} \text{ and } h\in W_{d_p}. \tag{22}$$

Finally, we set $\{\beta_{\langle h\rangle,\langle\gamma\rangle}\}=\{\beta^{d_1}_{\langle h\rangle,\langle\gamma\rangle}\}$. To obtain $\{H_{\langle h\rangle,\langle\gamma\rangle}\}$, set the initial values as

$$H^{(0)}_{\langle h\rangle,\langle\gamma\rangle}=\sum_{\lambda\in W_{d_a}:\lambda+h=\gamma}A_{\langle\lambda\rangle}\ \text{ for } \gamma\in W_{d_p+d_a} \text{ and } h\in W_{d_p}. \tag{23}$$

Then, iterating for $i=1,\dots d_2$, we let

$$H^{(i)}_{\langle h\rangle,\langle\gamma\rangle}=\sum_{\lambda\in W_1}H^{(i-1)}_{\langle h\rangle,\langle\gamma-\lambda\rangle}\quad \text{for } \gamma\in W_{d_{pa}+i} \text{ and } h\in W_{d_p}. \tag{24}$$

Finally, set $\{H_{\langle h\rangle,\langle\gamma\rangle}\}=\{H^{d_2}_{\langle h\rangle,\langle\gamma\rangle}\}$.

For the case of large-scale systems, computing and storing $\{\beta_{\langle h\rangle,\langle\gamma\rangle}\}$ and $\{H_{\langle h\rangle,\langle\gamma\rangle}\}$ is a significant challenge due to the number of these coefficients. Specifically, the number of terms increases with l (number of uncertain parameters in system (6)), d_p (degree of $P(\alpha)$), d_{pa} (degree of $P(\alpha)A(\alpha)$) and d_1,d_2 (Polya's exponents) as follows.

3) Number of $\beta_{\langle h\rangle,\langle\gamma\rangle}$ Coefficients: For given l,d_p and d_1, since $h\in W_{d_p}$ and $\gamma\in W_{d_p+d_1}$, the number of $\beta_{\langle h\rangle,\langle\gamma\rangle}$ coefficients is the product of $L_0:=\text{card}(W_{d_p})$ and $L:=\text{card}(W_{d_p+d_1})$. Recall that card (W_{d_p}) is the number of all

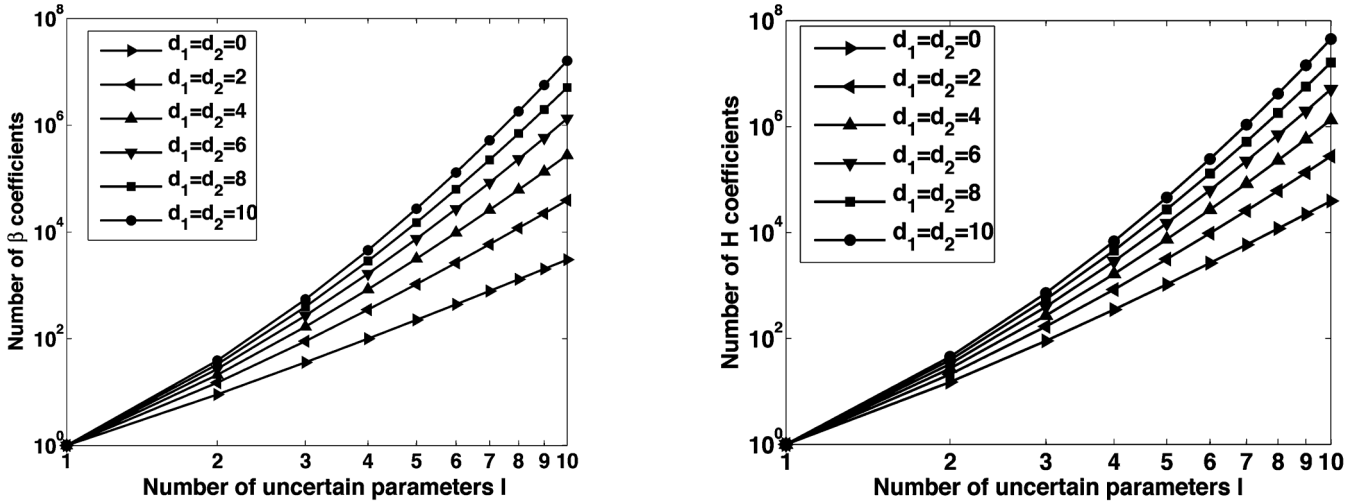


Fig. 1. Number of $\beta_{\langle h\rangle,\langle\gamma\rangle}$ and $H_{\langle h\rangle,\langle\gamma\rangle}$ coefficients versus the number of uncertain parameters for different Polya's exponents and for $d_p = d_a = 2$.

l-variate monomials of degree d_p and can be calculated using (2) as follows:

$$L_0 = f(l, d_p) = \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_p + l - 1}{l - 1} = \dfrac{(d_p + l - 1)!}{d_p!(l-1)!} & \text{for } l > 0. \end{cases} \tag{25}$$

Likewise, $\mathrm{card}(W_{d_p+d_1})$, that is, the number of all l-variate monomials of degree $d_p + d_1$ is calculated using (2) as follows:

$$\begin{aligned} L &= f(l, d_p + d_1) \\ &= \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_p+d_1+l-1}{l-1} = \dfrac{(d_p+d_1+l-1)!}{(d_p+d_1)!(l-1)!} & \text{for } l > 0. \end{cases} \end{aligned} \tag{26}$$

The number of $\beta_{\langle h\rangle,\langle\gamma\rangle}$ coefficients is $L_0 \cdot L$.

4) Number of $H_{\langle h\rangle,\langle\gamma\rangle}$ Coefficients: For given l, d_p, d_a and d_2, since $h \in W_{d_p}$ and $\gamma \in W_{d_{pa}+d_2}$, the number of $H_{\langle h\rangle,\langle\gamma\rangle}$ coefficients is the product of $L_0 := \mathrm{card}(W_{d_p})$ and $M := \mathrm{card}(W_{d_{pa}+d_2})$. By using (2), we have

$$\begin{aligned} M &= f(l, d_{pa} + d_2) \\ &= \begin{cases} 0 & \text{for } l = 0 \\ \binom{d_{pa}+d_2+l-1}{l-1} = \dfrac{(d_{pa}+d_2+l-1)!}{(d_{pa}+d_2)!(l-1)!} & \text{for } l > 0. \end{cases} \end{aligned} \tag{27}$$

The number of $H_{\langle h\rangle,\langle\gamma\rangle}$ coefficients is $L_0 \cdot M$.

The number of $\beta_{\langle h\rangle,\langle\gamma\rangle}$ and $H_{\langle h\rangle,\langle\gamma\rangle}$ coefficients and the required memory to store these coefficients are shown in Figs. 1 and 2 in terms of the number of uncertain parameters l and for different Polya's exponents. In all cases, $d_p = d_a = 2$.

It is observed from Fig. 2 that, even for small d_p and d_a, the required memory is in the Terabyte range. In [38], we proposed a decentralized computing approach to the calculation of $\{\beta_{\langle h\rangle,\langle\gamma\rangle}\}$ on large cluster computers. In this paper, we extend this method to the calculation of $\{H_{\langle h\rangle,\langle\gamma\rangle}\}$ and the SDP elements which will be discussed in Section V. We express the LMIs associated with conditions (15) and (16) as an SDP in primal and dual forms. We also discuss the structure of the primal and dual SDP variables and the constraints.

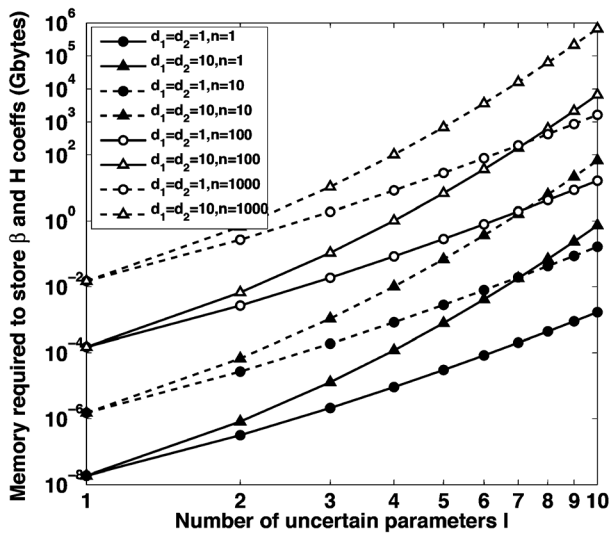


Fig. 2. Memory required to store β and H coefficients versus the number of uncertain parameters, for different d_1, d_2 and $d_p = d_a = 2$.

B. SDP Problem Elements

A semi-definite programming problem can be stated either in primal or dual format. Given $C \in \mathbb{S}^m$, $a \in \mathbb{R}^K$, and $B_i \in \mathbb{S}^m$, the **primal problem** is of the form

$$\begin{aligned} \max_X \quad & tr(CX) \\ \text{subject to} \quad & a - B(X) = 0 \\ & X \succeq 0, \end{aligned} \tag{28}$$

where the linear operator $B : \mathbb{S}^m \to \mathbb{R}^K$ is defined as

$$B(X) = \begin{bmatrix} tr(B_1X) & tr(B_2X) & \cdots & tr(B_KX) \end{bmatrix}^T. \tag{29}$$

$X \in \mathbb{S}^m$ is the primal variable. Given a primal SDP, the associated **dual problem** is

$$\begin{aligned} \min_{y,Z} \quad & a^T y \\ \text{subject to} \quad & B^T(y) - C = Z \\ & Z \succeq 0 \ , \ y \in \mathbb{R}^K, \end{aligned} \tag{30}$$

where $B^T : \mathbb{R}^K \to \mathbb{S}^m$ is the transpose operator and is given by

$$B^T(y) = \sum_{i=1}^{K} y_i B_i \tag{31}$$

and where $y \in \mathbb{R}^K$ and $Z \in \mathbb{S}^m$ are the dual variables. The elements C, B_i, and a of the SDP problem associated with the LMIs in (15) and (16) are defined as follows. We define the element C as

$$C := \operatorname{diag}(C_1, \dots C_L, C_{L+1}, \dots C_{L+M}), \tag{32}$$

where

$$C_i := \begin{cases} \delta I_n \cdot \left(\sum_{h \in W_{d_p}} \beta_{\langle h \rangle, i} \frac{d_p!}{h_1! \dots h_l!} \right), & 1 \le i \le L \\ 0_n, & L+1 \le i \le L+M. \end{cases} \tag{33}$$

Recall that $L = \operatorname{card}(W_{d_p+d_1})$ is the number of monomials in $\left(\sum_{i=1}^{l} \alpha_i\right)^{d_1} P(\alpha)$; $M = \operatorname{card}(W_{d_{pa}+d_2})$ is the number of monomials in $\left(\sum_{i=1}^{l} \alpha_i\right)^{d_2} P(\alpha)A(\alpha)$, where n is the dimension of system (6); l is the number of uncertain parameters; and δ is a small positive parameter.

For $i = 1, \dots, K$, define B_i elements as

$$B_i := \operatorname{diag}(B_{i,1}, \dots B_{i,L}, B_{i,L+1}, \dots B_{i,L+M}), \tag{34}$$

where K is the number of dual variables in (30) and is equal to the product of the number of upper-triangular elements in each $P_\gamma \in \mathbb{S}^n$ (the coefficients in $P(\alpha)$) and the number of coefficients in $P(\alpha)$ (i.e., the cardinality of W_{d_p}). Since there are $f(l, d_p) = \binom{d_p+l-1}{l-1}$ coefficients in $P(\alpha)$ and each coefficient has $\tilde{N} := (1/2)n(n+1)$ upper-triangular elements, we find

$$K = \frac{(d_p + l - 1)!}{d_p!(l-1)!} \tilde{N}. \tag{35}$$

To define the $B_{i,j}$ blocks, first, we define the function $V_{\langle h \rangle} : \mathbb{Z}^K \to \mathbb{Z}^{n \times n}$

$$V_{\langle h \rangle}(x) := \sum_{j=1}^{\tilde{N}} E_j \, x_{j+\tilde{N}(\langle h \rangle - 1)} \quad \text{for all} \quad h \in W_{d_p}, \tag{36}$$

which maps each variable to a basis matrix E_j, where we recall that E_j is the basis for $\mathbb{S}^n$. Note that a different choice of basis would require a different function $V_{\langle h \rangle}$. Then, for $i = 1, \dots, K$

$$B_{i,j} := \begin{cases} \sum_{h \in W_{d_p}} \beta_{\langle h \rangle, j} V_{\langle h \rangle}(e_i), & 1 \le j \le L \ (I) \\ -\sum_{h \in W_{d_p}} \Big(H^T_{\langle h \rangle, j-L} V_{\langle h \rangle}(e_i) \\ \quad + V_{\langle h \rangle}(e_i) H_{\langle h \rangle, j-L} \Big), & L+1 \le j \le L+M. \ (II). \end{cases} \tag{37}$$

Finally, to complete the SDP problem associated with Polya's algorithm set

$$a = \vec{1} \in \mathbb{R}^K. \tag{38}$$

C. Parallel Setup Algorithm

In this section, we propose a decentralized, iterative algorithm for calculating the terms $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$, $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$, C, and B_i as defined in (22), (24), (32), and (34). The algorithm has been implemented in C++, using message passing interface (MPI) and is available at: www.sites.google.com/a/asu.edu/kamyar/software. We present an abridged description of this algorithm in Algorithm 1, where N is the number of available processors.

Note that we have only addressed the problem of robust stability analysis, using the polynomial inequality

$$P(\alpha) \succ 0, A^T(\alpha)P(\alpha) + P(\alpha)A(\alpha) \prec 0$$

for $\alpha \in \Delta_l$. However, we can generalize the decentralized setup algorithm to consider a more general class of feasibility problems, i.e.,

$$\sum_{i=1}^{\hat{N}} \left(\tilde{A}_i(\alpha)\tilde{X}(\alpha)\tilde{B}_i(\alpha) + \tilde{B}_i^T(\alpha)\tilde{X}(\alpha)\tilde{A}_i^T(\alpha) + R_i(\alpha) \right) \prec 0 \tag{41}$$

for $\alpha \in \Delta_l$. One motivation behind the development of such a generalized setup algorithm is that the parameter-dependent versions of the LMIs associated with H_2 and H_∞ synthesis problems in [42] and [43] can be formulated in the form of (41).

D. Setup Algorithm: Complexity Analysis

Since checking the positive definiteness of all representatives of a square matrix with parameters on proper real intervals is intractable [7], the question of feasibility of (9) is also intractable. To solve the problem of inherent intractability, we establish a tradeoff between accuracy and complexity. In fact, we develop a sequence of decentralized polynomial-time algorithms whose solutions converge to the exact solution of the NP-hard problem. In other words, the translation of a polynomial optimization problem to an LMI problem is the main source of complexity. This high complexity is unavoidable and, in fact, is the reason we seek parallel algorithms.

Algorithm 1 distributes the computation and storage of $\{\beta_{\langle h \rangle, \langle \gamma \rangle}\}$ and $\{H_{\langle h \rangle, \langle \gamma \rangle}\}$ among the processors and their dedicated memories, respectively. In an ideal case, where the number of available processors is sufficiently large (equal to the number of monomials in $P(\alpha)A(\alpha)$, that is, M) only one monomial (L_0 of $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and L_0 of $H_{\langle h \rangle, \langle \gamma \rangle}$) are assigned to each processor.

1) Computational Complexity Analysis: The most computationally expensive part of the setup algorithm is the calculation of the $B_{i,j}$ blocks in (37). Considering that the cost of matrix-matrix multiplication is $\sim n^3$, the cost of calculating each $B_{i,j}$ block is $\sim \operatorname{card}(W_{d_p}) \cdot n^3$. According to (34) and (37), the total number of $B_{i,j}$ blocks is $K(L+M)$. Hence, per Algorithm 1, each processor processes K ($\texttt{floor}(L/N) + \texttt{floor}(M/N)$) of the $B_{i,j}$ blocks, where N is the number of available processors. Thus, the per processor computational cost of calculating the $B_{i,j}$ at each Polya's iteration is

$$\sim \operatorname{card}(W_{d_p}) \cdot n^3 \cdot K \left(\texttt{floor}\left(\frac{L}{N}\right) + \texttt{floor}\left(\frac{M}{N}\right) \right). \tag{42}$$

Algorithm 1: The parallel set-up algorithm

Inputs: d_p : degree of $P(\alpha)$, d_a : degree of $A(\alpha)$, n : number of states, l: number of uncertain parameters, d_1, d_2 : number of Polya's iterations, Coefficients of $A(\alpha)$.
Initialization: Set $\hat{d}_1 = \hat{d}_2 = 0$ and $d_{pa} = d_p + d_a$. Calculate L_0 as the number of monomials in $P(\alpha)$ using (25) and M as the number of monomials in $P(\alpha)A(\alpha)$ using (27). Set $L = L_0$. Calculate $L' = \text{floor}(\frac{L}{N})$ and $M' = \text{floor}\,(\frac{M}{N})$ as the number of monomials in $P(\alpha)$ and $P(\alpha)A(\alpha)$ assigned to each processor.
for $i = 1, \cdots, N$ *processor* i **do**
Initialize $\beta_{k,j}$ for $j = (i-1)L' + 1, \cdots, iL'$ and $k = 1, \cdots L_0$ using (21).
Initialize $H_{k,m}$ for $m = (i-1)M' + 1, \cdots, iM'$ and $k = 1, \cdots L_0$ using (23).
Calculating β and H coefficients:
while $\hat{d}_1 \leq d_1$ *or* $\hat{d}_2 \leq d_2$ **do**
if $\hat{d}_1 \leq d_1$ **then**
for $i = 1, \cdots, N$, *processor* i **do**
Set $d_p = d_p + 1$ and $\hat{d}_1 = \hat{d}_1 + 1$. Update L using (26). Update $L' = \text{floor}\,(\frac{L}{N})$.
Calculate $\beta_{k,j}$ for $j = (i-1)L' + 1, \cdots, iL'$ and $k = 1, \cdots L_0$ using (22).
if $\hat{d}_2 \leq d_2$ **then**
for $i = 1, \cdots, N$, *processor* i **do**
Set $d_{pa} = d_{pa} + 1$ and $\hat{d}_2 = \hat{d}_2 + 1$. Update M using (27). Update $M' = \text{floor}(\frac{M}{N})$.
Calculate $H_{k,m}$ for $m = (i-1)M' + 1, \cdots, iM'$ and $k = 1, \cdots L_0$ using (24).
Calculating the SDP elements:
for $i = 1, \cdots, N$, *processor* i **do**
Calculate the number of dual variables K using (35).
Set $T' = \text{floor}(\frac{L+M}{N})$.
Calculate the blocks of the SDP element C as

$$\begin{cases} C_j \text{ using (33)} & \text{for } j = (i-1)L' + 1, \cdots, iL' \\ C_j = 0_n & \text{for } j = L + (i-1)M' + 1, \cdots, L + iM' \end{cases}$$

Set the sub-blocks of the SDP element C as

$$\bar{\mathbf{C}}_i = \text{diag}\big(C_k|_{k=(i-1)T'+1}^{iT'}\big). \tag{39}$$

for $j = 1, \cdots K$ **do**
Calculate the blocks of the SDP elements B_j as

$$\begin{cases} B_{j,k} \text{ using (37)-}I & \text{for } k = (i-1)L' + 1, \cdots, iL' \\ B_{j,k} \text{ using (37)-}II & \text{for } k = L + (i-1)M' + 1, \\ & \qquad \cdots, L + iM' \end{cases}$$

Set the sub-blocks of the SDP element B_j as

$$\bar{\mathbf{B}}_{j,i} = \text{diag}\left(B_{j,k}|_{k=(i-1)T'+1}^{iT'}\right). \tag{40}$$

Outputs: Sub-blocks $\bar{\mathbf{C}}_i$ and $\bar{\mathbf{B}}_{j,i}$ of the SDP elements for $i = 1, \cdots, N$ and $j = 1, \cdots, K$.

By substituting for K from (35), card (W_{d_p}) from (25), L from (26), and M from (27), the per processor computation cost at each iteration is

$$\sim \left(\frac{(d_p + l - 1)!}{d_p!(l-1)!}\right)^2 \frac{n^4}{2(n+1)} \left(\texttt{floor}\left(\frac{\frac{(d_p+d_1+l-1)!}{(d_p+d_1)!(l-1)!}}{N}\right) + \texttt{floor}\left(\frac{\frac{(d_{pa}+d_2+l-1)!}{(d_{pa}+d_2)!(l-1)!}}{N}\right)\right) \tag{43}$$

assuming that $l > 0$ and $N \leq M$. For example, for the case of large-scale systems (large n and l), the computation cost per processor at each iteration is $\sim(l^{2d_p+d_1} + l^{2d_p+d_a+d_2})n^5$ having $N = L_0$ processors, $\sim(l^{2d_p+d_1} + l^{2d_p+d_a+d_2})n^5$ having $N = L$ processors, and $\sim l^{2d_p+d_a+d_2-d_1}n^5$ having $N = M$ processors. Thus, for the case where $d_p \geq 3$, the number of operations grows more slowly in n than in l.

2) Communication Complexity Analysis: Communication between processors can be modeled by a directed graph $G(V, E)$, where the set of nodes $V = \{1, \ldots, N\}$ is the set of indices of the available processors and the set of edges $E = \{(i, j) : i, j \in V\}$ is the set of all pairs of processors that communicate with each other. For every directed graph, we can define an adjacency matrix T_G. If processor i communicates with processor j, then $[T_G]_{i,j} = 1$; otherwise, $[T_G]_{i,j} = 0$. In this section, we only define the adjacency matrix for the part of the algorithm that performs Polya's iterations on $P(\alpha)$. For Polya's iterations on $P(\alpha)A(\alpha)$, the adjacency matrix can be defined in a similar manner. For simplicity, we assume that at each iteration, the number of available processors is equal to the number of monomials in $(\sum_{i=1}^{l} \alpha_i)^{d_1} P(\alpha)$. Using (26), let us define r_{d_1} and r_{d_1+1} as the numbers of monomials in $(\sum_{i=1}^{l} \alpha_i)^{d_1} P(\alpha)$ and $(\sum_{i=1}^{l} \alpha_i)^{d_1+1} P(\alpha)$. For $I = 1, \ldots, r_{d_1}$, define

$$\mathcal{E}_I := \{\text{lex. indices of monomials in } \left(\sum_{i=1}^{l} \alpha_i\right) \alpha^\gamma : \gamma \in W_{d_p+d_1} \text{ and } \langle\gamma\rangle = I\}.$$

Then, for $i = 1, \ldots, r_{d_1+1}$ and $j = 1, \ldots, r_{d_1+1}$

$$[T_G]_{i,j} := \begin{cases} 1, & \text{if } i \leq r_{d_1} \text{ and } j \in \mathcal{E}_i \text{ and } i \neq j \\ 0, & \text{otherwise.} \end{cases}$$

Note that this definition implies that the communication graph of the setup algorithm changes at every iteration. To help visualize the graph, the adjacency matrix for the case where $\alpha \in \Delta_2$ is

$$T_G := \left[\begin{array}{cccccc|ccc} 0 & 1 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 & & & \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & & \ddots & \ddots & 0 & & & \\ 0 & 0 & \cdots & \cdots & 0 & 1 & 0 & \cdots & 0 \\ \hline 0 & & & \cdots & & 0 & 0 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots & \vdots & \ddots & \vdots \\ 0 & & & \cdots & & 0 & 0 & \cdots & 0 \end{array}\right] \in \mathbb{R}^{r_{d_1+1} \times r_{d_1+1}},$$

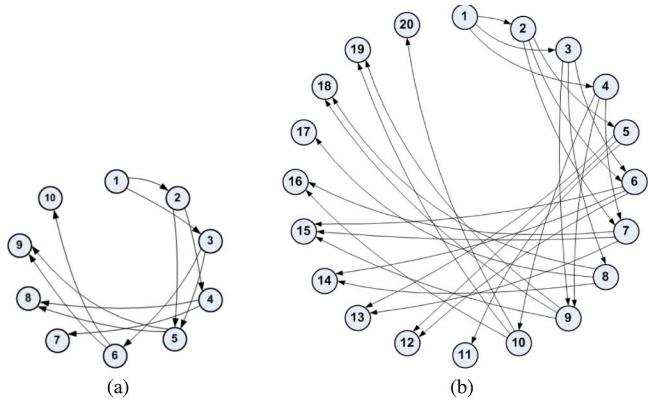


Fig. 3. Graph representation of the network communication of the setup algorithm. (a) Communication-directed graph for the case $\alpha \in \Delta_3$, $d_p = 2$. (b) Communication-directed graph for the case $\alpha \in \Delta_4$, $d_p = 2$.

where the nonzero sub-block of T_G lies in $\mathbb{R}^{r_{d_1} \times r_{d_1}}$. We can also illustrate the communication graphs for the cases $\alpha \in \Delta_3$ and $\alpha \in \Delta_4$ with $d_p = 2$ as seen in Fig. 3(a) and (b).

For a given algorithm, the communication complexity is defined as the sum of the size of all communicated messages. For simplicity, let us consider the worst case scenario, where each processor is assigned more than one monomial and sends all of its assigned $\beta_{\langle h \rangle, \langle \gamma \rangle}$ and $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients to other processors. In this case, the algorithm assigns $\texttt{floor}(L/N) \cdot \text{card}(W_{d_p})$ of the $\beta_{\langle h \rangle, \langle \gamma \rangle}$ coefficients, each of size 1, and $(\texttt{floor}(L/N) + \texttt{floor}(M/N)) \cdot \text{card}(W_{d_p})$ of the $H_{\langle h \rangle, \langle \gamma \rangle}$ coefficients, each of size n^2, to each processor. Thus, the communication complexity of the algorithm per processor and per iteration is

$$\text{card}(W_{d_p}) \left(\texttt{floor}\left(\frac{L}{N}\right) + \texttt{floor}\left(\frac{M}{N}\right) n^2 \right). \tag{44}$$

This indicates that increasing the number of processors (up to M) actually leads to less communication overhead per processor and improves the scalability of the algorithm. By substituting for card (W_{d_p}) from (25), L from (26), and M from (27), and considering large l and n, the communication complexity per processor at each Polya's iteration is $\sim l^{d_{pa}+d_2} n^2$ having $N = L_0$ processors, $\sim l^{d_{pa}+d_2-d_1} n^2$ having $N = L$ processors, and $\sim l^{d_p} n^2$ having $N = M$ processors.

IV. Parallel SDP Solver

In this section, we describe the steps of our primal-dual interior-point algorithm and show how, for the LMIs in (15) and (16), these steps can be distributed in a distributed-computing, distributed-memory environment.

A. Interior-Point Methods

Interior-point methods define a popular class of algorithms for solving linear and semidefinite programming problems. The most widely accepted interior-point algorithms are dual scaling [44], [45]; primal-dual [46]–[48]; and cutting-plane/spectral bundle [49]–[51]. In this paper, we use the central-path-following primal-dual algorithm described in [27] and [48]. Although we found it possible to use dual-scaling algorithms, we chose to pursue a primal-dual algorithm because, in general, primal-dual algorithms converge faster [45], [48] while still preserving the structure of the solution [see (59)] at each iteration. We prefer primal-dual to cutting plane/spectral bundle methods because, as we show in Section V-D, the centralized part of our primal-dual algorithm consists of solving a symmetric system of linear equations [see (80)], whereas for the cutting plane/spectral bundle algorithm, the centralized computation would consist of solving a constrained quadratic program (see [50] and [51]) with a number of variables equal to the size of the system of linear equations. Since centralized computation is the limiting factor in a parallel algorithm and because solving symmetric linear equations is simpler than solving a quadratic programming problem, we chose the primal-dual approach.

The choice of a central path-following primal-dual algorithm as in [48] and [52] was motivated by results in [53] which demonstrated better convergence, accuracy, and robustness over the other types of primal-dual algorithms. More specifically, we chose the approach in [48] over [52] because unlike the Schur complement matrix (SCM) approach of the algorithm in [52], the SCM of [48] is symmetric and only the upper-triangular elements need to be sent/received by the processors. This leads to less communication overhead. The other reason for choosing [48] is that the symmetric SCM of the algorithm in [48] can be factorized using Cholesky factorization, whereas the nonsymmetric SCM of [52] must be factorized by LU factorization (LU factorization is roughly twice as expensive as Cholesky factorization). Since factorization of SCM comprises the main portion of the centralized computation in our algorithm, it is crucial for us to use computationally cheaper factorization methods to achieve better scalability.

In the primal-dual algorithm, both primal and dual problems are solved by iteratively calculating primal and dual step directions and step sizes, and applying these to the primal and dual variables. Let X be the primal variable and y and Z be the dual variables. At each iteration, the variables are updated as

$$X_{k+1} = X_k + t_p \Delta X \tag{45}$$
$$y_{k+1} = y_k + t_d \Delta y \tag{46}$$
$$Z_{k+1} = Z_k + t_d \Delta Z, \tag{47}$$

where ΔX, Δy, and ΔZ are Newton's search direction and t_p and t_d are the primal and dual step sizes. We choose the step sizes using a standard line-search between 0 and 1 with the constraint that X_{k+1} and Z_{k+1} remain positive semidefinite. We use a Newton's search direction given by

$$\Delta X = \Delta \widehat{X} + \Delta \overline{X} \tag{48}$$
$$\Delta y = \Delta \widehat{y} + \Delta \overline{y} \tag{49}$$
$$\Delta Z = \Delta \widehat{Z} + \Delta \overline{Z}, \tag{50}$$

where $\Delta \widehat{X}$, $\Delta \widehat{y}$, and $\Delta \widehat{Z}$ are the predictor step directions and $\Delta \overline{X}$, $\Delta \overline{y}$, and $\Delta \overline{Z}$ are the corrector step directions. Per [48], the predictor step directions are found as

$$\Delta \widehat{y} = \Omega^{-1} \left(-a + B(Z^{-1} G X) \right) \tag{51}$$
$$\Delta \widehat{X} = -X + Z^{-1} G B^T(\Delta \widehat{y}) X \tag{52}$$
$$\Delta \widehat{Z} = B^T(y) - Z - C + B^T(\Delta \widehat{y}), \tag{53}$$

where C and the operators B and B^T are as defined in the previous section

$$G = -B^T(y) + Z + C \tag{54}$$

and

$$\Omega = \left[B(Z^{-1}B^T(e_1)X) \ \cdots \ B(Z^{-1}B^T(e_k)X)\right]. \tag{55}$$

Recall that $e_1, \ldots, e_k$ are the standard basis for $\mathbb{R}^k$. Once we have the predictor step directions, we can calculate the corrector step directions per [48]. Let $\mu = (1/3)tr(ZX)$. The corrector step directions are

$$\Delta\overline{y} = \Omega^{-1}\left(B(\mu Z^{-1}) - B(Z^{-1}\Delta\widehat{Z}\Delta\widehat{X})\right) \tag{56}$$

$$\Delta\overline{X} = \mu Z^{-1} - Z^{-1}\Delta\widehat{Z}\Delta\widehat{X} - Z^{-1}\Delta\overline{Z}X \tag{57}$$

$$\Delta\overline{Z} = B^T(\Delta\overline{y}). \tag{58}$$

The stopping criterion is $|a^T y - tr(CX)| \leq \epsilon$. Information regarding the selection of starting points and convergence of different variants of interior-point primal-dual algorithms, including the algorithm we use in this paper, are presented in [46]–[48].

B. Structure of SDP Variables

The key algorithmic insight of this paper, which allows us to use the primal-dual approach presented in [48], is that by choosing an initial value for the primal variable with a certain block structure corresponding to the distributed structure of the processors, the algorithm will preserve this structure at every iteration. Specifically, we define the following structured block-diagonal subspace where each block corresponds to a single processor.

$$S_{l,m,n} := \{Y \subset \mathbb{R}^{(l+m)n\times(l+m)n} : \\ Y = \mathrm{diag}(Y_1, \ldots Y_l, Y_{l+1}, \ldots Y_{l+m}) \text{ for } Y_i \in \mathbb{R}^{n\times n}\}. \tag{59}$$

According to the following theorem, the subspace $S_{l,m,n}$ is invariant under Newton's iteration in the sense that when the algorithm in [48] is applied to the SDP problem defined by the polynomial optimization problem with an initial value of the primal variable $X_0 \in S_{l,m,n}$, then the primal variable remains in the subspace at every Newton's iteration X_k.

Theorem 3: Consider the SDP problem defined in (28) and (30) with elements given by (32), (34), and (38). Suppose L and M are the cardinalities of $W_{d_p+d_1}$ and $W_{d_{pa}+d_2}$. If (45)–(47) are initialized by

$$X_0 \in S_{L,M,n}, \quad y_0 \in \mathbb{R}^K, \quad Z_0 \in S_{L,M,n}, \tag{60}$$

then for all $k \in \mathbb{N}$

$$X_k \in S_{L,M,n}, \quad Z_k \in S_{L,M,n}. \tag{61}$$

Proof: We proceed by induction. First, suppose for some $k \in \mathbb{N}$

$$X_k \in S_{L,M,n} \quad \text{and} \quad Z_k \in S_{L,M,n}. \tag{62}$$

We would like to show that this implies $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. To see this, observe that according to (45)

$$X_{k+1} = X_k + t_p\Delta X_k \quad \text{for all } k \in \mathbb{N}. \tag{63}$$

From (48), ΔX_k can be written as

$$\Delta X_k = \Delta\widehat{X}_k + \Delta\overline{X}_k \quad \text{for all } k \in \mathbb{N}. \tag{64}$$

To find the structure of ΔX_k, we focus on the structures of $\Delta\widehat{X}_k$ and $\Delta\overline{X}_k$ individually. Using (52), $\Delta\widehat{X}_k$ is

$$\Delta\widehat{X}_k = -X_k + Z_k^{-1}G_kB^T(\Delta\widehat{y}_k)X_k \quad \text{for all } k \in \mathbb{N} \tag{65}$$

where, according to (54), G_k is

$$G_k = C - B^T(y_k) + Z_k \quad \text{for all } k \in \mathbb{N}. \tag{66}$$

First, we examine the structure of G_k. According to the definition of C and B_i in (32) and (34), and the definition of $B^T(y)$ in (31), we know that

$$C \in S_{L,M,n}, \quad B^T : \mathbb{R}^K \mapsto S_{L,M,n}. \tag{67}$$

Since all terms on the right-hand side of (66) are in $S_{L,M,n}$ and $S_{L,M,n}$ is a subspace, we conclude

$$G_k \in S_{L,M,n}. \tag{68}$$

Returning to (65), using our assumption in (62) and noting that the structure of the matrices in $S_{L,M,n}$ is also preserved through multiplication and inversion, we conclude

$$\Delta\widehat{X}_k \in S_{L,M,n}. \tag{69}$$

Using (57), the second term in (64) is

$$\Delta\overline{X}_k = \mu Z_k^{-1} - Z_k^{-1}\Delta\widehat{Z}_k\Delta\widehat{X}_k \\ - Z_k^{-1}\Delta\overline{Z}_kX_k \quad \text{for all } k \in \mathbb{N}. \tag{70}$$

To determine the structure of $\Delta\overline{X}_k$, first we investigate the structure of $\Delta\widehat{Z}_k$ and $\Delta\overline{Z}_k$. According to (53) and (58), we have

$$\Delta\widehat{Z}_k = B^T(y_k) - Z_k - C \\ + B^T(\Delta\widehat{y}_k) \quad \text{for all } k \in \mathbb{N} \tag{71}$$

$$\Delta\overline{Z}_k = B^T(\Delta\overline{y}_k) \quad \text{for all } k \in \mathbb{N}. \tag{72}$$

Since all the terms in the right-hand side of (71) and (72) are in $S_{L,M,n}$, then

$$\Delta\widehat{Z}_k \in S_{L,M,n}, \quad \Delta\overline{Z}_k \in S_{L,M,n}. \tag{73}$$

Recalling (69), (70), and our assumption in (62), we have

$$\Delta\overline{X}_k \in S_{L,M,n}. \tag{74}$$

According to (69), (73), and (74), the total step directions are in $S_{L,M,n}$

$$\Delta X_k = \Delta\widehat{X}_k + \Delta\overline{X}_k \in S_{L,M,n} \tag{75}$$
$$\Delta Z_k = \Delta\widehat{Z}_k + \Delta\overline{Z}_k \in S_{L,M,n}, \tag{76}$$

and it follows that:

$$X_{k+1} = X_k + t_p\Delta X_k \in S_{L,M,n} \tag{77}$$
$$Z_{k+1} = Z_k + t_p\Delta Z_k \in S_{L,M,n}. \tag{78}$$

Thus, for any $y \in \mathbb{R}^K$ and $k \in \mathbb{N}$, if $X_k, Z_k \in S_{L,M,n}$, we have $X_{k+1}, Z_{k+1} \in S_{L,M,n}$. Since we have assumed that the initial values $X_0, Z_0 \in S_{L,M,n}$, we conclude by induction that $X_k \in S_{L,M,n}$ and $Z_k \in S_{L,M,n}$ for all $k \in \mathbb{N}$ ■

C. Parallel Implementation

In this section, a parallel algorithm for solving the SDP problems associated with Polya's algorithm is provided. We show how to map the block-diagonal structure of the primal variable and Newton updates described in Section V-A to a parallel computing structure consisting of a central root processor with N slave processors. Note that processor steps are simultaneous and transitions between root and processor steps are synchronous. Processors are idle when root is active and vice-versa. A C++ implementation of this algorithm, using MPI and Blas/Lapack libraries is provided at: www.sites.google.com/a/asu.edu/kamyar/software. Let N be the number of available processors and $J = \texttt{floor}\,(L + M/N)$. Per Algorithm 1, we assume processor i has access to the sub-blocks $\overline{\mathbf{C}}_i$ and $\overline{\mathbf{B}}_{j,i}$ defined in (39) and (40) for $j = 1, \ldots, K$. Be aware that minor parts of Algorithm 2 have been abridged in order to simplify the presentation.

Algorithm 2: The parallel SDP solver algorithm

Inputs: $\bar{\mathbf{C}}_i, \bar{\mathbf{B}}_{j,i}$ for $i = 1, \cdots, N$ and $j = 1, \cdots, K-$ the sub-blocks of the SDP elements provided to processor i by the set-up algorithm.

Processors Initialization step:

for $i = 1, \cdots, N, processor\, i$ **do**

Initialize primal and dual variables $\mathbf{X}_i^0$, $\mathbf{Z}_i^0$ and y^0 as

$$\mathbf{X}_i^0 = \begin{cases} I_{(J+1)n,} & 0 \le i \le L + M - NJ \\ I_{Jn,} & L + M - NJ \le i < N, \end{cases}'$$

$$\mathbf{Z}_i^0 = \mathbf{X}_i^0 \quad \text{and} \quad y^0 = \vec{0} \in \mathbb{R}^K,$$

Calculate the complementary slackness [49] $S_i = tr(\mathbf{Z}_i^0\mathbf{X}_1^0)$. Send S_i to processor root.

Root Initialization step:

Root processor **do**

Calculate the barrier parameter [49] $\mu = \frac{1}{3}\sum_{i=1}^N S_i$.

Set the SDP element $\alpha = \vec{1} \in \mathbb{R}^K$.

Processors step 1:

for $i = 1, \cdots, N, processor\, i$ **do**

for $k = 1, \cdots, K$ **do**

Calculate the elements of Ω_1 (R-H-S of (80))

$$\omega_{i,k} = tr\left(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}\left(-\sum_{j=1}^K y_j\bar{\mathbf{B}}_{j,i} + \mathbf{Z}_i + \bar{\mathbf{C}}_i\right)\mathbf{X}_i\right)$$

for $l = 1, \cdots, K$**do**

Calculate the elements of the SCM as

$$\lambda_{i,k,l} = tr(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}\bar{\mathbf{B}}_{l,i}\mathbf{X}_i) \tag{79}$$

Send $\omega_{i,k}$ and $\lambda_{i,k,l}, k = 1, \cdots, K$ and $l = 1, \cdots, K$ to root processor.

Root step 1:

Root processor **do**

Construct the R-H-S of (80) and the SCM as

$$\Omega_1 = \begin{pmatrix} \sum_{i=1}^N \omega_{i,1} \\ \sum_{i=1}^N \omega_{i,2} \\ \vdots \\ \sum_{i=1}^N \omega_{i,K} \end{pmatrix} - a \quad \text{and}$$

$$\Lambda = \left[\begin{pmatrix} \sum_{i=1}^N \lambda_{i,1,1} \\ \sum_{i=1}^N \lambda_{i,2,1} \\ \vdots \\ \sum_{i=1}^N \lambda_{i,K,1} \end{pmatrix}, \cdots, \begin{pmatrix} \sum_{i=1}^N \lambda_{i,1,K} \\ \sum_{i=1}^N \lambda_{i,2,K} \\ \vdots \\ \sum_{i=1}^N \lambda_i, K, K \end{pmatrix}\right]$$

Solve the following system of equations for the predictor dual step $\Delta\hat{y} \in \mathbb{R}^K$ and send $\Delta\hat{y}$ to all processors.

$$\Lambda\Delta\hat{y} = \Omega_1 \tag{80}$$

Processors step 2:

for $i = 1, \cdots, N\, processor\, i$ **do**

Calculate the predictor step directions

$$\Delta\hat{\mathbf{X}}_i = -\mathbf{X}_i + (\mathbf{Z}_i)^{-1}\left(-\sum_{j=1}^K y_j\bar{\mathbf{B}}_{j,i} + \mathbf{Z}_i + \bar{\mathbf{C}}_i\right)\sum_{j=1}^K \Delta\hat{y}_j\bar{\mathbf{B}}_{j,i}\mathbf{X}_i,$$

$$\Delta\hat{\mathbf{Z}}_i = \sum_{j=1}^K yj\bar{\mathbf{B}}_{j,i} - \mathbf{Z}_i - \bar{\mathbf{C}}_i + \sum_{j=1}^K \Delta\hat{y}_j\bar{\mathbf{B}}_{j,i}.$$

for $k = 1, \cdots, K$ **do**

Calculate the elements of Ω_2 (R-H-S of (81))

$$\delta_{i,k} = tr(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}), \tau_{i,k} = tr(\bar{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}\Delta\hat{\mathbf{Z}}_i\Delta\hat{\mathbf{X}}_i)$$

Send $\delta_{i,k}$ and $\tau_{i,k,}\, k = 1, \cdots, K$ to root processor.

Root step 2:
Root processor **do**
Construct the R-H-S of (81) as

$$\Omega_2 = \mu \left[\sum_{i=1}^{N} \delta_{i,1} \quad \sum_{i=1}^{N} \delta_{i,2} \cdots \sum_{i=1}^{N} \delta_{i,K} \right]^T - \left[\sum_{i=1}^{N} \tau_{i,1} \quad \sum_{i=1}^{N} \tau_{i,2} \cdots \sum_{i=1}^{N} \tau_{i,K} \right]^T$$

Solve the following system of equations for the corrector dual variable $\Delta\bar{y}$ and send $\Delta\bar{y}$ to all processors.

$$\Lambda \Delta\bar{y} = \Omega_2 \tag{81}$$

Processors step 3:
for $i = 1, \cdots, N, processor\, i$ **do**
Calculate the corrector step directions as follows.

$$\Delta\bar{\mathbf{Z}}_i = \sum_{j=1}^{K} \Delta\bar{y}_j \bar{\mathbf{B}}_{j,i}$$

$$\Delta\bar{\mathbf{X}}_i = -(\mathbf{Z}_i)^{-1}(\Delta\bar{\mathbf{Z}}_i\mathbf{X}_i + \Delta\hat{\mathbf{Z}}_i\Delta\hat{\mathbf{X}}_i) + \mu(\mathbf{Z}_i)^{-1}$$

Calculate primal dual step total step directions as follows.
$\Delta\mathbf{X}_i = \Delta\hat{\mathbf{X}}_i + \Delta\bar{\mathbf{X}}_i$, $\Delta\mathbf{Z}_i = \Delta\hat{\mathbf{Z}}_i + \Delta\bar{\mathbf{Z}}_i, \Delta y = \Delta\hat{y} + \Delta\bar{y}$.
Set primal step size t_p and dual step size t_d using an appropriate line search methos.
Update primal and dual variables as

$$\mathbf{X}_i \equiv \mathbf{X}_i + t_p\Delta\mathbf{X}_i, \quad \mathbf{Z}_i \equiv \mathbf{Z}_i + t_d\Delta\mathbf{Z}_i, \quad y \equiv y + t_d\Delta y$$

Processors step 4:
for $i = 1, \cdots, N$, $processor\, i$ **do**
Calculate the contribution to primal cost $\tilde{\phi}_i = tr\,(\bar{\mathbf{C}}_i\mathbf{X}_i)$ and the complementary slack $S_i = tr(\mathbf{Z}_i\mathbf{X}_i)$. Send S_i and $\tilde{\phi}_i$ to root processor.

Root step 4:
Root processor **do**
Update the barrier parameter $\mu = \frac{1}{3}\sum_{i=1}^{N} S_i$. Calculate primal and dual costs as $\phi = \sum_{i=1}^{N}\phi_i$ and $\psi = a^T y$. If $|\phi - \psi| > \varepsilon$, then go to Processors step 1; Otherwise calculate the coefficients of $P(\alpha)$ as $P_i = \sum_{j=1}^{N} E_j y_{(j+\tilde{N}i-1))}$ for $i = 1, \cdots, L_0$.

D. Computational Complexity Analysis: SDP Algorithm

NC $\subset$ P is defined to be the class of problems which can be solved in a polylogarithmic number of steps using a polynomially number processor and is often considered to be the class of problems that can be parallelized efficiently. The class P-complete is a set of problems which are equivalent up to an NC reduction, but contains no problem in NC and is thought to be the simplest class of "inherently sequential" problems. It has been proven that linear programming (LP) is P-complete [54] and SDP is P-hard (at least as hard as any P-complete problem) and, thus, is unlikely to admit a general-purpose parallel solution. Given this fact and given the observation that the problem we are trying to solve is NP-hard, it is important to thoroughly understand the complexity of the algorithms we are proposing and how this complexity scales with various parameters which define the size of the problem. To better understand these issues, we have broken our complexity analysis down into several cases which should be of interest to the control community. Note that the cases below do not discuss memory complexity. This is because in the cases when a sufficient number of processors are available, for a system with n states, the memory requirements per block are simply proportional to n^2.

1) Case 1: Systems With a Large Number of States: Suppose we are considering a problem with n states. For this case, the most expensive part of the algorithm is the calculation of the Schur complement matrix Λ by the processors in Processors step 1 (and summed by the root in Root step 1, although we neglect this part). In particular, the computational complexity of the algorithm is determined by the number of operations required to calculate (79), restated here

$$\lambda_{i,k,l} = tr\left(\overline{\mathbf{B}}_{k,i}(\mathbf{Z}_i)^{-1}\overline{\mathbf{B}}_{l,i}\mathbf{X}_i\right) \text{ for } k = 1, \ldots, K \text{ and } \quad l = 1, \ldots, K. \tag{82}$$

Since the cost of $n \times n$ matrix-matrix multiplication is $\sim n^3$ and each $\mathbf{X}_i, \mathbf{Z}_i, \overline{\mathbf{B}}_{l,i}$ has $\texttt{floor}(L + M/N)$ number of blocks in $\mathbb{R}^{n\times n}$, the number of operations performed by the ith processor to calculate $\lambda_{i,k,l}$ for $k = 1, \ldots, K$ and $l = 1, \ldots, K$ is

$$\begin{cases} \sim \texttt{floor}\left(\dfrac{L+M}{N}\right) K^2 n^3 & N < L + M \\ \sim K^2 n^3 & N \geq L + M \end{cases} \tag{83}$$

at each iteration, where $i = 1, \ldots, N$. By substituting K in (83) from (35), for $N \geq L + M$, each processor performs

$$\sim \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l-1)!)^2} n^7 \tag{84}$$

operations per iteration. Therefore, for systems with large n and fixed d_p and l, the number of operations per processor required to solve the SDP associated with parameter-dependent feasibility problem $A(\alpha)^T P(\alpha) + P(\alpha)A(\alpha) \prec 0$, is proportional to n^7. Solving the LMI associated with the parameter-independent problem $\mathrm{A}^T P + P\mathrm{A} \prec 0$ using our algorithm or most of the SDP solvers, such as [27], [28], [55], also requires $O(n^7)$ operations per processor. Therefore, if we have a sufficient number of processors, the proposed algorithm solves the stability and robust stability problems by performing $O(n^7)$ operations per processor in this case.

2) Case 2: High Accuracy/Low Conservativity: In this case, we consider the effect of raising Polya's exponent. Consider the definition of simplex as follows:

$$\tilde{\Delta}_l = \left\{ \alpha \in \mathbb{R}^l, \sum_{i=1}^{l} \alpha_i = r, \alpha_i \geqslant 0 \right\}. \tag{85}$$

Suppose we now define the accuracy of the algorithm as the largest value of r found by the algorithm (if it exists) such that if the uncertain parameters lie inside the corresponding simplex, the stability of the system is verified. Typically, increasing Polya's exponent d in (10) improves the accuracy of the algorithm. If we again only consider Processor step 1, according to (84), the number of processor operations is independent of the

Polya's exponent d_1 and d_2. Since this part of the algorithm does not vary with Polya's exponent, we look at the root-processing requirements associated with solving the systems of equations in (80) and (81) in Root step 1 using Cholesky factorization. Each system consists of K equations. The computational complexity of Cholesky factorization is $O(K^3)$. Thus, the number of operations performed by the root processor is proportional to

$$K^3 = \frac{((d_p + l - 1)!)^3}{(d_p!)^3((l-1)!)^3} n^6. \tag{86}$$

In terms of communication complexity, the most significant operation between the root and other processors is sending and receiving $\lambda_{i,k,l}$ for $i = 1, \ldots, N$; $k = 1, \ldots, K$; and $l = 1, \ldots, K$ in Processors step 1 and Root step 1. Thus, the total communication cost for N processors per iteration is

$$\sim N \cdot K^2 = N \frac{((d_p + l - 1)!)^2}{(d_p!)^2((l-1)!)^2} n^4. \tag{87}$$

From (84), (86), and (87), it is observed that the number of processors operations, root operations, and communication operations are independent of Polya's exponent d_1 and d_2. Therefore, we conclude that for a fixed d_p and a sufficiently large number of processors $N(N \geq L + M)$, improving the accuracy by increasing d_1 and d_2 does not add any computation per processor or communication overhead.

3) Case 3: Algorithm Scalability/Speedup: The speedup of a parallel algorithm is defined as $\mathrm{SP}_N = T_s/T_N$, where T_s is the execution time of the sequential algorithm and T_N is the execution time of the parallel algorithm using N processors. The speedup is governed by

$$\mathrm{SP}_N = \frac{N}{D + NS}, \tag{88}$$

where D is defined as the ratio of the total operations performed by all processors except root to total operations performed by all processors and root. S is the ratio of the operations performed by root to total operations performed by all processors and root. Suppose that the number of available processors is equal to the number of sub-blocks in C defined in (32). Using the aforementioned definitions for D and S, (84) as the decentralized computation, and (86) as the centralized computation, D and S can be approximated as

$$D \simeq \frac{N \frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2} n^7}{N \frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2} n^7 + \frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3} n^6} \quad \text{and} \tag{89}$$

$$S \simeq \frac{\frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3} n^6}{N \frac{((d_p+l-1)!)^2}{(d_p!)^2((l-1)!)^2} n^7 + \frac{((d_p+l-1)!)^3}{(d_p!)^3((l-1)!)^3} n^6}. \tag{90}$$

According to (26) and (27), the number of processors $N = L + M$ is independent of n; Therefore

$$\lim_{n\to\infty} D = 1 \quad \text{and} \quad \lim_{n\to\infty} S = 0.$$

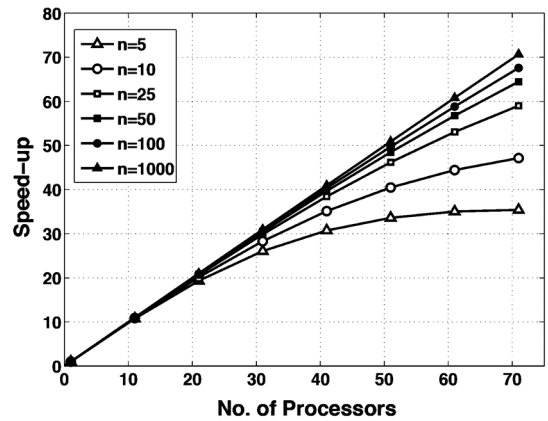


Fig. 4. Theoretical speedup versus number of processors for different system dimensions n for $l = 10$, $d_p = 2$, $d_a = 3$, and $d_1 = d_2 = 4$, where $L + M = 53625$.

By substituting D and S in (88) with their limit values, we have $\lim_{n\to\infty} SP_N = N$. Thus, for large n, by using $L + M$ processors, the presented decentralized algorithm solves large robust stability problems $L + M$ times faster than the sequential algorithms. For different values of the state-space dimension n, the theoretical speedup of the algorithm versus the number of processors is illustrated in Fig. 4. As shown in Fig. 4, for problems with large n, by using $N \leq L + M$ processors, the parallel algorithm solves the robust stability problems approximately N times faster than the sequential algorithm. As n increases, the trend of speedup becomes increasingly linear. Therefore, in case of problems with a large number of states n, our algorithm becomes increasingly efficient in terms of processor utilization.

4) Case 4: Synchronization and Load Balancing: The proposed algorithm is synchronous in that all processors must return values before the centralized step can proceed. However, in the case where we have fewer processors than blocks, some processors may be assigned one block more than other processors. In this case, some processors may remain idle while waiting for the more heavily loaded blocks to complete. In the worst case, this can result in a 50% decrease in speed. We have addressed this issue in the following manner:

1) We allocate almost the same number (± 1) of blocks of the SDP elements C and B_i to all processors, that is, $\texttt{floor}(L + M/N) + 1$ blocks to r processors and $\texttt{floor}(L + M/N)$ blocks to the other $N - r$ processors, where r is the remainder of dividing $L + M$ by N.
2) We assign the same routine to all of the processors in the Processors steps of Algorithm 2.

If $L + M$ is a multiple of N, then the algorithm assigns the same amount of data, that is, $L + M/N$ blocks of C and B_i to each processor. In this case, the processors are perfectly synchronized. If $L + M$ is not a multiple of N, then according to (83), r of N processors perform $K^2 n^3$ extra operations per iteration. This fraction is $1/1 + \texttt{floor}(L + M/N) \leq 0.5$ of the operations per iteration performed by each of the r processors. Thus, in the worst case, we have a 50% reduction, although this situation is rare. As an example, the load balancing (distribution

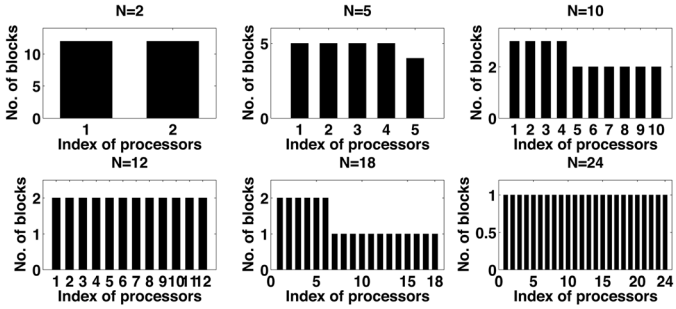


Fig. 5. Number of blocks of the SDP elements assigned to each processor—an illustration of load balancing.

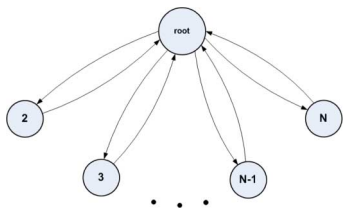


Fig. 6. Communication graph of the SDP algorithm.

of data and calculation) for the case of solving an SDP of the size $L + M = 24$ using different numbers of available processors N is demonstrated in Fig. 5. This figure shows the number of blocks that are allocated to each processor. According to this figure, for $N = 2$, 12, and 24, the processors are well-balanced, whereas for the case where $N = 18$, twelve processors perform 50% fewer calculations.

5) Case 5: Communication Graph: The communication-directed graph of the SDP algorithm (Fig. 6) is static (fixed for all iterations). At each iteration, root sends messages ($\Delta\widehat{y}$ and $\Delta\overline{y}$) to all of the processors and receives messages ($\lambda_{i,k,l}$ in (79)) from all of the processors. The adjacency matrix of the communication-directed graph is defined as follows. For $i = 1, \ldots, N$ and $j = 1, \ldots, N$

$$[T_G]_{i,j} := \begin{cases} 1, & \text{if } \big(i = 1 \text{ or } j = 1\big) \text{ and } \big(i \neq j\big) \\ 0, & \text{Otherwise.} \end{cases}$$

V. Testing and Validation

In this section, we present validation data in four key areas. First, we present analysis results for a realistic large-scale model of Tokamak operation using a discretized PDE model. Next, we present accuracy and convergence data and compare our algorithm to the SOS approach. Next, we analyze the scalability and speedup of our algorithm as we increase the number of processors and compare our results to the general-purpose parallel SDP solver SDPARA. Finally, we explore the limits of the algorithm in terms of problems size when implemented on a moderately powerful cluster computer and using a moderate processor allocation on the Blue Gene supercomputer.

1) Example 1: Application to Control of a Discretized PDE Model in Fusion Research: The goal of this example is to use the proposed algorithm to solve a real-world stability problem. A simplified model for the poloidal magnetic flux gradient in a Tokamak reactor [56] is

$$\frac{\partial \psi_x(x,t)}{\partial t} = \frac{1}{\mu_0 a^2}\frac{\partial}{\partial x}\left(\frac{\eta(x)}{x}\frac{\partial}{\partial x}\left(x\psi_x(x,t)\right)\right) \tag{91}$$

with the boundary conditions $\psi_x(0,t) = 0$ and $\psi_x(1,t) = 0$, where ψ_x is the deviation of the flux gradient from a reference flux gradient profile, μ_0 is the permeability of free space, $\eta(x)$ is the plasma resistivity and a is the radius of the last closed magnetic surface (LCMS). To obtain the finite-dimensional state-space representation of the PDE, we discretize the PDE in the spatial domain $(0, 1)$. The state-space model is then

$$\dot{\psi}_x(t) = A(\eta(x))\psi_x(t), \tag{92}$$

where $A(\eta(x)) \in \mathbb{R}^{N\times N}$ has the following nonzero entries:

$$a_{11} = \frac{-4}{3\mu_0\Delta x^2 a^2}\left(\frac{\eta(x_{3/2})}{x_{3/2}} + \frac{2\eta(x_{3/4})}{x_{3/4}}\right),$$
$$a_{12} = \frac{4}{3\mu_0\Delta x^2 a^2}\left(\frac{\eta(x_{3/2})x_2}{x_{3/2}}\right), \tag{93}$$
$$a_{j,j-1} = \frac{1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j-1/2})}{x_{j-1/2}}x_{j-1}\right) \text{ for } j = 2, \ldots, N-1 \tag{94}$$
$$a_{j,j} = \frac{-1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j+1/2})}{x_{j+1/2}} + \frac{\eta(x_{j-1/2})}{x_{j-1/2}}\right)x_j \text{ for } j = 2, \ldots, N-1 \tag{95}$$
$$a_{j,j+1} = \frac{1}{\Delta x^2\mu_0 a^2}\left(\frac{\eta(x_{j+1/2})}{x_{j+1/2}}x_{j+1}\right) \text{ for } j = 2, \ldots, N-1 \tag{96}$$
$$a_{N,N-1} = \frac{4}{3\Delta x\mu_0 a^2}\frac{\eta(x_{N-1/2})x_{N-1}}{x_{N-1/2}\Delta x},$$
$$a_{N,N} = \frac{-4}{3\Delta x\mu_0 a^2}\left(\frac{2\eta(x_{N+1/4})x_N}{x_{N+1/4}\Delta x} + \frac{\eta(x_{N-1/2})x_N}{x_{N-1/2}\Delta x}\right), \tag{97}$$

where $\Delta x = 1/N$ and $x_j := (j - 1/2)\Delta x$.

We discretize the model at $N = 7$ points. Typically, the $\eta(x_k)$ are not precisely known (they depend on other state variables), so we substitute for $\eta(x_k)$ in (92) with $\widehat{\eta}(x_k) + \alpha_j$, where $\widehat{\eta}(x_k)$ are the nominal values of $\eta(x_k)$, and α_j are the uncertain parameters. At $x_k = 0.036$, 0.143, 0.286, 0.429, 0.571, 0.714, 0.857, 0.964, we use data from the Tore Supra reactor to estimate the $\widehat{\eta}(x_k)$ as $1.775\cdot10^{-8}, 2.703\cdot10^{-8}, 5.676\cdot10^{-8}, 1.182\cdot10^{-7}, 2.058\cdot10^{-7}, 3.655\cdot10^{-7}, 1.076\cdot10^{-6}, 8.419\cdot10^{-6}$. The uncertain system is then written as

$$\dot{\psi}_x(t) = A(\alpha)\psi_x(t), \tag{98}$$

where A is affine, $A(\alpha) = A_0 + \sum_{i=1}^{8} A_i\alpha_i$ (the A_i are omitted for the sake of brevity). For a given ρ, we restrict the uncertain parameters α_j to S_ρ, defined as

$$S_\rho := \{\alpha \in \mathbb{R}^8 : \sum_{i=1}^{8}\alpha_i = -6|\rho|, -|\rho| \leq \alpha_i \leq |\rho|\}, \tag{99}$$

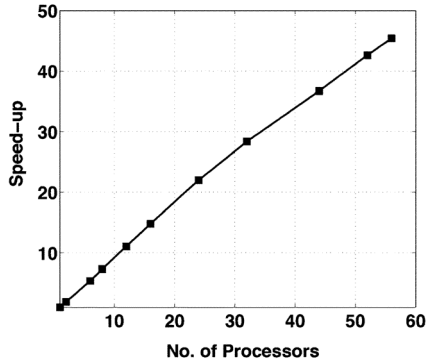


Fig. 7. Speedup of setup and SDP algorithms versus the number of processors for a discretized model of magnetic flux in Tokamak.

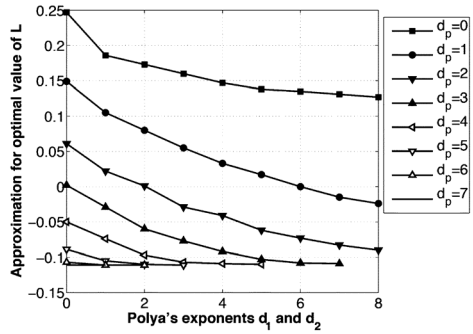


Fig. 8. Upper bound on optimal L versus Polya's exponents d_1 and d_2, for different degrees of $P(\alpha)$. ($d_1 = d_2$).

which is a simplex translated to the origin. We would like to determine the maximum value of ρ such that the system is stable by solving the following optimization problem:

$$\begin{aligned} \max \quad & \rho \\ \text{s.t.} \quad & \text{System (98) is stable for all } \alpha \in S_\rho. \end{aligned} \tag{100}$$

To represent S_ρ using the standard unit simplex defined in (7), we define the invertible map $g : \Delta_8 \to S_\rho$ as

$$g(\alpha) = [\, g_1(\alpha) \quad \cdots \quad g_8(\alpha)\,], \; g_i(\alpha) := 2|\rho|(\alpha_i - |\rho|). \tag{101}$$

Then, if we let $A'(\alpha) = A(g(\alpha))$, since g is one-to-one

$$\begin{aligned} \{A(\alpha') : \alpha' \in S_\rho\} &= \{A(g(\alpha)) : \alpha \in \Delta_8\} \\ &= \{A'(\alpha) : \alpha \in \Delta_8\}. \end{aligned}$$

Thus, the stability of $\dot{\psi}_x(t) = A'(\alpha)\psi_x(t)$, for all $\alpha \in \Delta_l$ is equivalent to the stability of (98) for all $\alpha \in S_\rho$.

We solve the optimization problem in (100) using bisection. For each trial value of ρ, we use the proposed parallel SDP solver to solve the associated SDP obtained by the parallel setup algorithm. The SDP problems have 224 constraints with the primal variable $X \in \mathbb{R}^{1092\times 1092}$. The normalized maximum value of ρ is found to be 0.0019. In this particular example, the optimal value of ρ does not change with the degrees of $P(\alpha)$ and Polya's exponents d_1 and d_2, primarily because the model is affine.

The SDPs are constructed and solved on a parallel Linux-based cluster Cosmea at Argonne National Laboratory. Fig. 7 shows the algorithm speed-up versus the number of processors. Note that solving this problem by SOSTOOLS [21] on the same machine is impossible due to the lack of unallocated memory.

2) Example 2: Accuracy and Convergence: The goal of this example is to investigate the effect of the degree of $P(\alpha)$, d_p, and the Polya's exponents d_1, d_2 on the accuracy of the algorithm. Given a computer with a fixed amount of RAM, we compare the accuracy of the proposed algorithm with the SOS algorithm. Consider the system $\dot{x}(t) = A(\alpha)x(t)$ where A is a polynomial degree 3 defined as

$$\begin{aligned} A(\alpha) = A_1\alpha_1^3 + A_2\alpha_1^2\alpha_2 + A_3\alpha_1\alpha_2\alpha_3 \\ + A_4\alpha_1\alpha_3^2 + A_5\alpha_2^3 + A_6\alpha_3^3 \end{aligned} \tag{102}$$

with the constraint

$$\alpha \in S_L := \left\{ \alpha \in \mathbb{R}^3 : \sum_{i=1}^{3} \alpha_i = 2L+1, L \le \alpha_i \le 1 \right\}$$

$$A_1 = \begin{bmatrix} -0.61 & -0.56 & 0.402 \\ -0.48 & -0.550 & 0.671 \\ -1.01 & -0.918 & 0.029 \end{bmatrix},$$

$$A_2 = \begin{bmatrix} -0.484 & -0.86 & 1.5 \\ -0.732 & -0.841 & -0.126 \\ 0.685 & 0.305 & 0.106 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} -0.357 & 0.344 & -0.661 \\ -0.210 & -0.505 & 0.588 \\ 0.268 & 0.487 & -0.846 \end{bmatrix},$$

$$A_4 = \begin{bmatrix} -0.881 & -0.436 & 0.228 \\ 0.503 & -0.812 & 0.249 \\ -0.012 & 0.542 & -0.536 \end{bmatrix}$$

$$A_5 = \begin{bmatrix} -0.703 & -0.298 & -0.178 \\ 0.402 & -0.761 & -0.300 \\ -0.010 & 0.461 & -0.588 \end{bmatrix},$$

$$A_6 = \begin{bmatrix} -0.201 & -0.182 & -0.557 \\ 0.803 & -0.412 & -0.203 \\ -0.440 & 0.011 & -0.881 \end{bmatrix}.$$

Defining g as in Example 1, the problem is

$$\begin{aligned} \min \quad & L \\ \text{s.t.} \quad & \dot{x}(t) = A(g(\alpha))x(t) \text{ is stable for all } \alpha \in \Delta_3. \end{aligned} \tag{103}$$

Using bisection in L, as in Example 1, we varied the parameters d_p, d_1, and d_2. The cluster computer Karlin at the Illinois Institute of Technology with 24 GB/node of RAM (216-GB total memory) was used to run our algorithm. The upper bounds on the optimal L are shown in Fig. 8 in terms of d_1 and d_2 and for different d_p. Considering the optimal value of L to be $L_{\text{opt}} = -0.111$, Fig. 8 shows how increasing d_p and/or d_1,

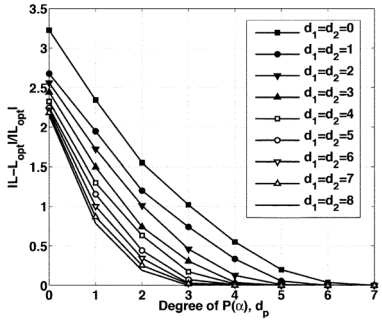


Fig. 9. Error of the approximation for the optimal value of L versus degrees of $P(\alpha)$ for different Polya's exponents.

TABLE I
UPPER BOUNDS FOUND FOR $L_{\rm opt}$ BY THE SOS ALGORITHM USING DIFFERENT DEGREES FOR x AND α (INF: INFEASIBLE, O.M.: OUT OF MEMORY)

Degree in x \ Degree in α	0	1	2
1	inf.	inf.	inf.
2	inf.	-0.102	O.M.
3	inf.	O.M.	O.M.

d_2—when they are still relatively small—improves the accuracy of the algorithm. Fig. 9 demonstrates how the error in our upper bound for $L_{\rm opt}$ decreases by increasing d_p and/or d_1, d_2.

For comparison, we solved the same stability problem using the SOS algorithm [21] using only a single node of the same cluster computer and 24 GB of RAM. We used the Positivstellensatz approach based on [57] to impose the constraints $\sum_{i=1}^{3}\alpha_i = 2L+1$ and $L \le \alpha_i \le 1$. Table I shows the upper bounds on L given by the SOS algorithm using different degrees for x and α. By considering a Lyapunov function of degree two in x and degree one in α, the SOS algorithm gives -0.102 as the upper bound on $L_{\rm opt}$ compared with our value of -0.111. Increasing the degree of α in the Lyapunov function beyond degree two resulted in a failure due to a lack of memory. Note that while relevant, this comparison may not be entirely fair since the SOS algorithm has not been decentralized and it can handle global nonlinear stability problems, which our algorithm cannot.

3) Example 3: Speedup: In this example, we evaluate the efficiency of the algorithm in using additional processors to decrease computation time. As mentioned in Section V-D on computational complexity, the measure of this efficiency is called "speedup" and in Case 3, we gave a formula for this number. To evaluate the true speedup, we first ran the setup algorithm on the Blue Gene supercomputer at Argonne National Laboratory using three random linear systems with different state-space dimensions and numbers of uncertain parameters. Fig. 10 shows a log-log plot of the computation time of the setup algorithm versus the number of processors. As can be seen, the scalability of the algorithm is practically ideal for several different state-space dimensions and numbers of uncertain parameters.

To evaluate the speedup of the SDP portion of the algorithm, we solved three random SDP problems with different dimensions using the Karlin cluster computer. Fig. 11 gives a log-log plot of the computation time of the SDP algorithm versus the number of processors for three different dimensions

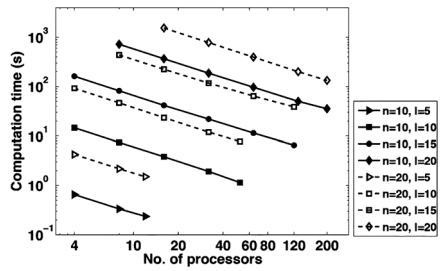


Fig. 10. Computation time of the parallel setup algorithm versus the number of processors for different dimensions of linear system n and numbers of uncertain parameters l- executed on the Blue Gene supercomputer of the Argonne National Laboratory.

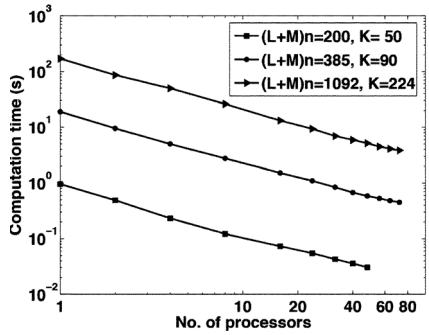


Fig. 11. Computation time of the parallel SDP algorithm versus the number of processors for different dimensions of primal variable $(L+M)n$ and of dual variable K—executed on the Karlin cluster computer of Illinois Institute of Technology.

of the primal variable X and the dual variable y. As indicated in the figure, the three dimensions of the primal variable X are 200, 385, and 1092, and the dimensions of the dual variable y are $K = 50$, 90, and 224, respectively. In all cases, $d_p = 2$ and $d_1 = d_2 = 1$. The linearity of the Time versus Number of Processors curves in all three cases demonstrates the scalability of the SDP algorithm.

For comparison, we plot the speedup of our algorithm versus that of the general-purpose parallel SDP solver SDPARA 7.3.1 as illustrated in Fig. 12. Although similar for a small number of processors, for a larger number of processors, SDPARA saturates, while our algorithm remains approximately linear.

4) Example 4: Max State-Space and Parameter Dimensions for a Nine-Node Linux Cluster Computer: The goal of this example is to show that given moderate computational resources, the proposed decentralized algorithms can solve robust stability problems for systems with 100+ states. We used the Karlin cluster computer with 24 GB/node RAM and nine nodes. We ran the setup and SDP algorithms to solve the robust stability problem with dimension n and l uncertain parameters on one and nine nodes of the Karlin cluster computer. Thus, the total memory access was thus 24 Gig and 216 Gig, respectively. Using trial and error, for different n and d_1, d_2, we found the largest l for which the algorithms do not terminate due to insufficient memory (Fig. 13). In all of the runs $d_a = d_p = 1$. Fig. 13 shows that by using 216-GB RAM, the algorithms can solve the stability problem of size $n = 100$ with four uncertain parameters in $d_1 = d_2 = 1$ Polya's iteration and with three uncertain parameters in $d_1 = d_2 = 4$ Polya's iterations.

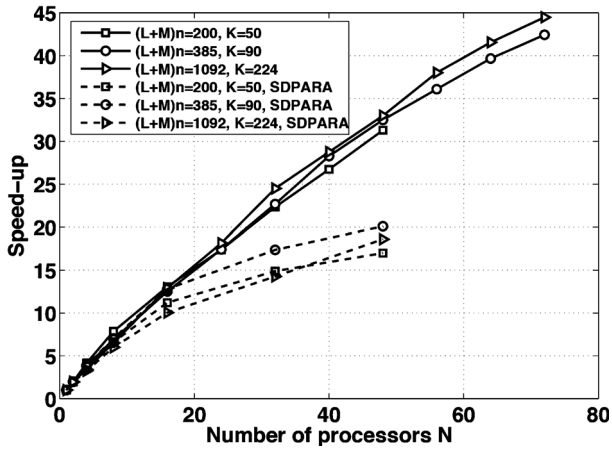


Fig. 12. Comparison between the speedup of the present SDP solver and SDPARA 7.3.1, executed on the Karlin cluster computer.

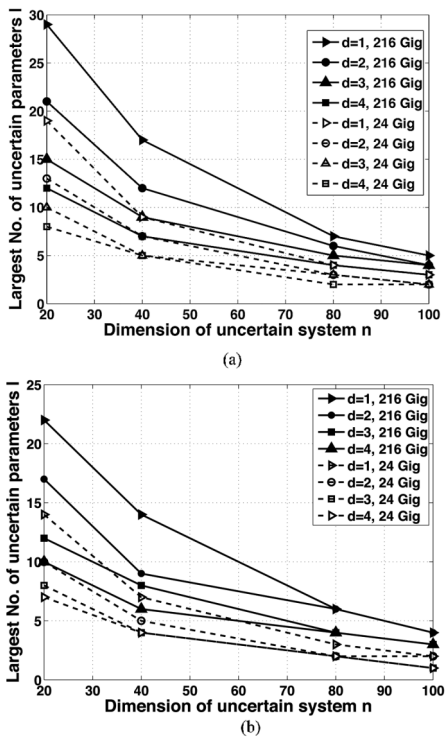


Fig. 13. Largest number of uncertain parameters of n-dimensional systems for which the setup algorithm (a) and SDP solver (b) can solve the robust stability problem of the system using 24- and 216-GB RAM.

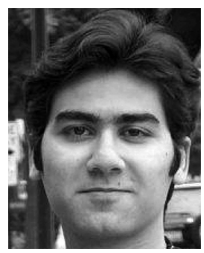
VI. Conclusion

In this paper, we have presented a cluster-computing and supercomputing approach to stability analysis of large-scale linear systems of the form $\dot{x}(t) = A(\alpha)x(t)$, where A is polynomial, $\alpha \in \Delta_l \subset \mathbb{R}^l$ and $x \in \mathbb{R}^n$, and where $n \cong 100$ or $\alpha \cong 10$. The approach is based on mapping the structure of the LMI conditions associated with Polya's theorem to a decentralized computing environment. We have shown that for a sufficient number of processors, the proposed algorithm can solve the NP-hard robust stability problem with the same per-core computation cost as solving the Lyapunov inequality for a system with no parametric uncertainty. Theoretical and experimental results verify near-perfect scalability and speedup for up to 200 processors. Moreover, numerical examples demonstrate the ability of the algorithm to perform robust analysis of systems with 100+ states and several uncertain parameters using a simple nine-node Linux cluster computer. We have also argued that our algorithms can also be extended to solve nonlinear stability analysis and robust controller synthesis problems, although this is left for future work.

References

[1] S. Gugercin and A. Antoulas, "A survey of model reduction by balanced truncation and some new results," *Int. J. Control*, vol. 77, no. 8, pp. 748–766, 2004.
[2] J. Ackermann, A. Bartlett, D. Kaesbauer, W. Sienel, and R. Steinhauser, *Robust Control: Systems with Uncertain Physical Parameters*. New York, USA: Springer-Verlag, 2001.
[3] S. P. Bhattacharyya, H. Chapellat, and L. H. Keel, *Robust Control: The Parametric Approach*. Upper Saddle River, NJ: Prentice-Hall, 1995.
[4] M. Green and D. J. N. Limebeer, *Linear Robust Control*. Upper Saddle River, NJ, USA: Prentice-Hall, 1995.
[5] K. Zhou and J. Doyle, *Essentials of Robust Control*. Upper Saddle River, NJ: Prentice-Hall, 1998.
[6] V. Blondel and J. Tsitsiklis, "A survey of computational complexity results in systems and control," *Automatica*, vol. 36, no. 9, pp. 1249–1274, 2000.
[7] A. Nemirovskii, "Several Np-hard problems arising in robust stability analysis," *Math. Control, Signals, Syst.*, vol. 6, no. 2, pp. 99–105, 1993.
[8] D. Walker and J. Dongarra, "Mpi: A standard message passing interface," *Supercomputer*, vol. 12, pp. 56–68, 1996.
[9] A. Packard and J. Doyle, "Quadratic stability with real and complex perturbations," *IEEE Trans. Autom. Control*, vol. 35, no. 2, pp. 198–201, Feb. 1990.
[10] B. R. Barmish and C. L. DeMarco, "A new method for improvement of robustness bounds for linear state equations," presented at the Conf. Inform. Sci. Syst., NJ, 1986.
[11] P. Gahinet, P. Apkarian, and M. Chilali, "Affine parameter-dependent Lyapunov functions and real parametric uncertainty," *IEEE Trans. Autom. Control*, vol. 41, no. 3, pp. 436–442, Mar. 1996.
[12] R. C. L. F. Oliveira and P. L. D. Peres, "Stability of polytopes of matrices via affine parameter-dependent Lyapunov functions: Asymptotically exact LMI conditions," *Linear Algebra Appl.*, vol. 405, no. 3, pp. 209–228, Aug. 2005.
[13] R. C. L. F. Oliveira and P. L. D. Peres, "A less conservative LMI condition for the robust stability of discrete-time uncertain systems," *Syst. Control Lett.*, vol. 43, no. 4, pp. 371–378, Aug. 2001.
[14] D. Ramos and P. Peres, "An LMI approach to compute robust stability domains for uncertain linear systems," in *Proc. Amer. Control Conf.*, Jun. 2001, pp. 4073–4078.
[15] P. A. Bliman, "An existence result for polynomial solutions of parameter dependent LMIs," *Syst. Control Lett.*, no. 3–4, pp. 165–169, 2004.
[16] M. Peet, "Exponentially stable nonlinear systems have polynomial Lyapunov functions on bounded regions," *IEEE Trans. Autom. Control*, vol. 54, no. 5, pp. 979–987, May 2009.
[17] A. Ben-Tal and A. Nemirovski, "Robust convex optimization," *Math. Oper. Res.*, vol. 23, no. 4, pp. 769–805, 1998.
[18] P. A. Bliman, "A convex approach to robust stability for linear systems with uncertain scalar parameters," *SIAM J. Control Optim*, vol. 42, no. 3–4, pp. 2016–2042, 2004.
[19] X. Zhang and P. Tsiotras, "Parameter-dependent Lyapunov functions for stability analysis of lti parameter dependent systems," in *Proc. IEEE 42nd Conf. Decision Control*, 2003, pp. 5168–5173.
[20] X. Zhang, P. Tsiotras, and P. A. Bliman, "Multi-parameter dependent Lyapunov functions for the stability analysis of parameter-dependent LTI systems," in *Proc. IEEE Int. Symp. Mediterrean Conf. Control Autom.*, 2005, pp. 1263–1268.
[21] S. Prajna, A. Papachristodoulou, and P. A. Parrilo, "Introducing SOSTOOLS: A general purpose sum of squares programming solver," in *Proc. IEEE Conf. Decision Control*, 2002, pp. 741–746.
[22] D. Henrion and J. B. Lassere, "Gloptipoly: Global optimization over polynomials with Matlab and SeDuMi," in *Proc. IEEE Conf. Decision Control*, Mar. 2003, pp. 747–752.
[23] C. W. Scherer and C. W. J. Hol, "Matrix sum-of squares relaxations for robust semi-definite programs," *Math. Programming Ser. B*, vol. 107, no. 1–2, pp. 189–211, 2006.
[24] G. Chesi, A. Garulli, A. Tesi, and A. Vicino, "Polynomially parameter-dependent Lyapunov functions for robust stability of polytopic systems: An LMI approach," *IEEE Trans. Autom. Control*, vol. 50, no. 3, pp. 365–370, Mar. 2005.

[25] R. C. L. F. Oliveira and P. L. D. Peres, "Parameter-dependent LMIs in robust analysis: Characterization of homogeneous polynomially parameter-dependent solutions via LMI relaxations," *IEEE Trans. Autom. Control*, vol. 52, no. 7, pp. 1334–1340, Jul. 2007.
[26] R. C. L. F. Oliveira, P.-A. Bliman, and P. L. D. Peres, "Robust LMIs with parameters in multi-simplex: Existence of solutions and applications," in *Proc. IEEE Conf. Decision Control*, 2008, pp. 2226–2231.
[27] B. Borchers and J. G. Young, "Implementation of a primal dual method for SDP on a shared memory parallel architecture," *Comput. Optimiz. Appl.*, vol. 37, no. 3, pp. 355–369, 2007.
[28] M. Yamashita, K. Fujisawa, and M. Kojima, "SDPARA: Semidefinite programming algorithm parallel version," *Parallel Comput.*, vol. 29, pp. 1053–1067, 2003.
[29] G. M. Amdahl, "Validity of the single processor approach to achieving large-scale computing capabilities," in *Proc. AFIPS Conf.*, 1967, pp. 483–485.
[30] L. Kalé, B. Ramkumar, A. Sinha, and A. Gursoy, "The charm parallel programming language and system: Part i-description of language features" Parallel Programming Lab. Tech. Rep. 95-02, 1994.
[31] S. Deitz, "High-level programming language abstractions for advanced and dynamic parallel computations," Ph.D. dissertation, Comput. Sci. Eng. Dept., University of Washington, , 2005.
[32] K. Gatermann and P. Parrilo, "Symmetry groups, semidefinite programs, and sums of squares," *J. Pure Appl. Algebra*, vol. 192, no. 1, pp. 95–128, 2004.
[33] P. Parrilo, "Exploiting algebraic structure in sum of squares programs," *Positive Polynomials Control*, pp. 580–580, 2005.
[34] S. Kim, M. Kojima, and H. Waki, "Generalized lagrangian duals and sums of squares relaxations of sparse polynomial optimization problems," *SIAM J. Optimiz.*, vol. 15, no. 3, pp. 697–719, 2005.
[35] H. Waki, S. Kim, M. Kojima, M. Muramatsu, and H. Sugimoto, "Algorithm 883: Sparsepop—A sparse semidefinite programming relaxation of polynomial optimization problems," *ACM Trans. Math. Softw.*, vol. 35, no. 2, 2008.
[36] D. Cox, J. Little, and D. O'Shea*, Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. New York, USA: Springer-Verlag, 2007, vol. 10.
[37] B. Buchberger and F. Winkler*, Gröbner Bases and Applications*. Cambridge, U.K.: Cambridge Univ Press, 1998, vol. 251.
[38] M. M. Peet and Y. V. Peet, "A parallel-computing solution for optimization of polynomials," in *Proc. Amer. Control Conf.*, Jun./Jul. 2010, pp. 4851–4856.
[39] E. Scheinerman*, Mathematics: A Discrete Introduction*, 2nd ed. USA: Thomson Brooks/Cole Publishing, 2005.
[40] G. Hardy, J. E. Littlewood, and G. Pólya*, Inequalities*. Cambridge, U.K.: Cambridge University Press, 1934.
[41] M. Castle, V. Powers, and B. Reznick, "A quantitative polya's theorem with zeros," *Effective Methods in Algebraic Geometry*, vol. 44, no. 9, pp. 1285–1290, 2009.
[42] P. Gahinet and P. Apkarian, "A linear matrix inequality approach to H infinity control," *Int. J. Robust Nonlinear Control*, vol. 4, pp. 421–448, 1994.
[43] G. Dullerud and F. Paganini*, A Course in Robust Control Theory*. New York: Springer, 2000, vol. 6.
[44] S. J. Benson, "DSDP3: Dual scaling algorithm for general positive semidefinite programs," Tech Rep. ANL/MCS-P851-1000, 2001, Argonne National Labs.
[45] S. J. Benson, Y. Ye, and X. Zhang, "Solving large-scale sparse semidefinite programs for combinatorial optimization," *SIAM J. Optimiz.*, vol. 10, pp. 443–461, 1998.
[46] F. Alizadeh, J. A. Haeberly, and M. Overton, "Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results," *SIAM J. Optimiz.*, vol. 8, no. 3, pp. 746–768, 1998.
[47] R. D. C. Monteiro, "Primal-dual path following algorithms for semidefinite programming," *SIAM J. Optimiz.*, vol. 7, no. 3, 1997.
[48] C. Helmberg, F. R. R. J. Vanderbei, and H. Wolkovicz, "An interior-point method for semidefinite programming," *SIAM J. Optimiz.*, vol. 6, pp. 342–361, 1996.
[49] C. Helmberg and F. Rendl, "A spectral bundle method for semidefinite programming," *SIAM J. Optimiz.*, vol. 10, no. 3, pp. 673–696, 2000.
[50] K. K. Sivaramakrishnan, "A parallel interior point decomposition algorithm for block angular semidefinite programs," *Comput. Optim. Appl.*, vol. 46, no. 1, pp. 1–29, 2010.
[51] M. Nayakkankuppam, "Solving large-scale semidefinite programs in parallel," *Math. Program.*, vol. 109, no. 2, pp. 477–504, 2007.
[52] M. L. O. F. Alizadeh and J. P. A. Haeberly, "Primal-dual interior-point methods for semidefinite programming," presented at the Math Programming Symp., Ann Arbor, MI, 1994.
[53] F. Alizadeh, J. Haeberly, and M. Overton, "Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results," *SIAM J. Optimiz.*, vol. 8, no. 3, pp. 746–768, 1998.
[54] R. Greenlaw, H. Hoover, and W. Ruzzo*, Limits to Parallel Computation: P-Completeness Theory*. New York, USA: Oxford University Press, 1995.
[55] J. Sturm, "Using sedumi 1.02, a MATLAB toolbox for optimization over symmetric cones," in *Proc. Optimiz. Meth. Softw.*, 1999, vol. 11–12, pp. 625–653.
[56] E. Witrant, E. Joffrin, S. Brémont, G. Giruzzi, D. Mazon, O. Barana, and P. Moreau, "A control-oriented model of the current profile in tokamak plasma," *Plasma Phys. Controlled Fusion*, vol. 49, pp. 1075–1105, 2007.
[57] G. Stengle, "A Nullstellensatz and a Positivstellensatz in semialgebraic geometry," *Math. Ann.*, vol. 207, no. 2, pp. 87–97, 1973.



Reza Kamyar received the B.S. and M.S. degrees in aerospace engineering from Sharif University of Technology, Tehran, Iran, in 2008 and 2010, respectively, and is currently pursuing the Ph.D. degree in mechanical engineering from Arizona State University, Tempe, AZ, USA.

He is a Research Assistant with Cybernetic Systems and Controls Laboratory (CSCL), School for Engineering of Matter, Transport and Energy (SEMTE), Arizona State University. His research focuses on the development of decentralized algorithms applied to the problems of stability and control of large-scale complex systems.

Matthew M. Peet received the B.S. degree in physics and in aerospace engineering from the University of Texas, Austin, TX, USA, in 1999 and the M.S. and Ph.D. degrees in aeronautics and astronautics from Stanford University, Stanford, CA, in 2001 and 2006, respectively.

He was a Postdoctoral Fellow at the National Institute for Research in Computer Science and Control (INRIA), Paris, France, from 2006 to 2008, where he worked in the SISYPHE and BANG groups. He was an Assistant Professor of Aerospace Engineering in the Mechanical, Materials, and Aerospace Engineering Department, Illinois Institute of Technology, Chicago, IL, USA, from 2008 to 2012. Currently, he is an Assistant Professor of Aerospace Engineering, School for the Engineering of Matter, Transport, and Energy, Arizona State University, Tempe, AZ, USA, and Director of the Cybernetic Systems and Controls Laboratory. His research interests are in the role of computation as it is applied to the understanding and control of complex and large-scale systems. Applications include fusion energy and immunology.

Dr. Peet received a National Science Foundation CAREER award in 2011.

Yulia Peet received the B.S. degree in applied mathematics and physics and the M.S. degree in aerospace engineering from Moscow Institute of Physics and Technology, Moscow, Russia, in 1997 and 1999, respectively, and the Ph.D. degree in aeronautics and astronautics from Stanford University, Stanford, CA, USA, in 2006.

Currently, she is an Assistant Professor of Mechanical and Aerospace Engineering, School for Engineering of Matter, Transport and Energy, Arizona State University, Tempe, AZ, USA. Her previous appointments include a postdoctoral position at the University of Pierre and Marie Curie, Paris, France, in 2006–2008, and a dual appointment as a National Science Foundation Research and Teaching Fellow at Northwestern University, Evanston, IL, USA, and Assistant Computational Scientist at the Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL, USA, in 2009–2012.