Automatica 45 (2009) 776–782

Contents lists available at ScienceDirect

Automatica

journal homepage: www.elsevier.com/locate/automatica

Brief paper

# Probabilistic sorting and stabilization of switched systems☆

Hideaki Ishii [a], Roberto Tempo [b,*]

[a] Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology, 4259 Nagatsuta-cho, Midori-ku Yokohama 226-8502, Japan

[b] IEIIT-CNR, Politecnico di Torino, Corso Duca degli Abruzzi 24, 10129 Torino, Italy

ARTICLE INFO

*Article history:*
Received 13 September 2007
Received in revised form
12 May 2008
Accepted 4 October 2008
Available online 13 January 2009

*Keywords:*
Switched systems
Common Lyapunov functions
Randomized algorithms

ABSTRACT

In this paper, we consider Lyapunov stability of switched linear systems whose switching signal is constrained to a subset of indices. We propose a switching rule that chooses the most stable subsystem among those belonging to the subset. This rule is based on an ordering of the subsystems using a common Lyapunov function. We develop randomized algorithms for finding the ordering as well as for finding a subset of systems for which a common Lyapunov function exists. It is shown that the class of randomized algorithms known as the Las Vegas type is useful in the design procedure. A third-order example illustrating the efficacy of the approach is presented.



## 1. Introduction

Switched systems refer to a class of hybrid dynamical systems characterized by a set of subsystems and a switching rule which specifies a subsystem at each time. Stability of such systems is a difficult problem and has been studied extensively; see, e.g., Liberzon (2003) and Sun and Ge (2005) and the references therein. The approach based on common Lyapunov functions is of particular interest because asymptotic stability under arbitrary switching is achieved if and only if such a function exists. Further, in the design of switching rules, once stability is guaranteed, one can focus on other performance measures of the system.

In this paper, we consider one of the basic instances of switched systems with linear time-invariant subsystems and quadratic common Lyapunov functions. It is well known that the existence of such a Lyapunov function is equivalent to a set of linear matrix inequalities (LMIs). A number of sufficient conditions have been developed; see the abovementioned references and, e.g., Cheng, Guo, and Huang (2003), Gurvits, Shorten, and Mason (2007) and Shorten and Narendra (2002). However, in general, though LMIs can be solved using efficient computational algorithms, the computation of a solution often becomes intractable when the number of subsystems increases.

Within this context, we would like to address the following questions: How can we find a quadratic common Lyapunov function, if not for all of the original subsystems, for a large subset of them? Moreover, once this is accomplished, how can we design a switching rule for achieving good performance by utilizing the common Lyapunov function at hand? These are fairly basic questions, which have not been studied much in the literature.

We formulate a switching control problem in the setting where the system contains discrete parameters which can be tuned at each time instant, but are constrained to a certain range varying over the time. The switching rule determines the parameters from the range in such a way that the system response would be desirable in terms of its control performance.

Specifically, we consider the two problems as follows: (i) To introduce an order among the subsystems based on their relative stability levels with respect to a given common Lyapunov function. (ii) To find the largest subset of the given subsystems such that a common Lyapunov function exists. It is noted that both problems exhibit combinatorial aspects and hence can have high computational complexity. To order the subsystems, one needs to compare each subsystem to the others, while to find the largest subset may require examining LMIs for every possible subset.

We remark that the number of subsystems can be significantly large in the following cases. For example, the switched systems considered may be obtained from linear parameter-varying (LPV) systems (Fujisaki, Dabbene, & Tempo, 2003; Rugh & Shamma, 2000), where the time-varying parameter contains either exogenous parameters that are discretized or internal

☆ The material of this paper was partially presented at the 17th IFAC World Congress, Seoul, Korea, 2008. This paper was recommended for publication in revised form by Associate Editor Masayuki Fujita under the direction of Editor Ian R. Petersen.

* Corresponding author. Tel.: +39 011 564 5408; fax: +39 011 564 5429.
*E-mail addresses:* ishii@dis.titech.ac.jp (H. Ishii), roberto.tempo@polito.it (R. Tempo).

0005-1098/$ – see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.automatica.2008.10.016

discrete parameters such as on/off type switches. The number of subsystems can be exponential in the number of parameters and switches. Another instance where the approach may be useful is the analysis of interval systems (e.g., Alamo, Tempo, Ramírez, and Camacho (2008)); in this case, the vertex systems would be ordered with respect to the stability levels. However, the number of subsystems grows exponentially with the system order.

In this paper, to derive efficient algorithms, we take a probabilistic approach, which is known to overcome the curse of dimensionality. In recent years, methods based on randomized algorithms have been successfully developed for analysis and synthesis of uncertain systems (e.g., Tempo, Calafiore, and Dabbene (2005) and Vidyasagar (1998)) and hybrid systems (e.g., Ishii, Başar, and Tempo (2005) and Liberzon and Tempo (2004)). For control problems which are difficult to solve deterministically, the approach is to introduce control specifications in probabilistic terms. While the specifications can be met only under a certain probability, the computational complexity is less stringent.

In this regard, a contribution of this paper lies in the application of a class of randomized algorithms which has not been well recognized in the control literature. In Tempo and Ishii (2007), we have pointed out that the probabilistic approach for control has mainly relied on the use of Monte Carlo randomized algorithms. Such algorithms may produce incorrect outputs, but the probability of such an event is bounded. On the other hand, in computer science, another class known as the Las Vegas randomized algorithms has been used (Motwani & Raghavan, 1995). This type, in contrast, always produces correct outputs with probability one.

In particular, for the first problem studied in this paper, we develop a Las Vegas algorithm. To order the subsystems with respect to their stability levels, we extend the Randomized Quick Sort (RQS) algorithm, which performs sorting of real numbers very efficiently (Knuth, 1998). Regarding the second problem, for which there is a combinatorial issue as mentioned above, we propose a Monte Carlo type algorithm.

This paper is organized as follows: In Section 2, the system setup is introduced. In Section 3, we describe the algorithm for sorting the subsystems while, in Section 4, we propose methods for finding a subset of systems and a common Lyapunov function. The results are illustrated through a third-order numerical example in Section 5. The paper is concluded in Section 6.

## 2. Problem formulation

In this section, we introduce the switched system setup and the problem of the switching rule design.

Consider the switched linear system specified by

$$\dot{x}(t) = A_{\sigma(t)}x(t), \tag{1}$$

where $x(t) \in \mathbb{R}^n$ is the state and $\sigma(t) \in \mathcal{I}_N$ is the mode of the switching signal with the index set given by $\mathcal{I}_N := \{1, \ldots, N\}$. We introduce two features to the switched system (1) which may limit its switching behavior.

(i) The switching signal is constrained to a subset $\Sigma_0$ of $\mathcal{I}_N$, called the *admissible set*. This set can be chosen arbitrarily prior to operation of the system.

(ii) There is a time-varying index set $\Sigma(t) \subset \Sigma_0$ such that, at each time $t$, the switching signal can take any value from this set as

$$\sigma(t) \in \Sigma(t), \quad \forall t \geq 0.$$

The subsystems whose indices belong to $\Sigma(t)$ are referred to as the *admissible systems* at time $t$.

These two features imply that the designer can specify the index set to which the system should switch. Further, the designer can choose among the index set $\Sigma(t)$ the exact subsystem at each time, but does not have control over determining the set $\Sigma(t)$. Hence, the objective is to find the admissible set $\Sigma_0$ and a rule to determine a subsystem among the admissible systems at each time $t$ that is optimal from a control performance viewpoint.

As mentioned in the Introduction, this setup can arise in LPV systems with discrete time-varying parameters. Such parameters may come from discretizing continuous parameters or from on/off switches. Their time-varying nature is modeled by the set $\Sigma(t)$. We note that the number of subsystems is exponential to the number of the (original) continuous parameters and the switches.

To address this general problem, we follow the approach based on common quadratic Lyapunov functions. We divide the design procedure into two steps.

The first step is to order the systems according to their levels of stability with respect to a Lyapunov function. For this part, we fix the admissible set $\Sigma_0 \subset \mathcal{I}_N$. Suppose that there is a quadratic function $V(x) = x^{\mathrm{T}}Px$ with a positive-definite matrix $P \in \mathbb{R}^{n\times n}$ which is a common Lyapunov function for all the subsystems, that is,

$$L(P, A_i) < 0 \quad \text{for all } i \in \Sigma_0, \tag{2}$$

where $L(P, A) := A^{\mathrm{T}}P + PA$. We consider to order the matrices $L(P, A_i)$, $i \in \Sigma_0$, since they represent the overall decay rates of the subsystems. In this context, these matrices provide a simple and useful measure of stability. Specifically, for each subsystem $i$, we look for the subset $\overline{\mathcal{A}}_i$ of systems having slower decay rates given by

$$\overline{\mathcal{A}}_i := \left\{A_j : L(P, A_i) \leq L(P, A_j), j \in \mathcal{I}_N, j \neq i\right\}, \tag{3}$$

and the subset $\underline{\mathcal{A}}_i$ of systems with faster decay rates as

$$\underline{\mathcal{A}}_i := \left\{A_j : L(P, A_j) \leq L(P, A_i), j \in \mathcal{I}_N, j \neq i\right\}. \tag{4}$$

Once this goal is achieved, the switching rule is to select the index as $\sigma(t)$ among $\Sigma(t)$ that is the smallest with respect to this ordering; if such an index is not unique, one of them may be arbitrarily chosen. For example, a particular switching rule can be given as

$$\sigma(t) \in \arg\max_{i\in\Sigma(t)} \left|\overline{\mathcal{A}}_i \cup \mathcal{A}_{\Sigma(t)}\right|, \tag{5}$$

where the right-hand side denotes the set of indices in $\Sigma(t)$ achieving the maximum, $\mathcal{A}_{\Sigma(t)} := \{A_i : i \in \Sigma(t)\}$, and $|\cdot|$ is the cardinality of a set.

The second problem is to find the admissible set $\Sigma_0$ such that a common Lyapunov function exists. It is formulated as follows: Find the set $\Sigma_0$ with the maximum cardinality among those having a common Lyapunov function satisfying the condition (2), that is,

$$\Sigma_0 = \arg\max\left\{|\Sigma_1| : \Sigma_1 \subset \mathcal{I}_N, \exists P > 0 \text{ s.t. } L(P, A_i) < 0,\ i \in \Sigma_1\right\}. \tag{6}$$

Clearly, this is closely related to the problem of finding a common Lyapunov function for all subsystems and hence can be handled by methods that directly solve the LMIs in (2). Nevertheless, for such methods, combinatorial issues would arise. In the worst case, we need to check all possible subsets of the index set $\mathcal{I}_N$ whether a common Lyapunov function exists. The number of such subsets is exponential as $2^N$. In contrast, we propose a probabilistic algorithm with polynomial running time.

The results for the first and second problems are presented in Sections 3 and 4, respectively.

## 3. Sorting the subsystems

We consider the first problem of sorting the subsystems in (1). We develop an algorithm for general symmetric matrices, which is a generalization of the RQS for scalars.

We state the problem considered in this section. Given a set $\mathcal{X}$ of $N$ distinct symmetric matrices in $\mathbb{R}^{n\times n}$ as

$$\mathcal{X} := \{X_1, X_2, \ldots, X_N\}, \tag{7}$$

find the following three sets for each $i \in \mathcal{I}_N$:

$$\begin{aligned} \underline{\mathcal{X}}_{X_i} &:= \{X \in \mathcal{X} : X \le X_i,\ X \ne X_i\},\\ \overline{\mathcal{X}}_{X_i} &:= \{X \in \mathcal{X} : X_i \le X,\ X \ne X_i\},\\ \mathcal{N}_{X_i} &:= \mathcal{X} \setminus \left(\underline{\mathcal{X}}_{X_i} \cup \overline{\mathcal{X}}_{X_i} \cup \{X_i\}\right). \end{aligned} \tag{8}$$

The set $\underline{\mathcal{X}}_{X_i}$ consists of matrices smaller than $X_i$ (in the positive-definite sense) while $\overline{\mathcal{X}}_{X_i}$ contains the matrices larger than $X_i$ (in the same sense as above).

In the scalar case, clearly, all elements in $\mathcal{X}$ can be ordered. In the matrix case, however, this is not true; for example, consider the set $\mathcal{X} = \{\mathrm{diag}(1,1), \mathrm{diag}(0,2), \mathrm{diag}(2,0)\}$. Hence, we introduce the set $\mathcal{N}_{X_i}$ of matrices that share no order with $X_i$. Note that the three sets in (8) are unique for the given set $\mathcal{X}$.

For the sorting of scalars, various algorithms are available; see, e.g., Knuth (1998). The RQS is one such algorithm which is known to be very efficient especially when well implemented (Motwani & Raghavan, 1995). Here, we propose a matrix version of the RQS. It is a natural and elegant extension of the original RQS, which allows us to study worst-case and average-case complexity. As in the scalar case, randomization is performed under the uniform distribution. This is because no prior knowledge on the matrices is assumed. Hence, each time a matrix is randomly chosen, equal probabilities are allocated. Another way to extend the original RQS is to sort matrices by their maximum eigenvalues; however, in this case, the results will not take account of all modes. Other approaches based on using matrix transformation of the eigenvalue matrices may be also of interest, but are not analyzed here.

### 3.1. Matrix randomized quick sort

The RQS algorithm consists of two phases: The first phase is based on quick sort ideas, and the second one performs extra comparisons among the matrices.

We now present the first phase. It generates the ternary tree structure shown in Fig. 1 and provides the subsets $\underline{\mathcal{Y}}_{X_i}$ and $\overline{\mathcal{Y}}_{X_i}$ of $\underline{\mathcal{X}}_{X_i}$ and $\overline{\mathcal{X}}_{X_i}$, respectively, for $i \in \mathcal{I}_N$.

**Algorithm 3.1** (*Phase* 1 *of the Matrix RQS*).

(1) Set $\mathcal{S}_1 := \mathcal{X}$ and $k = 1$.
(2) At step $k$, if $|\mathcal{S}_k| > 1$, then do the following, and otherwise go to step 3.
   (a) Take $\boldsymbol{S}_k \in \mathcal{S}_k$ randomly according to the uniform distribution. This is called the *pivot*.
   (b) Compare each matrix in $\mathcal{S}_k$ with $\boldsymbol{S}_k$ and determine the following three sets:
   $$\underline{\mathcal{S}}_k := \{X \in \mathcal{S}_k : X \le \boldsymbol{S}_k,\ X \ne \boldsymbol{S}_k\},$$
   $$\overline{\mathcal{S}}_k := \{X \in \mathcal{S}_k : \boldsymbol{S}_k \le X,\ X \ne \boldsymbol{S}_k\},$$
   $$\mathcal{M}_k := \mathcal{S}_k \setminus \left(\underline{\mathcal{S}}_k \cup \overline{\mathcal{S}}_k \cup \{\boldsymbol{S}_k\}\right).$$
   The set $\mathcal{M}_k$ is a set of matrices which have no order with the pivot $\boldsymbol{S}_k$.
   (c) Relabel the sets according to the tree structure in Fig. 1. That is, let $\boldsymbol{S}_l$ be the left child of $\boldsymbol{S}_k$, and then let
   $$\mathcal{S}_l = \underline{\mathcal{S}}_k, \qquad \mathcal{S}_{l+1} = \mathcal{M}_k, \qquad \mathcal{S}_{l+2} = \overline{\mathcal{S}}_k.$$

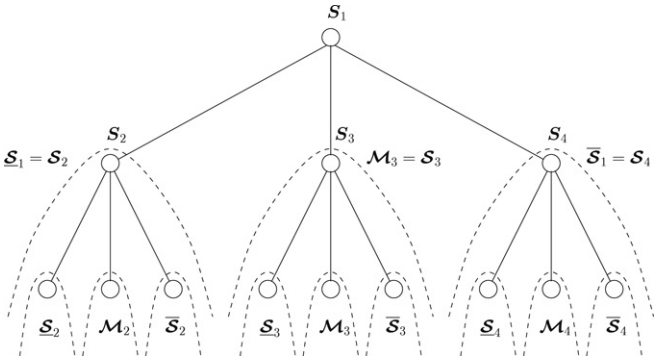


**Fig. 1.** Ternary tree structure.

(3) Let $k = k + 1$. Stop if all leaves in the tree are singletons. Otherwise, go to step $k$ in 2.
(4) After stopping, in the tree in Fig. 1, for each node $\boldsymbol{S}_k$, construct the sets $\underline{\mathcal{Y}}_{\boldsymbol{S}_k}$ and $\overline{\mathcal{Y}}_{\boldsymbol{S}_k}$ as follows:
   Let $\underline{\mathcal{Y}}_{\boldsymbol{S}_k}$ be the set containing $\underline{\mathcal{S}}_k$, all matrices in the left sub-tree of each ancestor $\boldsymbol{S}_l$ of $\boldsymbol{S}_k$ such that $\boldsymbol{S}_l \le \boldsymbol{S}_k$ or $\boldsymbol{S}_k \in \underline{\mathcal{S}}_l$, and the ancestor $\boldsymbol{S}_l$ itself.
   Let $\overline{\mathcal{Y}}_{\boldsymbol{S}_k}$ be the set containing $\overline{\mathcal{S}}_k$, all matrices in the right sub-tree of each ancestor $\boldsymbol{S}_r$ of $\boldsymbol{S}_k$ such that $\boldsymbol{S}_k \le \boldsymbol{S}_r$ or $\boldsymbol{S}_k \in \underline{\mathcal{S}}_r$, and the ancestor $\boldsymbol{S}_r$ itself.

In the scalar case, Algorithm 3.1 coincides with the basic RQS algorithm, and this phase outputs the desired result of sorted numbers in the form of sets: $\underline{\mathcal{X}}_{X_i} = \underline{\mathcal{Y}}_{X_i}$ and $\overline{\mathcal{X}}_{X_i} = \overline{\mathcal{Y}}_{X_i}$, $i \in \mathcal{I}_N$. In the matrix case, however, the sorting requires an additional procedure. This is performed in the second phase of the algorithm.

To illustrate this point more clearly, we show below the candidate matrices after Phase 1 that belong to the sets $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$ and $\overline{\mathcal{X}}_{\boldsymbol{S}_k}$, but may not be included in the sets $\underline{\mathcal{Y}}_{\boldsymbol{S}_k}$ and $\overline{\mathcal{Y}}_{\boldsymbol{S}_k}$, respectively. For each node $\boldsymbol{S}_k$, the sets in (8) can be described as follows.

- $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$: In the tree structure in Fig. 1, all matrices to the left side of $\boldsymbol{S}_k$ are candidate elements of this set. They can be characterized as below:
  . $\underline{\mathcal{S}}_k$ is in $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$.
  . Let $\boldsymbol{S}_l$ be an ancestor of $\boldsymbol{S}_k$ such that $\boldsymbol{S}_l \le \boldsymbol{S}_k$, or $\boldsymbol{S}_k \in \overline{\mathcal{S}}_l$. Then, the left sub-tree of $\boldsymbol{S}_l$ is contained in this set $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$. Also, some matrices in the middle sub-tree of $\boldsymbol{S}_l$ are in $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$.
  . Let $\boldsymbol{S}_l$ be an ancestor of $\boldsymbol{S}_k$ such that $\boldsymbol{S}_k \in \mathcal{M}_l$. Then, some matrices in the left sub-trees of $\boldsymbol{S}_l$ are in $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$.
- $\overline{\mathcal{X}}_{\boldsymbol{S}_k}$: This set has a structure similar to $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$ by taking the right sub-trees instead of the left ones. All matrices to the right of $\boldsymbol{S}_k$ are candidate elements.

It is clear now that further comparisons are necessary especially between matrices in left/right sub-trees and those in middle sub-trees. No comparison is required for matrices in left sub-trees and those in right sub-trees.

In the second phase, in an inverse-order traversal, visit each node $l$, and compare the elements in $\mathcal{M}_l$ with those in $\underline{\mathcal{S}}_l$ and $\overline{\mathcal{S}}_l$. This is formally described as follows.

**Algorithm 3.2** (*Phase* 2 *of the Matrix RQS*).

(1) Perform a traversal in an inverse order. That is, starting from the root, at each node, visit the right child, the middle child, the left child, and then the node itself in a recursive manner.
(2) When visiting the node $l$, if its middle child $\mathcal{M}_l$ exists, then do the following for each matrix $\boldsymbol{M} \in \mathcal{M}_l$; and otherwise go to step 3.

(a) Compare $\boldsymbol{M}$ with the elements in the left child $\underline{\mathcal{S}}_l$ and the right child $\overline{\mathcal{S}}_l$ of the node $l$.
(b) Based on the results, update the sets $\underline{\mathcal{Y}}_{\boldsymbol{M}}$, $\overline{\mathcal{Y}}_{\boldsymbol{M}}$ for the matrix $\boldsymbol{M}$, and also $\underline{\mathcal{Y}}_{\boldsymbol{X}}$, $\overline{\mathcal{Y}}_{\boldsymbol{X}}$ for each $\boldsymbol{X}$ in $\underline{\mathcal{S}}_l$ and $\overline{\mathcal{S}}_l$.

(3) Stop if the current node $l$ is the root (after the traversal). Otherwise, visit the next node and then goto step 2.

Regarding these algorithms, the following worst-case complexity result holds. In Section 3.2, we provide a detailed average complexity analysis.

**Proposition 3.3.** *For a given set $\mathcal{X}$ of symmetric matrices in (7), the matrix RQS algorithm consisting of Algorithms 3.1 and 3.2 always finds the sets in (8) as*

$$\underline{\mathcal{X}}_{X_i} = \underline{\mathcal{Y}}_{X_i}, \qquad \overline{\mathcal{X}}_{X_i} = \overline{\mathcal{Y}}_{X_i}, \qquad \mathcal{N}_{X_i} = \mathcal{X} \setminus \left( \underline{\mathcal{Y}}_{X_i} \cup \overline{\mathcal{Y}}_{X_i} \cup \{X_i\} \right) \tag{9}$$

*for each $i \in \mathcal{I}_N$. The total number of comparisons is no greater than $N(N-1)/2$.*

**Proof.** It is straightforward to show that each matrix $X_i \in \mathcal{X}$ is compared at most once with other matrices $X_j$, $j \neq i$. Hence, the total number of comparisons is always smaller than or equal to the worst-case number $N(N-1)/2$.

When a comparison is performed between two matrices in either Algorithm 3.1 or 3.2, the result is reflected to the corresponding sets $\overline{\mathcal{Y}}_{X_i}$ and $\underline{\mathcal{Y}}_{X_i}$. It is clear that after Algorithm 3.2 stops, the matrices $X_i$ and $X_j$ that have not been compared directly are those satisfying the following: There exists a common ancestor $X_k$ such that $X_i \in \underline{\mathcal{S}}_{X_k}$ and $X_j \in \overline{\mathcal{S}}_{X_k}$. However, from the tree structure, it is clear that $X_i \leq X_j$. Hence, in step 4 of Algorithm 3.1, this information is incorporated so that $X_j \in \overline{\mathcal{Y}}_{X_i}$ and $X_i \in \underline{\mathcal{Y}}_{X_j}$. Therefore, the sets in (8) for each $i \in \mathcal{I}_N$ are correctly constructed as in (9). □

The statement says that, for any set of matrices, the proposed algorithm always finds the correct solution despite the randomization steps. Such algorithms are classified as Las Vegas type randomized algorithms (Motwani & Raghavan, 1995). This class has not been exploited much in the field of control (Tempo & Ishii, 2007).

### 3.2. Analysis on the number of comparisons

In this section, we analyze the complexity of the proposed algorithm. In a Las Vegas algorithm, the running time is random and may be different in each execution. As in the scalar case, we measure the complexity, or the running time, in terms of the expected number of comparisons. Now, denote by $p_{ij}$ the probability that a comparison between two matrices $X_i$ and $X_j$ in $\mathcal{X}$ occurs.

**Theorem 3.4.** *In Algorithm 3.1, for a given set $\mathcal{X}$ of $N$ symmetric matrices, the expected number $C_{ave}$ of comparisons is given by*

$$C_{ave} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} p_{ij} \geq \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \frac{2}{|\Omega_{ij}|},$$

*where*

$$\Omega_{ij} := \mathcal{X} \setminus \left[ \left( \overline{\mathcal{X}}_{X_i} \cap \overline{\mathcal{X}}_{X_j} \right) \cup \left( \underline{\mathcal{X}}_{X_i} \cap \underline{\mathcal{X}}_{X_j} \right) \cup \left( \mathcal{N}_{X_i} \cap \mathcal{N}_{X_j} \right) \right], \quad i, j \in \mathcal{I}_N. \tag{10}$$

**Proof.** Let $\boldsymbol{C}_{ij}$ be the random variable for $i < j$, $i, j \in \mathcal{I}_N$ taking 1 if $X_i$ and $X_j$ are compared during a run and 0 otherwise. Then, the total number $\boldsymbol{C}$ of comparisons in the run is given by $\boldsymbol{C} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \boldsymbol{C}_{ij}$, and hence its expectation $C_{\text{ave}}$ is

$$C_{\text{ave}} = E[\boldsymbol{C}] = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} E\left[\boldsymbol{C}_{ij}\right]. \tag{11}$$

Note that $E\left[\boldsymbol{C}_{ij}\right]$ is equal to the probability $p_{ij}$ that the matrices $X_i$ and $X_j$ are compared during the run.

Let $\mathcal{K}_i := \{k \geq 1 : X_i \in \mathcal{S}_k\}$. Now, we claim that $X_i$ and $X_j$ are compared at step $k$ if and only if $\boldsymbol{S}_k = X_i$ or $X_j$, and for all $l \in \mathcal{K}_i$ such that $l < k$, it holds $\boldsymbol{S}_l \notin \Omega_{ij}$. In words, this condition says that either $X_i$ or $X_j$ is chosen first from the set $\Omega_{ij} \cap \mathcal{S}_k$, $k \in \mathcal{K}_i$, during the run. Suppose that $X_i$ is chosen at step $k$ and that for each $l \in \mathcal{K}_i$ with $l < k$, the matrix $\boldsymbol{S}_l$ is chosen from $\mathcal{X} \setminus \Omega_{ij}$. It can be verified that, as a consequence, $X_j$ is still in the set $\mathcal{S}_k$. Hence, at this step, the two matrices will be compared.

On the other hand, if at step $k \in \mathcal{K}_i$, a matrix in the set $\Omega_{ij}$ is chosen for the first time as $\boldsymbol{S}_k$ and if this matrix is neither $X_i$ nor $X_j$, then during this step, $X_i$ and $X_j$ will be put into separate sets among $\underline{\mathcal{S}}_k$, $\mathcal{M}_k$, and $\overline{\mathcal{S}}_k$. This can be easily shown by noticing

$$\Omega_{ij} = \{X_i, X_j\} \cup \left( \overline{\mathcal{X}}_{X_i} \cap \underline{\mathcal{X}}_{X_j} \right) \cup \left( \underline{\mathcal{X}}_{X_i} \cap \overline{\mathcal{X}}_{X_j} \right) \cup \left( \mathcal{N}_{X_i} \setminus \mathcal{N}_{X_j} \right) \cup \left( \mathcal{N}_{X_j} \setminus \mathcal{N}_{X_i} \right).$$

For example, if $\boldsymbol{S}_k \in \overline{\mathcal{X}}_{X_i} \cap \underline{\mathcal{X}}_{X_j}$, then clearly $X_i$ will be put into $\underline{\mathcal{S}}_k$ while $X_j$ goes into $\overline{\mathcal{S}}_k$. Therefore, the two matrices will not be compared in Phase 1.

At each step $k$, the matrix $\boldsymbol{S}_k$ is chosen independently and uniformly from the set $\mathcal{S}_k$. Hence, the probability that $X_i$ or $X_j$ is chosen first from $\Omega_{ij} \cap \mathcal{S}_k$ satisfies $E\left[\boldsymbol{C}_{ij}\right] \geq 2/|\Omega_{ij}|$. Substituting this into (11), we obtain the result. □

There are two extreme cases in Phase 1. One is when all matrices in $\mathcal{X}$ can be ordered. Then, we have $\mathcal{N}_{X_i} = \emptyset$ for all $i$. This case coincides with the scalar case, and the expected number $C_{\text{ave}}$ of comparisons can be computed explicitly as shown in Mitzenmacher and Upfal (2005): $C_{\text{ave}} = 2(N+1)\sum_{i=1}^{N} 1/i - 4N = 2(N+1)\ln N + \Theta(1)$. The other extreme case is when there is no order at all among the matrices in $\mathcal{X}$. We then have $\overline{\mathcal{X}}_{X_i} = \underline{\mathcal{X}}_{X_i} = \emptyset$ for all $i$. As a consequence, in Phase 1, all matrices are compared to each other. It is easy to check that the number $C_{\text{ave}}$ becomes the worst-case one of $N(N-1)/2$. In these two extreme cases, Phase 2 is unnecessary. Other cases fall between these extreme cases, where in Phase 2, extra comparisons must be carried out. As shown in Proposition 3.3, the total number is always smaller than or equal to the worst-case number $N(N-1)/2$.

The number of comparisons in Phase 2 is difficult to estimate. This fact can be shown through an example as follows. Further, in the numerical example in Section 5, some simulations will be presented.

**Example 3.5.** We study the number of possible tree structures that can be generated by a set $\mathcal{X}$ having $N$ elements. In particular, we consider the case $N = 5$. It is shown that the number of non-sortable matrices after the first phase is not the only factor that determines the number of comparisons in the second phase. Specifically, we present several cases where the associated trees have only one non-sortable matrix (i.e., one matrix that belongs to a mid-tree) after Phase 1, but require different numbers of comparisons in Phase 2.

(1) One possibility is the tree in Fig. 2(a). The first three nodes are in a line with each child on the left, and then the tree branches into one matrix on the left side and one in the middle. In this case, Phase 1 requires nine comparisons, and in Phase 2 there is an additional comparison (between the bottom two nodes). Hence, in total, ten comparisons are made, which is the worst-case number $N(N-1)/2$.

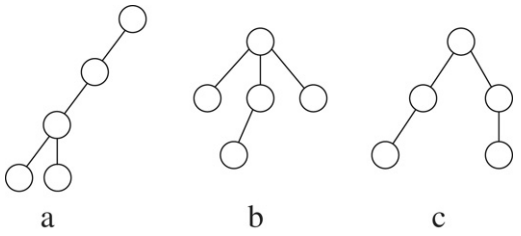


**Fig. 2.** Different trees with 5 nodes.

(2) For the case of Fig. 2(b), in Phase 1, five comparisons are made while, in Phase 2, comparisons are performed four times. The total number is nine.
(3) One of the best cases with a small number of comparisons is the tree in Fig. 2(c). It requires only six comparisons in total, all occurring in Phase 1. ▽

One question of interest is, what are the characteristics of the trees that require a small number of comparisons? In the example above, the case (3) is suggestive. Good cases with fewer comparisons have the following features: (i) Fewer levels in the tree or, in other words, more branches at each node. (ii) Fewer mid-trees in higher levels. These are rather qualitative features. Clearly, due to randomization, from the same set $\mathcal{X}$, trees with and without such features can be produced.

## 4. Finding a subset of systems and a common Lyapunov function

We consider the problem stated in (6), which is to find the largest subset of systems such that a common Lyapunov function exists.

We first introduce some notation. The problem as in (6) is a maximization one. This can be rewritten using the function $J : \mathbb{R}^{n\times n} \to \mathbb{Z}_+$ defined as

$$J(P) = \sum_{i\in\mathcal{I}_N} I_{\{L(P,A_i)<0\}}(P),$$

where $I_{\{L(P,A_i)<0\}}(P)$ is an indicator function given by

$$I_{\{L(P,A_i)<0\}}(P) := \begin{cases} 1 & \text{if } L(P,A_i) < 0, \\ 0 & \text{otherwise.} \end{cases}$$

This function $J(P)$ provides the number of subsystems for which the quadratic function $V(x) = x^{\mathrm{T}}Px$ serves as a common Lyapunov function.

Note that $J(P)$ is invariant under constant scaling: $J(\alpha P) = J(P)$ for any positive real scalar $\alpha$. Due to this property, without loss of generality, we can limit the domain of the function to the unit ball $\mathcal{B}_+ \subset \mathbb{R}^{n\times n}$ of positive-definite matrices given by

$$\mathcal{B}_+ := \left\{P \in \mathbb{R}^{n\times n} : P = P^{\mathrm{T}} > 0,\ \|P\| \le 1\right\}, \tag{12}$$

where $\|\cdot\|$ is a matrix norm. Now, the problem in (6) is reduced to the following maximization problem:

$$J_{\max} := \max_{P\in\mathcal{B}_+} J(P).$$

In the setting of the probabilistic approach, we assume that the matrix $\boldsymbol{P} \in \mathcal{B}_+$ is a random matrix. For simplicity, we employ a uniform density function $f_{\boldsymbol{P}}(\cdot)$ having $\mathcal{B}_+$ as the support set. Let $\mathrm{Prob}_{\boldsymbol{P}}(\cdot)$ be the probability measure induced by this density function.

We employ an algorithm involving the computation of the empirical maximum of $J$. This is defined as follows:

$$\hat{J}_{\max} := \max_{j=1,2,\ldots,M} J(\boldsymbol{P}^{(j)}), \tag{13}$$

where $\boldsymbol{P}^{(j)} \in \mathcal{B}_+$, $j = 1, 2, \ldots, M$, are i.i.d. samples of the random matrix $\boldsymbol{P}$ generated according to the density function $f_{\boldsymbol{P}}$. The empirical maximum $\hat{J}_{\max}$ is determined by a finite number of samples drawn from the set $\mathcal{B}_+$. This is in contrast to the actual worst-case value $J_{\max}$, which is the maximum over $\mathcal{B}_+$ having infinite cardinality. We also note that the empirical maximum $\hat{J}_{\max}$ is a random variable itself since its value depends on the random samples chosen for its computation.

It is clear that the empirical maximum $\hat{J}_{\max}$ is always smaller than $J_{\max}$. Hence, we must question how well $\hat{J}_{\max}$ estimates the true maximum. Under a sufficiently large sample size $M$, a probabilistic statement can be made as shown next (Tempo, Bai, & Dabbene, 1997).

**Proposition 4.1.** *Let $\epsilon, \delta \in (0, 1)$. If the sample size $M$ is such that $M \ge \lceil \log(1/\delta)/\log[1/(1-\epsilon)] \rceil$, then, with probability greater than $1-\delta$, the empirical maximum $\hat{J}_{\max}$ in (13) satisfies the inequality*

$$\mathrm{Prob}_{\boldsymbol{P}}\left\{J(\boldsymbol{P}) \le \hat{J}_{\max}\right\} \ge 1 - \epsilon.$$

*That is,* $\mathrm{Prob}_{\boldsymbol{P}^{(1,\ldots,M)}}\left\{\mathrm{Prob}_{\boldsymbol{P}}\left\{J(\boldsymbol{P}) \le \hat{J}_{\max}\right\} \ge 1-\epsilon\right\} > 1-\delta$, *where* $\mathrm{Prob}_{\boldsymbol{P}^{(1,\ldots,M)}}(\cdot)$ *denotes the probability measure with respect to the multi-sample* $\left\{\boldsymbol{P}^{(1)}, \ldots, \boldsymbol{P}^{(M)}\right\}$.

The result implies that the empirical maximum is an estimate of the true value within an a priori specified *accuracy* $\epsilon$ with *confidence* $1-\delta$ given that the sample size $M$ satisfies the bound. The algorithm may produce an answer not approximately correct, but the probability of this event to occur is no greater than $\delta$. It is emphasized that the sample size $M$ is not dependent on the matrix dimension but only on $\epsilon$ and $\delta$. For methods to generate random matrices required in the algorithm, we refer to Tempo et al. (2005).

## 5. Numerical example

We present an example to illustrate the results.

For the system in (1), we used 20 subsystems. The matrices $A_i$, $i \in \mathcal{I}_N$, were chosen as $A_i = A^* + K_i$, where

$$A^* := \begin{bmatrix} -1.5 & -2 & 0 \\ 2 & 0.5 & 0 \\ 0 & 0 & -0.1 \end{bmatrix},$$

$$K_i := -0.1\sqrt{i}\begin{bmatrix} 1 & 0.3\, r_{i,1} & 0 \\ -1.5\, r_{i,2} & r_{i,3} & 0 \\ 0 & 0 & 0.5 - 0.1 r_{i,4} \end{bmatrix} + 0.05(i)^{0.4}\begin{bmatrix} r_{i,5} & r_{i,6} & r_{i,7} \\ r_{i,8} & r_{i,9} & r_{i,10} \\ r_{i,11} & r_{i,12} & r_{i,13} \end{bmatrix}, \quad i \in \mathcal{I}_N,$$

and $r_{i,j}$ are random numbers uniformly distributed on $[-1, 1]$ for $j$.

For this system, a conventional approach failed to provide a common Lyapunov function. Since $\|sI - A^*\|_\infty \le 10$, by a small-gain type argument (Liberzon, 2003), a sufficient condition for the existence of a common Lyapunov function is given by $\max_i \|K_i\| < 1/10$. However, we computed $\max_i \|K_i\| = 0.611$.

Next, we applied the algorithm in Section 4 and found a matrix $P$ for the common Lyapunov function $V(x)$ in less than 500 iterations in most of the trials. Thus, in this case, the admissible set is $\Sigma_0 = \mathcal{I}_N$.

To find the order in the subsystems, using one of the common Lyapunov functions, we ran the matrix RQS in Section 3. The summary of 30 runs is given in Table 1. On average, the number of comparisons is about 124, which is significantly smaller than the worst-case number $N(N-1)/2 = 190$. From Theorem 3.4, we computed a lower bound on the expected number of comparisons obtaining $C_{\mathrm{ave}} \ge 63.84$; this bound clearly holds in this example.

**Table 1**
Number of comparisons in the matrix RQS (30 runs).

| | Minimum | Maximum | Average |
|---|---|---|---|
| Phase 1 | 74 | 117 | 92.23 |
| Phase 2 | 27 | 36 | 32.10 |
| Total | 102 | 152 | 124.33 |

**Table 2**
The cardinalities of the sets $\overline{\mathcal{A}}_i$ and $\underline{\mathcal{A}}_i$.

| $i$ | $\lvert\overline{\mathcal{A}}_i\rvert$ | $\lvert\underline{\mathcal{A}}_i\rvert$ | $i$ | $\lvert\overline{\mathcal{A}}_i\rvert$ | $\lvert\underline{\mathcal{A}}_i\rvert$ |
|---|---|---|---|---|---|
| 1 | | 7 | 11 | 1 | |
| 2 | | 6 | 12 | | 2 |
| 3 | | 5 | 13 | | |
| 4 | | 6 | 14 | 7 | |
| 5 | | 4 | 15 | 5 | 2 |
| 6 | | 5 | 16 | 12 | |
| 7 | | 1 | 17 | 9 | |
| 8 | | 3 | 18 | | 1 |
| 9 | 5 | 1 | 19 | | |
| 10 | | | 20 | 4 | |

In one such trial, Phase 1 required 90 comparisons and in Phase 2, there were 29 comparisons. Thus, the total is 119. The tree generated had 11 levels. The result is summarized in Table 2. For each subsystem $i$, the number of subsystems slower than it, that is, the cardinality of the set $\overline{\mathcal{A}}_i$ in (3), is given. Also, the number of faster subsystems is shown; this is equal to the cardinality of the set $\underline{\mathcal{A}}_i$ in (4). We see that the faster ones include the subsystems 9, 14, 15, 16, and 17, while the slower ones are the subsystems 1, 2, 3, 4, and 6. Some subsystems such as 10, 13, and 19 do not share any order with others.

The switched system was simulated assuming that every $h = 0.1$ s, the system is provided with 10 randomly selected indices of subsystems from $\mathcal{I}_N$; recall that this index set at the switching time $t = kh$ is denoted by $\Sigma(kh) \subset \Sigma_0$, $k \in \mathbb{Z}_+$. The controller must choose an index from this set $\Sigma(kh)$ and then $\sigma(t)$ takes that value during the next period for $t \in [kh, (k+1)h)$.

We employed three switching rules as follows:

(a) *The random rule*: This is a simple rule. An index from $\Sigma(kh)$ is picked at random.
(b) *The fast rule*: Based on the results of the quick sort, this rule looks at the order of $L(P, A_i)$ for $i \in \Sigma(kh)$ and chooses the index $i$ having the maximum number of subsystems $j$ such that $L(P, A_i) \le L(P, A_j)$. This is similar to the rule given in (5).
(c) *The slow rule*: This is the opposite of the rule (b) above and chooses the subsystem with the largest number of subsystems faster than itself. This rule will produce trajectories with slow decays.

Sample paths for the three systems with $x(0) = [10\ -20\ 5]^{\mathrm{T}}$ are shown in Fig. 3. Clearly, the rule (b) yields the fastest trajectory. These plots exhibit that the proposed sorting algorithm is useful in stabilizing switched systems with fast decays.

## 6. Conclusion

We developed randomized algorithms for two switched-systems problems. Based on the ordering of subsystems, the proposed switching rule chooses one subsystem among the admissible ones at each time to achieve fast decay. Future research topics include developing methods to find common Lyapunov functions suitable for sorting possibly based on optimization criteria different from (6). Also, application of Las Vegas type algorithms to other hybrid systems problems should be of interest.

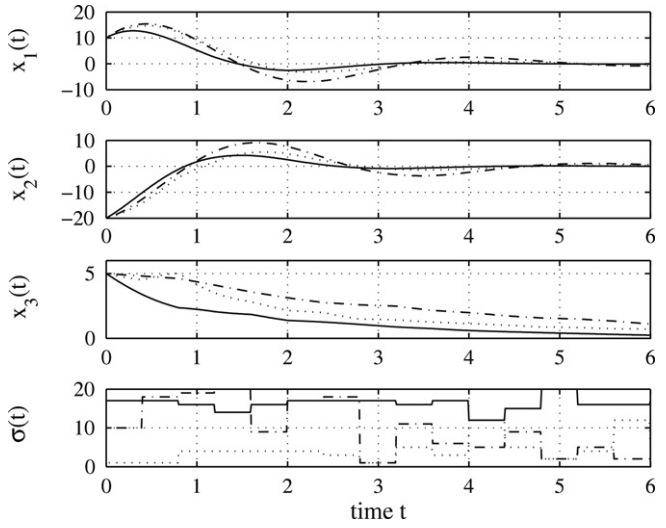


**Fig. 3.** Sample paths of $(x_1, x_2, x_3)$ and $\sigma$: (a) The random rule (dotted), (b) the fast rule (solid), and (c) the slow rule (dash-dot).

## References


Alamo, T., Tempo, R., Ramírez, D. R., & Camacho, E. F. (2008). A new vertex result for robustness problems with interval matrix uncertainty. *Systems & Control Letters*, *57*, 474–481.

Cheng, D., Guo, L., & Huang, J. (2003). On quadratic Lyapunov functions. *IEEE Transactions on Automatic Control*, *48*, 885–890.

Fujisaki, Y., Dabbene, F., & Tempo, R. (2003). Probabilistic design of LPV control systems. *Automatica*, *39*, 1323–1337.

Gurvits, L., Shorten, R., & Mason, O. (2007). On the stability of switched positive linear systems. *IEEE Transactions on Automatic Control*, *52*, 1099–1103.

Ishii, H., Başar, T., & Tempo, R. (2005). Randomized algorithms for synthesis of switching rules for multimodal systems. *IEEE Transactions on Automatic Control*, *50*, 754–767.

Knuth, D. E. (1998). *Sorting and searching*: *Vol. 3. The art of computer programming* (2nd edition). Reading, MA: Addison-Wesley.

Liberzon, D. (2003). *Switching in systems and control*. Boston: Birkhäuser.

Liberzon, D., & Tempo, R. (2004). Common Lyapunov functions and gradient algorithms. *IEEE Transactions on Automatic Control*, *49*, 990–994.

Mitzenmacher, M., & Upfal, E. (2005). *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press.

Motwani, R., & Raghavan, P. (1995). *Randomized algorithms*. Cambridge University Press.

Rugh, W. J., & Shamma, J. S. (2000). Research on gain scheduling. *Automatica*, *36*, 1401–1425.

Shorten, R. N., & Narendra, K. S. (2002). Necessary and sufficient conditions for the existence of a common quadratic Lyapunov function for a finite number of stable second order linear time-invariant systems. *International Journal of Adaptive Control Signal Process*, *16*, 709–728.

Sun, Z., & Ge, S. S. (2005). Analysis and synthesis of switched linear control systems. *Automatica*, *41*, 181–195.

Tempo, R., & Ishii, H. (2007). Monte Carlo and Las Vegas randomized algorithms for systems and control: An introduction. *European Journal of Control*, *13*, 189–203.

Tempo, R., Bai, E. W., & Dabbene, F. (1997). Probabilistic robustness analysis: Explicit bounds for the minimum number of samples. *Systems & Control Letters*, *30*, 237–242.

Tempo, R., Calafiore, G., & Dabbene, F. (2005). *Randomized algorithms for analysis and control of uncertain systems*. London: Springer.

Vidyasagar, M. (1998). Statistical learning theory and randomized algorithms for control. *IEEE Control Systems Magazine*, *18*(6), 69–85.


**Hideaki Ishii** was born in 1972 in Tokyo, Japan. He received the M.Eng. degree in applied systems science from Kyoto University in 1998, and the Ph.D. degree in electrical and computer engineering from the University of Toronto in 2002. He was a Postdoctoral Research Associate of the Coordinated Science Laboratory, the University of Illinois at Urbana-Champaign from 2001 to 2004, and a Research Associate of the Department of Information Physics and Computing, The University of Tokyo from 2004 to 2007. He is currently an Associate Professor of the Department of Computational Intelligence and Systems Science, Tokyo Institute of Technology. His research interests are in networked control systems, hybrid systems, and probabilistic algorithms. He is an Associate Editor of Automatica and the IEEE Transactions on Automatic Control.

**Roberto Tempo** is currently a Director of Research of Systems and Computer Engineering at the institute IEIIT, National Research Council (CNR), Politecnico di Torino, Italy. He has held visiting positions at various institutions, including Kyoto University, University of Illinois at Urbana-Champaign, German Aerospace Research Organization in Oberpfaffenhofen and Columbia University in New York.

Dr. Tempo's research activities are mainly focused on analysis and design of complex systems with uncertainty, and related control applications. He is author or co-author of more than 150 research papers published in international journals, books and conferences. He is also a co-author of the book "Randomized Algorithms for Analysis and Control of Uncertain Systems," Springer-Verlag, 2005. He is a recipient of the "Outstanding Paper Prize Award" from the International Federation of Automatic Control (IFAC) for a paper published in Automatica, and of the "Distinguished Member Award" from the IEEE Control Systems Society. He is a Fellow of the IEEE and a Fellow of the IFAC.

Roberto Tempo is currently an Editor and Deputy Editor-in-Chief of Automatica. He is also Editor for Technical Notes and Correspondence of the IEEE Transactions on Automatic Control. He has served as Program Chair of the first joint IEEE Conference on Decision and Control and European Control Conference, Seville, Spain, 2005, and he is the 2009 President-Elect of the IEEE Control Systems Society.