Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Probabilistic sorting and stabilization of switched systems

Hideaki Ishii [*] **Roberto Tempo** [**]

[*] *Department of Computational Intelligence and Systems Science Tokyo Institute of Technology, Yokohama 226-8502, Japan*
[**] *IEIIT-CNR, Politecnico di Torino, 10129 Torino, Italy*
Emails: ishii@dis.titech.ac.jp, roberto.tempo@polito.it

Abstract: 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. We show that the class of Las Vegas randomized algorithms is useful in the design.

Keywords: Switched systems, Lyapunov function, Randomized algorithms.

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 switched systems is a difficult issue and has been studied extensively; see, e.g., (Liberzon 2003, Sun and Ge 2005). There, 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. In the design of switching rules, once stability is guaranteed, we can focus on performance measures other than stability.

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., (Shorten and Narendra 2002, Cheng *et al.* 2003, Gurvits *et al.* 2007). However, though LMIs can be solved using efficient algorithms, as the number of subsystems increases, the computation often becomes intractable.

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, after 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, but we believe that they have not been studied much in the literature.

Here, we formulate a switching control problem 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 its control performance.

Specifically, we consider two problems as follows: (1) To introduce an order among the subsystems based on their relative stability levels with respect to the common Lyapunov function. (2) 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. To order the subsystems, one needs to compare the subsystems to each other, while to find the largest subset may require examining LMIs for every possible subset.

For these two problems, in this paper, we take a probabilistic approach. In recent years, methods based on randomized algorithms have been successfully developed for analysis and synthesis of uncertain systems (e.g., (Vidyasagar 1998), (Tempo *et al.* 2005)) and hybrid systems (e.g., (Liberzon and Tempo 2004, Ishii *et al.* 2005)). For control problems 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 lower.

In this regard, one 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 and Raghavan 1995). This type, in contrast, always produces correct outputs with probability one.

In particular, for the first problem of 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

[⋆] This work was supported in part by the Ministry of Education, Culture, Sports, Science and Technology, Japan, under Grant-in-Aid for Scientific Research No. 17760344.

real numbers very efficiently (Knuth 1998). An analysis on the complexity of the algorithm is provided as well. Regarding the second problem, as mentioned above, there is a combinatorial issue. While conventional deterministic approaches may result in exponential running time, we propose an efficient Monte Carlo type algorithm. Furthermore, we discuss an approach for the generation of random matrices necessary there.

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 an example in Section 5. The paper is concluded in Section 6.

2. PROBLEM FORMULATION

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 Σ_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 set of indices 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 here is to find the admissible set Σ_0 and a rule to determine a subsystem among the admissible systems at each time t that is optimal from a control performance viewpoint. This type of setups may arise when systems have multiple tuning parameters which are discrete and whose range depends on external factors such as the systems environment.

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

The first one 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 $V(x) = x^T P x$ with a positive-definite matrix $P \in \mathbb{R}^{n \times n}$ which is a common Lyapunov function for all the subsystems, that is,

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

where $L(P,A) := A^T P + PA$. 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. More specifically, for each subsystem i, we look for the subset $\overline{\mathcal{A}}_i$ of systems having slower decay rates given by

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

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

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

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 $\mathcal{A}_{\Sigma(t)} := \{A_i : i \in \Sigma(t)\}$ and $|\cdot|$ denotes cardinality.

The second problem is to find the admissible set Σ_0 such that a common Lyapunov function exists. It is formulated as follows: Find the set Σ_0 with the maximum cardinality among those having a common Lyapunov function satisfying the condition (2), that is,

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

This problem can clearly 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 a sorting 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 $i \in \mathcal{I}_N$:

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

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

For the scalar case, the problem is the usual sorting, where 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 and Raghavan 1995). Here, we propose a matrix version of the RQS. It is a natural and elegant extension of the original RQS, which allows to study worst-case and average-case complexity. As in the RQS for scalars, the randomization is performed under the uniform distribution. Because of no prior knowledge on the matrices, each time a matrix is randomly chosen from a set, equal probabilities are allocated to the elements.

3.1 Matrix Randomized Quick Sort

The RQS algorithm for symmetric matrices is divided into two phases: The first is based on quick sort ideas, and the second performs extra comparisons among the matrices.

We now present the first phase of the algorithm. This phase generates the ternary tree structure shown in Fig. 1.

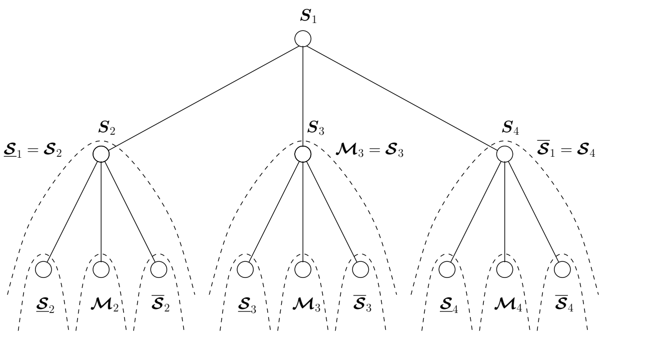
Algorithm 3.1. (*Phase 1 of the matrix RQS*)

(1) Set $\mathcal{S}_1 := \mathcal{X}$ and $k = 1$.
(2) At step k, if $|\boldsymbol{\mathcal{S}}_k| > 1$, then do the following, and otherwise go to step 3.
    (a) Take $\boldsymbol{S}_k \in \boldsymbol{\mathcal{S}}_k$ randomly according to the uniform distribution. This is called the *pivot* matrix.
    (b) Compare each matrix in $\boldsymbol{\mathcal{S}}_k$ with $\boldsymbol{S}_k$ and determine the following three sets:
$$\underline{\boldsymbol{\mathcal{S}}}_k := \{X \in \boldsymbol{\mathcal{S}}_k :\ X \le \boldsymbol{S}_k,\ X \ne \boldsymbol{S}_k\},$$
$$\overline{\boldsymbol{\mathcal{S}}}_k := \{X \in \boldsymbol{\mathcal{S}}_k :\ \boldsymbol{S}_k \le X,\ X \ne \boldsymbol{S}_k\},$$
$$\boldsymbol{\mathcal{M}}_k := \boldsymbol{\mathcal{S}}_k \setminus \left(\underline{\boldsymbol{\mathcal{S}}}_k \cup \overline{\boldsymbol{\mathcal{S}}}_k \cup \{\boldsymbol{S}_k\}\right).$$
    The set $\boldsymbol{\mathcal{M}}_k$ 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 $\boldsymbol{\mathcal{S}}_l = \underline{\boldsymbol{\mathcal{S}}}_k$, $\boldsymbol{\mathcal{S}}_{l+1} = \boldsymbol{\mathcal{M}}_k$, and $\boldsymbol{\mathcal{S}}_{l+2} = \overline{\boldsymbol{\mathcal{S}}}_k$.
(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 in (8) as follows:
    Let $\underline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{S}_k}$ be the set containing $\underline{\boldsymbol{\mathcal{S}}}_k$, 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 $\boldsymbol{S}_k \in \overline{\boldsymbol{\mathcal{S}}}_l$, and the ancestor $\boldsymbol{S}_l$ itself.
    Let $\overline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{S}_k}$ be the set containing $\overline{\boldsymbol{\mathcal{S}}}_k$, 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 $\boldsymbol{S}_k \in \underline{\boldsymbol{\mathcal{S}}}_r$, 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: $\underline{\mathcal{X}}_{X_i} = \underline{\boldsymbol{\mathcal{Y}}}_{X_i}$ and $\overline{\mathcal{X}}_{X_i} = \overline{\boldsymbol{\mathcal{Y}}}_{X_i}$, $i \in \mathcal{I}_N$. 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 $\underline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{S}_k}$ and $\overline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{S}_k}$, 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:
    - $\underline{\boldsymbol{\mathcal{S}}}_k$ 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 $\boldsymbol{S}_k \in \overline{\boldsymbol{\mathcal{S}}}_l$. 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 mid sub-tree of $\boldsymbol{S}_l$ are in $\underline{\mathcal{X}}_{\boldsymbol{S}_k}$.
    - Let $\boldsymbol{S}_l$ be an ancestor of $\boldsymbol{S}_k$ with $\boldsymbol{S}_k \in \boldsymbol{\mathcal{M}}_l$. Then, some matrices in 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.


$\boldsymbol{S}_1$
$\boldsymbol{S}_2$
$\underline{\boldsymbol{\mathcal{S}}}_1 = \boldsymbol{\mathcal{S}}_2$
$\boldsymbol{S}_3$
$\boldsymbol{\mathcal{M}}_3 = \boldsymbol{\mathcal{S}}_3$
$\boldsymbol{S}_4$
$\overline{\boldsymbol{\mathcal{S}}}_1 = \boldsymbol{\mathcal{S}}_4$
$\underline{\boldsymbol{\mathcal{S}}}_2$
$\boldsymbol{\mathcal{M}}_2$
$\overline{\boldsymbol{\mathcal{S}}}_2$
$\underline{\boldsymbol{\mathcal{S}}}_3$
$\boldsymbol{\mathcal{M}}_3$
$\overline{\boldsymbol{\mathcal{S}}}_3$
$\underline{\boldsymbol{\mathcal{S}}}_4$
$\boldsymbol{\mathcal{M}}_4$
$\overline{\boldsymbol{\mathcal{S}}}_4$
Fig. 1. Ternary tree structure

It is clear now that further comparisons are necessary especially between matrices in left/right sub-trees and those in mid 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 $\boldsymbol{\mathcal{M}}_l$ with those in $\underline{\boldsymbol{\mathcal{S}}}_l$ and $\overline{\boldsymbol{\mathcal{S}}}_l$. This can be 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 mid child, the left child, and then the node itself in a recursive manner.
(2) When visiting the node l, if $\boldsymbol{\mathcal{M}}_l$ exists, then do the following, and otherwise go to step 3.
    (a) For each matrix $\boldsymbol{M} \in \boldsymbol{\mathcal{M}}_l$, compare it with the elements in $\underline{\boldsymbol{\mathcal{S}}}_l$ and $\overline{\boldsymbol{\mathcal{S}}}_l$.
    (b) Based on the results, update the sets $\underline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{M}}$, $\overline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{M}}$, and also $\underline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{X}}$, $\overline{\boldsymbol{\mathcal{Y}}}_{\boldsymbol{X}}$ for each $\boldsymbol{X} \in \underline{\boldsymbol{\mathcal{S}}}_l \cup \overline{\boldsymbol{\mathcal{S}}}_l$.
(3) Stop if the current node l is the root. Otherwise, visit the next node, and goto step 2.

For these algorithms, the following worst-case complexity result holds. Since this complexity is quadratic, 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
$$\underline{\mathcal{X}}_{X_i} = \underline{\boldsymbol{\mathcal{Y}}}_{X_i},\quad \overline{\mathcal{X}}_{X_i} = \overline{\boldsymbol{\mathcal{Y}}}_{X_i},$$
$$\mathcal{N}_{X_i} = \mathcal{X} \setminus \left(\underline{\boldsymbol{\mathcal{Y}}}_{X_i} \cup \overline{\boldsymbol{\mathcal{Y}}}_{X_i} \cup \{X_i\}\right)$$
for each $i \in \mathcal{I}_N$. The total number of comparisons is no greater than $N(N-1)/2$.

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 and Raghavan 1995). As mentioned in the Introduction, this class has not been exploited yet in the field of control.

3.2 Analysis on the number of comparisons

We now 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 (Motwani and Raghavan 1995), we measure 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

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

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

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 coincides with the scalar case, and the expected number C_{ave} of comparisons can be computed explicitly as shown in (Mitzenmacher and Upfal 2005):

$$C_{\text{ave}} = 2(N+1) \sum_{i=1}^{N} \frac{1}{k} - 4N = 2(N+1) \ln N + \Theta(1).$$

The other extreme is when there is no order at all among the matrices in $\mathcal{X}$. In this case, we 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 C_{ave} becomes the worst-case number $N(N-1)/2$. In these two extreme cases, Phase 2 is unnecessary.

Other cases fall between these two extreme cases. However, in Phase 2, some of the comparisons that did not occur in Phase 1 must be carried out. As shown in Proposition 3.3, the total number is always less than or equal to the worst-case number $N(N-1)/2$. The number of comparisons in Phase 2 is difficult to estimate.

One question of interest is, what are the characteristics of the trees that require a small number of comparisons? Working on some examples, we can deduce that 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, depending on the result of randomization in each run, from the same set $\mathcal{X}$ of matrices, different trees with and without such features can be produced.

4. FINDING A SUBSET OF SYSTEMS AND A COMMON LYAPUNOV FUNCTION

We now consider the problem stated in (6), which is to find the largest subset of systems having a common Lyapunov function. The proposed algorithm requires random matrix generation; discussion on some methods is also provided.

4.1 A randomized approach

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 $V(x) = x^T P x$ 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 α. 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

$$\mathcal{B}_+ := \left\{ P \in \mathbb{R}^{n \times n} : \; P = P^T > 0, \; \|P\| \leq 1 \right\}, \qquad (9)$$

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 probabilistic approach, we assume that the matrix $\boldsymbol{P} \in \mathcal{B}_+$ is a random matrix. For simplicity, we employ the uniform density function $f_{\boldsymbol{P}}(\cdot)$ associated to P having $\mathcal{B}_+$ as the support set. Let $\text{Prob}_{\boldsymbol{P}}(\cdot)$ 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:

$$\hat{\boldsymbol{J}}_{\max} := \max_{j=1,2,\ldots,M} J(\boldsymbol{P}^{(j)}), \qquad (10)$$

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 $\hat{\boldsymbol{J}}_{\max}$ is determined by a finite number of samples drawn from the set $\mathcal{B}_+$.

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

Proposition 4.1. Let $\epsilon, \delta \in (0,1)$. If the sample size M is such that $M \geq \lceil \log(1/\delta) / \log[1/(1-\epsilon)] \rceil$, then, with probability greater than $1-\delta$, the empirical maximum $\hat{\boldsymbol{J}}_{\max}$ in (10) satisfies the following inequality

$$\text{Prob}_{\boldsymbol{P}} \left\{ J(\boldsymbol{P}) \leq \hat{\boldsymbol{J}}_{\max} \right\} \geq 1 - \epsilon.$$

That is, $\text{Prob}_{\boldsymbol{P}^{(1,\ldots,M)}} \left\{ \text{Prob}_{\boldsymbol{P}} \left\{ J(\boldsymbol{P}) \leq \hat{\boldsymbol{J}}_{\max} \right\} \geq 1 - \epsilon \right\} > 1 - \delta$, where $\text{Prob}_{\boldsymbol{P}^{(1,\ldots,M)}}(\cdot)$ 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* ϵ with *confidence* $1-\delta$ given that the sample size M satisfies the bound. We emphasize that, the sample size M is not dependent on the dimension of the matrices in the set $\mathcal{B}_+$ but only on ϵ and δ.

The approach here is reminiscent of the learning theoretic methods proposed in (Vidyasagar 1998), where the design parameters (the matrix P in our case) are randomized in order to solve an optimization problem arising in the context of hybrid systems. We note that Proposition 4.1 was originally developed in (Tempo et al. 1997) for problems of uncertain systems.

4.2 Generation of random matrices

In this section, we discuss the generation of random matrices required in the computation of the empirical maximum in (10). We focus on generating matrices uniformly in the unit ball $\mathcal{B}_+$ of positive-definite matrices in (9) [1].

We remark that the choice of the norm employed in the ball (9) determines the computation method; see, e.g., (Calafiore and Dabbene 2002),(Tempo et al. 2005). Moreover, the norm changes the probability density of the random matrices and, in turn, the performance in calculating the empirical maximum. We present a specific algorithm for the case with the Frobenius norm, which is based upon the hit-and-run method.

For a symmetric matrix $Z \in \mathbb{R}^{n\times n}$, the Frobenius norm is defined as $\|Z\| := (\text{Tr } Z^2)^{1/2}$. Note that $\|Z\| = \|[\zeta_1^T \ \cdots \ \zeta_n^T]^T\|$, where ζ_i, $i = 1, \dots, n$, are the columns of Z and the vector norm is the Euclidean norm in $\mathbb{R}^{n^2}$. Also, define the inner product $\langle \cdot, \cdot \rangle$ by $\langle X, Y \rangle = \text{Tr } XY$ for symmetric matrices X, Y.

Let the unit ball $\mathcal{B}$ of symmetric (but not necessarily positive-definite) matrices be $\mathcal{B} := \{Z \in \mathbb{R}^{n\times n} : Z = Z^T, \ \|Z\| \le 1\}$. Clearly, we have $\mathcal{B}_+ \subset \mathcal{B}$.

We next describe that the generation of random matrices can be reduced to that of random vectors. Let $\bar{n} := n(n+1)/2$. Also, denote by $\mathcal{B}_{\mathbb{R}^{\bar{n}}}$ the unit ball with center 0 in $\mathbb{R}^{\bar{n}}$ under the Euclidean norm. Then, we define the mapping g from $\mathbb{R}^{\bar{n}}$ to the set of symmetric matrices in $\mathbb{R}^{n\times n}$ as

$$g(z) := \begin{bmatrix} z_1 & \frac{1}{\sqrt{2}} z_2 & \cdots & \cdots & \frac{1}{\sqrt{2}} z_n \\ * & z_{n+1} & \frac{1}{\sqrt{2}} z_{n+2} & \cdots & \frac{1}{\sqrt{2}} z_{2n-1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & z_{\bar{n}-2} & \frac{1}{\sqrt{2}} z_{\bar{n}-1} \\ * & \cdots & \cdots & * & z_{\bar{n}} \end{bmatrix},$$

where $z = [z_1 \ \cdots \ z_{\bar{n}}]^T \in \mathbb{R}^{\bar{n}}$ and the lower-triangular entries are determined by the symmetry of the matrix. With g, we can verify the isomorphism between $\mathcal{B}_{\mathbb{R}^{\bar{n}}}$ and $\mathcal{B}$. Furthermore, the mapping is norm preserving. Hence, generating matrices uniformly in $\mathcal{B}_+$ is equivalent to obtaining uniform vectors in the corresponding set contained in $\mathcal{B}_{\mathbb{R}^{\bar{n}}}$. In fact, computing uniform vectors in $\mathcal{B}_{\mathbb{R}^{\bar{n}}}$ can be done easily (e.g., (Tempo et al. 2005)); this is exploited in the algorithm to be presented.

Noticing that $\mathcal{B}_+$ is a convex subset of $\mathcal{B}$, we employ the so-called hit-and-run algorithm. It is a sampling technique based on Markov chain for generating uniformly distributed real vectors in a convex set. The asymptotic distribution of the chain is uniform. Further, it is guaranteed that this distribution is reached after a given number of steps, which is called the burn-in period T.

The hit-and-run algorithm outlined below provides a matrix $\boldsymbol{P}$ whose distribution is asymptotically uniform in $\mathcal{B}_+$. In particular, it is based on (Lovász and Vempala 2006).

[1] We note that in fact the matrix generation is on the closure of the ball $\mathcal{B}_+$. However, since the boundary points have measure zero, they are irrelevant in the context of this paper.

Algorithm 4.2. (*Hit-and-run algorithm for generating uniform matrices in $\mathcal{B}_+$ under the Frobenius norm*)

```
(1) Set $P_1 := (\sqrt{n}+1)^{-1} I \in \mathcal{B}_+$ and $k = 1$.
(2) At step $k$, generate a matrix $\boldsymbol{S}_k$ uniformly distributed in $\mathcal{B}$ as follows:
    (a) Let $\boldsymbol{q}_k$ be a vector in $\mathbb{R}^{\bar{n}}$ whose entries are independent Gaussian real numbers with mean 0 and variance 1.
    (b) Generate a random real number $\boldsymbol{t}_k$ according to the uniform distribution in $[0, 1]$.
    (c) Set $\boldsymbol{S}_k = g(\boldsymbol{t}_k^{1/\bar{n}} \boldsymbol{q}_k / \|\boldsymbol{q}_k\|)$.
(3) Let $\boldsymbol{W}_{k,0}, \boldsymbol{W}_{k,1}$ be the matrices that are (i) on the line going through $\boldsymbol{P}_k$ in the direction of $\boldsymbol{S}_k$ and (ii) on the boundary of $\mathcal{B}_+$.
(4) Generate a random real number $\boldsymbol{y}_k$ according to the uniform distribution in $[0, 1]$.
(5) Let $\boldsymbol{P}_{k+1}$ be a convex combination of $\boldsymbol{W}_{k,0}$ and $\boldsymbol{W}_{k,1}$ as $\boldsymbol{P}_{k+1} := \boldsymbol{y}_k \boldsymbol{W}_{k,0} + (1-\boldsymbol{y}_k)\boldsymbol{W}_{k,1}$.
(6) Set $k = k+1$. Stop if the burn-in period is reached ($k = T$) and return $\boldsymbol{P}_k$. Otherwise, go to step $k$ in 2.
```

A special feature here is that the initial matrix P_1 is fixed. In typical hit-and-run algorithms, additional processing is required for obtaining an initial vector having a certain distribution. We note that the burn-in period is of order $O(n^7 \ln n)$ (Lovász and Vempala 2006).

5. NUMERICAL EXAMPLE

For the system in (1), we used 30 subsystems, i.e., $N = 30$. The matrices A_i, $i \in \mathcal{I}_N$, were chosen as

$$\begin{aligned} A_i &= A^* + K_i \\ &:= \begin{bmatrix} -1.5 & -2 \\ 2 & 0.5 \end{bmatrix} + 0.1\sqrt{i} \begin{bmatrix} 1 & 0.3\, r_{i,1} \\ 1.5\, r_{i,2} & 0.5\, r_{i,3} \end{bmatrix}, \quad i \in \mathcal{I}_N, \end{aligned}$$

where $r_{i,j}$ are random numbers uniformly distributed on $[-1, 1]$ for $j = 1, 2, 3$.

For this particular system, several conventional approaches failed to provide a common Lyapunov function. First, since $\|sI - A^*\|_\infty \le 2.30$, by a small-gain type argument (Liberzon 2003), a sufficient condition for the existence of a common Lyapunov function is given by $\max_i \|K_i\| < 1/2.30 = 0.435$. However, we had $\max_i \|K_i\| = 0.913$.

The other method is based on interval matrices: Let $\overline{a}_{jm}$ and $\underline{a}_{jm}$ be, respectively, the maximum and the minimum values of the (j, m) entries of A_i, $i \in \mathcal{I}_N$. Then, the objective is to find a common Lyapunov function for the *vertex* matrices, whose (j, m) entries are either $\overline{a}_{jm}$ or $\underline{a}_{jm}$ for all j, m. The set of systems is now smaller because the number of vertex matrices is $2^4 = 16$, but it provides a conservative solution. In fact, the LMIs for this case were found infeasible.

Next, we used the algorithm in Section 4.1. We found a matrix P for the quadratic common Lyapunov function $V(x)$ in less than 100 iterations in most 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. In one such trial, Phase 1 required 135 comparisons and in Phase 2, there were 54 comparisons. Thus, the total number is 189, which is smaller than the

Table 1. 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	6		16		15
2			17	2	4
3	3		18	2	4
4		2	19	7	1
5	8		20	2	6
6			21	5	
7	5		22	2	7
8			23	1	7
9			24	14	
10		9	25	8	1
11			26	6	1
12		15	27	3	1
13		1	28		
14		10	29	5	3
15			30	8	

worst-case number of $N(N-1)/2 = 435$. The result is summarized in Table 1. For each subsystem i, the number of the subsystems slower than it, that is, the cardinality of the set $\overline{\mathcal{A}}_i$ in (3), is given. Also, the number of the 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 5, 19, 24, 25, and 30, while the slower ones are the subsystems 10, 12, 14, 16, 22, and 23.

The switched system was simulated assuming that every $h = 0.1$ sec, 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 one 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 switching rule is expected to give the fastest decay rate of all possible rules, at least on average. 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 which has the largest number of subsystems faster than itself. This rule will produce trajectories with slow decays.

In Fig. 2, sample paths of the systems are plotted in the state space. 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 will focus on other hybrid systems problems using Las Vegas type algorithms.

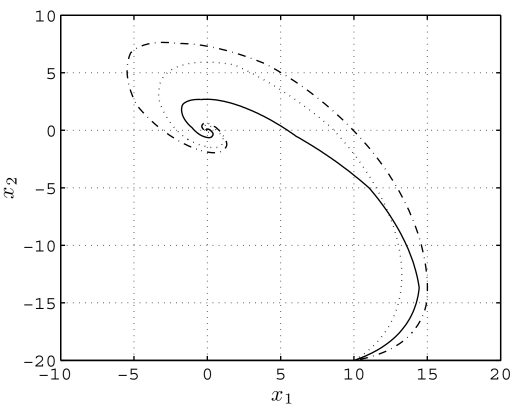


Fig. 2. Sample paths of (x_1, x_2): (a) The random rule (dotted), (b) the fast rule (solid), and (c) the slow rule (dash-dot)

REFERENCES

Calafiore, G. and F. Dabbene (2002). A probabilistic framework for problems with real structured uncertainty in systems and control. *Automatica* **38**, 1265–1276.

Cheng, D., L. Guo and J. Huang (2003). On quadratic Lyapunov functions. *IEEE Trans. Autom. Control* **48**, 885–890.

Gurvits, L., R. Shorten and O. Mason (2007). On the stability of switched positive linear systems. *IEEE Trans. Autom. Control* **52**, 1099–1103.

Ishii, H., T. Başar and R. Tempo (2005). Randomized algorithms for synthesis of switching rules for multimodal systems. *IEEE Trans. Autom. Control* **50**, 754–767.

Knuth, D. E. (1998). *The Art of Computer Programming,* 2nd edition. Vol. 3: Sorting and Searching. Addison-Wesley. Reading, MA.

Liberzon, D. (2003). *Switching in Systems and Control.* Birkhäuser. Boston.

Liberzon, D. and R. Tempo (2004). Common Lyapunov functions and gradient algorithms. *IEEE Trans. Autom. Control* **49**, 990–994.

Lovász, L. and S. Vempala (2006). Hit-and-run from a corner. *SIAM J. Comput.* **35**, 985–1005.

Mitzenmacher, M. and E. Upfal (2005). *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press.

Motwani, R. and P. Raghavan (1995). *Randomized Algorithms.* Cambridge University Press.

Shorten, R. N. and K. S. Narendra (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. *Int. J. Adaptive Control Signal Proc.* **16**, 709–728.

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

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

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

Tempo, R., G. Calafiore and F. Dabbene (2005). *Randomized Algorithms for Analysis and Control of Uncertain Systems.* Springer. London.

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