ESTIMATION AND VERIFICATION OF PARTIALLY OBSERVED DISCRETE-EVENT SYSTEMS

1. INTRODUCTION

State estimation is one of the most fundamental problems in the systems and control theory. In many real-world systems, it is not always possible to access the full information of the system due to measurement noises/uncertainties and the existence of adversaries. Therefore, how to *estimate* the state of the system is crucial when one wants to make decisions based on the limited and incomplete information.

Given a system, one of the most important questions is to prove/disprove the correctness of the system with respect to some desired requirement (or specification). In particular, we would like to check the correctness of the system in a formal manner in the sense that the checking procedures are algorithmic, and results have provable correctness guarantees. Such a formal satisfaction checking problem is referred to as the *verification problem*. Performing formal property verification is very important for many complicated but safety-critical infrastructures.

This article considers state estimation and verification problems for an important class of man-made cyber-physical systems called discrete-event systems (DESs). Roughly speaking, DESs are dynamic systems with discrete state spaces and event-triggered dynamics. DES models are widely used in the study of complex automated systems where the behavior is inherently event driven, as well as in the study of discrete abstractions of continuous, hybrid, and/or cyber-physical systems. Over the past decades, the theory of DES has been successfully applied to many real-world problems, for example, the control of automated systems, fault diagnosis/prognosis, and information-flow security analysis.

The main purpose of this article is to provide a tutorial and an overview of state estimation techniques and their related property verification problems for *partially observed DES*. Specifically, we focus on the verification of *observational properties*, that is, information-flow properties whose satisfactions are based on the observation of the system. The outline of this article is as follows:

- In Section 2, we briefly introduce some necessary terminologies and the partially observed DES model considered in this article.
- In Section 3, we introduce three types of state-estimation problems, namely, current-state estimation, initial-state estimation, and delayed-state estimation. Then, we provide state estimation techniques for different state-estimation problems.
- In Section 4, we introduce several important observational properties that arise in the analysis of partially observed DES. In particular, we consider detectability, diagnosability, prognosability, distinguishability, and opacity, which cover most of the important properties in partially observed DES. One feature of this section is that we study all these properties in a uniform manner by defining them in terms of state estimates introduced in Section 3.
- In Section 5, we provide verification procedures for all properties introduced in Section 4. All verification procedures are summarized as detailed algorithms using state estimation techniques provided in Section 3.
- In Section 6, some related problems and further readings on estimation and verification of partially observed DES are provided.

The entire article is self-contained. Related references are provided in each section.

2. PARTIALLY OBSERVED DISCRETE-EVENT SYSTEMS

This section provides the basic model of partially observed DES and some related terminologies. The reader is referred to the textbook (1) for more details on DES.

Let Σ be a finite set of events. A string $s = \sigma_1 \ldots \sigma_n, \sigma_i \in \Sigma$ is a finite sequence of events. We denote by Σ^* the set of all strings over Σ including the empty string ϵ. For any string $s \in \Sigma^*$, we denote by $|s|$ its length with $|\epsilon| = 0$. A language $L \subseteq \Sigma^*$ is a set of strings. For any string $s \in L$ in language L, we denote by L/s the postlanguage of s in L, that is, $L/s := \{w \in \Sigma^* : sw \in L\}$. The prefix closure of language L is defined by $\overline{L} = \{s \in \Sigma^* : \exists w \in \Sigma^* \text{ s.t. } sw \in L\}$; L is said to be prefix closed if $L = \overline{L}$.

A DES is modeled as a nondeterministic finite-state automaton (NFA)

$$G = (X, \Sigma, \delta, X_0) \tag{1}$$

where X is a finite set of states; Σ is a finite set of events; $\delta : X \times \Sigma \to 2^X$ is the (partial) nondeterministic transition function, where for any $x, x' \in X, \sigma \in \Sigma$, $x' \in \delta(x, \sigma)$ means that there exists a transition from state x to state x' with event label σ; and $X_0 \subseteq X$ is the set of initial states. The transition function is also extended to $\delta : X \times \Sigma^* \to 2^X$ recursively by: (i) $\delta(x, \epsilon) = \{x\}$ and (ii) for any $x \in X, s \in \Sigma^*, \sigma \in \Sigma$, we have $\delta(x, s\sigma) = \cup_{x' \in \delta(x,s)} \delta(x', \sigma)$. For the sake of simplicity, we write $\delta(s) = \cup_{x_0 \in X_0} \delta(x_0, s)$. We define $\mathcal{L}(G, x) = \{s \in \Sigma^* : \delta(x, s)!\}$ as the set of strings that can be generated by G from state $x \in X$, where "!" means "is defined". We define $\mathcal{L}(G) := \cup_{x_0 \in X_0} \mathcal{L}(G, x_0)$ the language generated by system G. System G is a deterministic finite-state automaton (DFA) if (i) $|X_0| = 1$ and (ii) $\forall x \in X, \sigma \in \Sigma : |\delta(x, \sigma)| \leq 1$. We say that a sequence of state $x_0 x_1 \ldots x_n$ forms a *cycle* in G if (i) $\forall i = 0, \ldots, n-1, \exists \sigma \in \Sigma : x_{i+1} \in \delta(x_i, \sigma)$ and (ii) $x_0 = x_n$.

Let $G_1 = (X_1, \Sigma_1, \delta_1, X_{0,1})$ and $G_2 = (X_2, \Sigma_2, \delta_2, X_{0,2})$ be two automata. The *product* of G_1 and G_2, denoted by $G_1 \times G_2$, is defined as the accessible part of a new NFA

$$G_1 \times G_2 = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \delta_{1,2}, X_{0,1} \times X_{0,2}) \tag{2}$$

where the transition function is defined by: for any $(x_1, x_2) \in X_1 \times X_2$, and for any $\sigma \in \Sigma_1 \cap \Sigma_2$, we have

$$\delta_{1,2}((x_1, x_2), \sigma) = \begin{cases} \delta_1(x_1, \sigma) \times \delta_2(x_2, \sigma) & \text{if } \sigma \in \Sigma_1 \cap \Sigma_2 \\ \text{undefined} & \text{otherwise} \end{cases} \tag{3}$$

J. Webster (ed.), *Wiley Encyclopedia of Electrical and Electronics Engineering*.
DOI: 10.1002/047134608X.W8400

In the partial-observation setting, not all events generated by the system can be observed. Formally, we assume that the event set is partitioned as follows:

$$\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo} \tag{4}$$

where Σ_o is the set of observable events; and Σ_{uo} is the set of unobservable events. The natural projection from Σ to Σ_o is a mapping $P : \Sigma^* \to \Sigma_o^*$ defined recursively as follows:

$$P(\epsilon) = \epsilon \text{ and } P(s\sigma) = \begin{cases} P(s)\sigma & \text{if } \sigma \in \Sigma_o \\ P(s) & \text{if } \sigma \notin \Sigma_o \end{cases} \tag{5}$$

That is, for any string $s \in \Sigma^*$, $P(s)$ is obtained by erasing all unobservable events in it. We denote by $P^{-1} : \Sigma_o^* \to 2^{\Sigma^*}$ the inverse projection, that is, for any $\alpha \in \Sigma_o^*$, we have $P^{-1}(\alpha) = \{s \in \Sigma^* : P(s) = \alpha\}$. Note that the codomain of P^{-1} is 2^{Σ^*} as the inverse projection of a string is not unique in general. We also extend the natural projection to $P : 2^{\Sigma^*} \to 2^{\Sigma_o^*}$ by: for any $L \subseteq \Sigma^*$, $P(L) = \{P(s) \in \Sigma_o^* : s \in L\}$. The inverse mapping is also extended to $P^{-1} : 2^{\Sigma_o^*} \to 2^{\Sigma^*}$ analogously.

Hereafter, we make the following standard assumptions in the analysis of partially observed DES:

A1 System G is live, that is, $\forall x \in X, \exists \sigma \in \Sigma : \delta(x,\sigma)!$ and
A2 System G does not contain an unobservable cycle, that is, $\forall x, x' \in X, s \in \Sigma_{uo}^* \backslash \{\epsilon\} : x' \notin \delta(x,s)$.

3. STATE ESTIMATION UNDER PARTIAL OBSERVATION

3.1. State-Estimation Problems

Since the system is partially observed, one important question is what do we know about the system's state when a (projected) string is observed? This is referred to as the *state-estimation problem* in the system's theory.

One of the most fundamental state-estimation problems is the current-state-estimation problem, that is, we want to estimate all possible states the system can be in currently based on the observation.

Definition 1 (Current-State Estimate) Let G be a DES with observable events $\Sigma_o \subseteq \Sigma$ and $\alpha \in P(\mathcal{L}(G))$ be an observed string. The current-state estimate upon the occurrence of α, denoted by $\hat{X}_G(\alpha)$, is the set of states the system could be in currently based on observation α, that is,

$$\hat{X}_G(\alpha) = \{x \in X : \exists x_0 \in X_0, s \in \mathcal{L}(G, x_0) \text{ s.t. } x \in \delta(x_0, s) \wedge P(s) = \alpha\} \tag{6}$$

Instead of knowing the current state of the system, in some applications, one may also be interested in knowing which initial states the system may start from. This is referred to as the initial-state-estimation problem defined as follows.

Definition 2 (Initial-State Estimate) Let G be a DES with observable events $\Sigma_o \subseteq \Sigma$ and $\alpha \in P(\mathcal{L}(G))$ be an observed string. The initial-state estimate upon the occurrence of α, denoted by $\hat{X}_{0,G}(\alpha)$, is the set of initial states the system could start from initially based on observation α, that is,

$$\hat{X}_{0,G}(\alpha) = \{x_0 \in X_0 : \exists s \in \mathcal{L}(G, x_0) \text{ s.t. } P(s) = \alpha\} \tag{7}$$

Note that, in the current-state estimate, we use the observation up to the current instant to estimate the set of all possible states the system can be in at this instant. If we keep observing more events in the future, our knowledge of the system's state at that instant may be further improved as some states in the current-state estimate may not be consistent with the future observation. In other words, we can use future information to further improve our knowledge about the system's state for some previous instant. This is also referred to as the "smoothing" process; such a smoothed state estimate is also called the delayed-state estimate defined as follows.

Definition 3 (Delayed-State Estimate) Let G be a DES with observable events $\Sigma_o \subseteq \Sigma$ and $\alpha\beta \in P(\mathcal{L}(G))$ be an observed string. The delayed-state estimate for the instant of α upon the occurrence of $\alpha\beta$, denoted by $\hat{X}_G(\alpha \mid \alpha\beta)$, is the set of states the system could be in $|\beta|$ steps ago when $\alpha\beta$ is observed, that is,

$$\hat{X}_G(\alpha \mid \alpha\beta) = \left\{ x \in X : \begin{array}{c} \exists x_0 \in X_0, sw \in \mathcal{L}(G, x_0) \text{s.t.} \\ P(s) = \alpha \wedge P(sw) = \alpha\beta \wedge x \in \delta(x_0, s) \end{array} \right\} \tag{8}$$

Example 1 Let us consider system G shown in Figure 1a, where $\Sigma_o = \{a, b, c\}$ and $X_0 = \{0, 2\}$. Let us consider the observable string $aa \in P(\mathcal{L}(G))$. We have $P^{-1}(aa) \cap \mathcal{L}(G) = \{aa, uuaa\}$. Then we know that the system may start from state 0 or 2, that is, $\hat{X}_{0,G}(aa) = \{0, 2\}$. Also, the system may be currently in states 4 or 6, that is, $\hat{X}_G(aa) = \{4, 6\}$. From string aa, if we further observe event c in the next instant, then we know that $\hat{X}_G(aa \mid aac) = \{6\}$, since event c cannot occur from state 4.

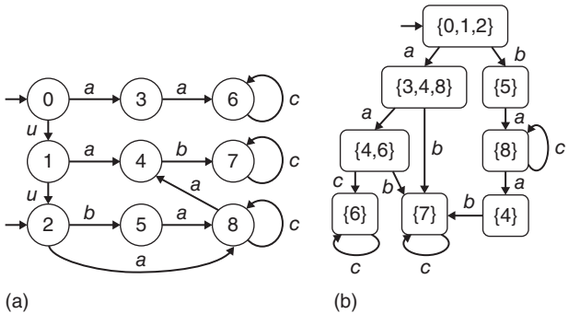

Figure 1. An arrow labeled with event σ from state x to state x' means that $x' \in \delta(x, \sigma)$. States with arrows having no predecessor state denote initial states. (a) System G with $\Sigma_o = \{a, b, c\}$; (b) Obs(G).

3.2. State Estimation Techniques

In this section, we provide techniques for computing different notions of state estimates.

Computation of Current-State Estimate. The general idea for computing the current-state estimate is to construct a structure that tracks all possible states consistent with the current observation. This construction is well known as the *subset construction* technique that can be used to convert an NFA to a DFA. In the DES literature, this structure is usually referred to as the *observer* automaton.

Definition 4 (Observer) Given system $G = (X, \Sigma, \delta, X_0)$ with observable events $\Sigma_o \subseteq \Sigma$, the observer is a new DFA

$$\text{Obs}(G) = (X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0}) \tag{9}$$

where $X_{\text{obs}} \subseteq 2^X \setminus \emptyset$ is the set of states, $x_{\text{obs},0} = \{x \in X : \exists x_0 \in X_0, w \in \Sigma_{uo}^* \text{s.t. } x \in \delta(x_0, w)\}$ is the unique initial state, and δ_{obs} is the deterministic transition function defined by: for any $q \in X_{\text{obs}}, \sigma \in \Sigma_o$, we have

$$\delta_{\text{obs}}(q, \sigma) = \{x' \in X : \exists x \in q, \exists w \in \Sigma_{uo}^* \text{s.t. } x' \in \delta(x, \sigma w)\} \tag{10}$$

For the sake of simplicity, we will only consider the accessible part of Obs(G).

Intuitively, the observer state tracks all possible states the system can be in currently based on the observation. Formally, we have the following result.

Proposition 1 For any system G with $\Sigma_o \subseteq \Sigma$, its observer Obs($G$) has the following properties:

1. $\mathcal{L}(\text{Obs}(G)) = P(\mathcal{L}(G))$ and
2. For any $\alpha \in P(\mathcal{L}(G))$, we have $\hat{X}_G(\alpha) = \delta_{\text{obs}}(x_{\text{obs},0}, \alpha)$.

Therefore, given an observation $\alpha \in P(\mathcal{L}(G))$, its current-state estimate can simply be computed by Algorithm 1. Note that the complexity of building Obs(G) is $O(|\Sigma|2^{|X|})$, which is exponential in the size of G. However, we can update the current-state estimate recursively online, and each update step only requires a polynomial complexity.

Algorithm 1 Current-State Estimation

Inputs: G and $\alpha \in P(\mathcal{L}(G))$
Output: $\hat{X}_G(\alpha)$
1: Build $\text{Obs}(G) = (X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0})$
2: $\hat{X}_G(\alpha) \leftarrow \delta_{\text{obs}}(x_{\text{obs},0}, \alpha)$
3: **return** $\hat{X}_G(\alpha)$

Computation of Initial-State Estimate. We provide two approaches for the computation of initial-state estimate.

The first approach is based on the *augmented automaton* that augments the state space of the original system by tracking where each state starts from. This approach sometimes is also referred to as the trellis-based approach in the literature.

Formally, given a system G, its augmented automaton is a new NFA

$$G_{\text{aug}} = (X_{\text{aug}}, \Sigma, \delta_{\text{aug}}, X_{0,\text{aug}}) \tag{11}$$

where $X_{\text{aug}} \subseteq X_0 \times X$ is the set of states, $\delta_{\text{aug}} : X_{\text{aug}} \times \Sigma \to 2^{X_{\text{aug}}}$ is the transition function defined by: for any $(x_0, x) \in X_{\text{aug}}, \sigma \in \Sigma$, we have $\delta_{\text{aug}}((x_0, x), \sigma) = \{(x_0, x') \in X_0 \times X : x' \in \delta(x, \sigma)\}$, and $X_{0,\text{aug}} = \{(x_0, x_0) \in X_{\text{aug}} : x_0 \in X_0\}$ is the initial state.

We can see easily that $\mathcal{L}(G) = \mathcal{L}(G_{\text{aug}})$, and for each state in G_{aug}, its first component contains its initial-state information and its second component contains its current-state information. Let $\text{Obs}(G_{\text{aug}}) = (X_{\text{obs}}^{\text{aug}}, \Sigma_o, \delta_{\text{obs}}^{\text{aug}}, x_{\text{obs},0}^{\text{aug}})$ be the observer of G_{aug}. We can easily show that

$$\hat{X}_{0,G}(\alpha) = I_0(\delta_{\text{obs}}^{\text{aug}}(x_{\text{obs},0}^{\text{aug}}, \alpha))$$

Algorithm 2 Initial-State-Estimation-Aug

Inputs: G and $\alpha \in P(\mathcal{L}(G))$
Output: $\hat{X}_{0,G}(\alpha)$
1: Build G_{aug}
2: Build $\text{Obs}(G_{\text{aug}}) = (X_{\text{obs}}^{\text{aug}}, \Sigma_o, \delta_{\text{obs}}^{\text{aug}}, x_{\text{obs},0}^{\text{aug}})$
3: $\hat{X}_{0,G}(\alpha) \leftarrow I_0(\delta_{\text{obs}}^{\text{aug}}(x_{\text{obs},0}^{\text{aug}}, \alpha))$
4: **return** $\hat{X}_{0,G}(\alpha)$

where for any $q \in X_{\text{obs}}^{\text{aug}}$, $I_0(q)$ denotes the projection to its first component, that is, $I_0(q) = \{x_0 \in X_0 : \exists x \in X \text{ s.t. } (x_0, x) \in q\}$. This observation suggests Algorithm 2 for computing the initial-state estimate.

The idea of the augmented-automaton-based approach is also very useful for many other purposes, for example, control synthesis for initial-state estimation, as it only uses the dynamic of the system up to the current point. However, the complexity of Algorithm 2 is $O(|\Sigma|2^{|X|^2})$ as the size of G_{aug} is quadratic in the size of G. Here, we provide the second approach for computing the initial-state estimate with a lower complexity based on the *reversed automaton* of G.

For any NFA $G = (X, \Sigma, \delta, X_0)$, its reversed automaton is a new NFA

$$G_{\text{R}} = (X, \Sigma, \delta_R, X) \tag{12}$$

where the transition function $\delta_R : X \times \Sigma \to 2^X$ is defined by: $\forall x, x' \in X, \sigma \in \Sigma : x' \in \delta_R(x, \sigma) \iff x \in \delta(x', \sigma)$. Note that the initial state of G_{R} is the entire state space. For any string $s = \sigma_1\sigma_2 \ldots \sigma_n \in \Sigma^*$, we denote by s_R its reversed string, that is, $s_R = \sigma_n \ldots \sigma_2\sigma_1$. Then, the following result shows that the initial-state estimate of a string in G can be computed based on the current-state estimate of its reversed string in the reversed automaton G_{R}.

Proposition 2 (2) For any $\alpha \in P(\mathcal{L}(G))$, we have $\hat{X}_{0,G}(\alpha) = \hat{X}_{G_{\text{R}}}(\alpha_R) \cap X_0$.

Based on the above result, Algorithm 3 is proposed to compute the initial-state estimate, and its complexity is only $O(|\Sigma|2^{|X|})$.

Algorithm 3 Initial-State-Estimation-Rev

Inputs: G and $\alpha \in P(\mathcal{L}(G))$
Output: $\hat{X}_{0,G}(\alpha)$
1: Build G_{R}
2: Build $\mathrm{Obs}(G_{\mathrm{R}}) = (X^R_{\mathrm{obs}}, \Sigma_o, \delta^R_{\mathrm{obs}}, X)$
3: $\hat{X}_{G_{\mathrm{R}}}(\alpha_R) \leftarrow \delta^R_{\mathrm{obs}}(X, \alpha_R)$
4: **return** $\hat{X}_{0,G}(\alpha) \leftarrow \hat{X}_{G_{\mathrm{R}}}(\alpha_R) \cap X_0$

Computation of Delayed-State Estimate. For any $\alpha\beta \in P(\mathcal{L}(G))$, the delayed-state estimate $\hat{X}_G(\alpha \mid \alpha\beta)$ involves two parts of information: the current information α and the future information β. Recall that we are interested in estimating states for the instant of α. The following result shows that the delayed-state estimate can simply be separated as two parts that do not depend on each other.

Proposition 3 **(3)** For any $\alpha\beta \in P(\mathcal{L}(G))$, we have

$$\hat{X}_G(\alpha \mid \alpha\beta) = \hat{X}_G(\alpha) \cap \hat{X}_{0,G}(\beta) = \hat{X}_G(\alpha) \cap \hat{X}_{G_{\mathrm{R}}}(\beta_R)$$

Based on Proposition 3, Algorithm 4 can be used to compute the delayed-state estimate, and its complexity is also $O(|\Sigma|2^{|X|})$.

Algorithm 4 Delayed-State-Estimation

Inputs: G and $\alpha\beta \in P(\mathcal{L}(G))$
Output: $\hat{X}_G(\alpha \mid \alpha\beta)$
1: Build $\mathrm{Obs}(G) = (X_{\mathrm{obs}}, \Sigma_o, \delta_{\mathrm{obs}}, x_{\mathrm{obs},0})$
2: $\hat{X}_G(\alpha) \leftarrow \delta_{\mathrm{obs}}(x_{\mathrm{obs},0}, \alpha)$
3: Build G_{R}
4: Build $\mathrm{Obs}(G_{\mathrm{R}}) = (X^R_{\mathrm{obs}}, \Sigma_o, \delta^R_{\mathrm{obs}}, X)$
5: $\hat{X}_{G_{\mathrm{R}}}(\beta_R) \leftarrow \delta^R_{\mathrm{obs}}(X, \beta_R)$
6: **return** $\hat{X}_G(\alpha \mid \alpha\beta) \leftarrow \hat{X}_G(\alpha) \cap \hat{X}_{G_{\mathrm{R}}}(\beta_R)$

Example 2 We still consider system G shown in Figure 1a, where $\Sigma_o = \{a, b, c\}$ and $X_0 = \{0, 2\}$. Its observer $Obs(G)$ is shown in Figure 1b. For example, string aa reaches state $\{4, 6\}$ from the initial state in $Obs(G)$. Therefore, we have $\hat{X}_G(aa) = \{4, 6\}$. To perform initial-state estimation and delayed-state estimation, we need to build the reversed automaton G_R and its observer $Obs(G_R)$ shown in Figures 2a and b, respectively. For example, string cba reaches state $\{0, 1, 8\}$ from the initial state in $Obs(G_R)$. Therefore, we have $\hat{X}_{0,G}(abc) = \hat{X}_{G_R}((abc)_R) \cap X_0 = \{0, 1, 8\} \cap \{0, 2\} = \{0\}$, that is, we know for sure that the system was initially from state 0. To compute the delayed-state estimate $\hat{X}_G(a \mid aac)$, we have $\hat{X}_G(a \mid aac) = \hat{X}_G(a) \cap \hat{X}_{G_R}(ca) = \{3, 4, 8\} \cap \{0, 1, 2, 3, 5\} = \{3\}$.

4. PROPERTIES OF PARTIALLY OBSERVED DES

In this section, we discuss several important properties in partially observed DES. As we mentioned before, we will only focus on observational properties.

4.1. Detectability

In the previous section, we have provided algorithms for computing state estimates. Then, the natural question arises as to can the state estimation algorithm (eventually) provide precise state information? This is referred to as the *detectability verification problem*. Here, we consider three most fundamental types of detectability:

Definition 5 **(Detectability (4–6))** Let G be a DES with $\Sigma_o \subseteq \Sigma$. Then, G is said to be

- current-state detectable if $(\exists n \in \mathbb{N})(\forall \alpha \in P(\mathcal{L}(G)) : |\alpha| \geq n)[|\hat{X}_G(\alpha)| = 1]$;
- initial-state detectable if $(\exists n \in \mathbb{N})(\forall \alpha \in P(\mathcal{L}(G)) : |\alpha| \geq n)[|\hat{X}_{0,G}(\alpha)| = 1]$;
- delayed detectable (w.r.t. parameters $k_1, k_2 \in \mathbb{N}$) if $(\forall \alpha\beta \in P(\mathcal{L}(G)) : |\alpha| \geq k_1, |\beta| \geq k_2)[|\hat{X}_G(\alpha \mid \alpha\beta)| = 1]$.

Intuitively, the current-state detectability (respectively, initial-state detectability) requires that the current state (initial state) of the system can always be detected unambiguously within a finite delay. Note that, for initial-state

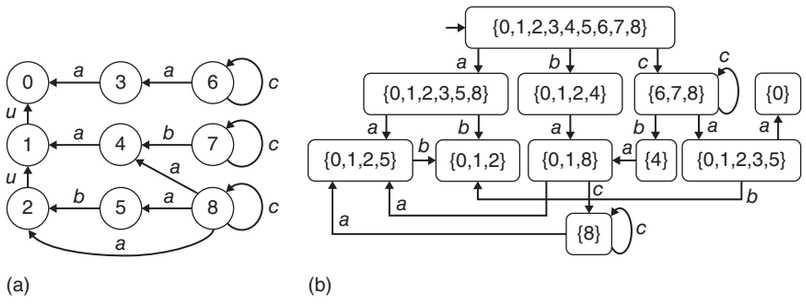

Figure 2. Initial-state estimation using the reversed automaton. Note that all states in G_{R} are initial. (a) G_{R} with $\Sigma_o = \{a, b, c\}$; (b) $\mathrm{Obs}(G_{\mathrm{R}})$.

estimate, once the initial state is detected, we know it for sure forever, which is not the case for current-state estimation. Delayed detectability requires that, for any specific instant after k_1 steps, we can always unambiguously determine the precise state of the system at that instant with at most k_2 steps of information delay. That is, one is allowed to use future information to "smooth" the state estimate of a previous instant.

The concept of detectability was initially proposed by Shu and Lin in Reference 4, which generalizes the concept of observability studied in Reference 7. There are also other notions of detectability proposed in the literature. For example, weak detectability (4) requires that the state of the system can be detected for *some* path generated by the system. Also, Reference 4 proposed the notion of periodic detectability, which requires that the state of the system can be detected periodically. In Reference 8, the authors proposed the notion of K-detectability by replacing the detection condition $|\hat{X}_G(\alpha)| = 1$ as $|\hat{X}_G(\alpha)| \leq K$, where $K \in \mathbb{N}$ is a positive integer specifying the detection precision. A generalized version of detectability, in the sense of detection condition, was proposed in Reference 9. The reader is referred to References 10–16 for more references on detectability.

4.2. Diagnosability and Prognosability of Fault

Another important application of partially observed DES is the fault diagnosis/prognosis problem. In this setting, we assume that system G may have some fault modeled as fault events $\Sigma_F \subseteq \Sigma$. For any string $s \in \Sigma^*$, we write $\Sigma_F \in s$ if s contains a fault event in Σ_F. We define $\Psi(\Sigma_F) := \{se_f \in \mathcal{L}(G) : e_f \in \Sigma_F\}$ as the set of strings that end up with fault events. In the fault diagnosis problem, we assume that all fault events are unobservable; otherwise, it can be diagnosed trivially. Without loss of generality, we can further assume that the state space of G is partitioned as fault states and nonfault states

$$X = X_N \dot{\cup} X_F \tag{13}$$

such that

- $\forall x_0 \in X_0, \forall s \in \mathcal{L}(G, x_0) : \Sigma_F \notin s \Rightarrow \delta(x_0, s) \subseteq X_N$ and
- $\forall x_0 \in X_0, \forall s \in \mathcal{L}(G, x_0) : \Sigma_F \in s \Rightarrow \delta(x_0, s) \subseteq X_F$.

This assumption can be fulfilled by taking the product between G and a new automaton with two states capturing the occurrence of fault; see, for example, Reference 1.

To diagnose the occurrence of fault, the current-state-estimation technique can be applied. Specifically, for any observation $\alpha \in P(\mathcal{L}(G))$, we know that

- the fault has occurred for sure if $\hat{X}_G(\alpha) \subseteq X_F$;
- the fault has not occurred for sure if $\hat{X}_G(\alpha) \subseteq X_N$; and
- the fault may have occurred, but it is uncertain if $\hat{X}_G(\alpha) \cap X_N \neq \emptyset$ and $\hat{X}_G(\alpha) \cap X_F \neq \emptyset$.

Then, the natural question arises as to can we always determine the occurrence of fault within a finite number of delays? This is captured by the notion of diagnosability as follows.

Definition 6 (Diagnosability) System G is said to be diagnosable w.r.t. Σ_o and Σ_F if $(\forall s \in \Psi(\Sigma_F))(\exists n \in \mathbb{N})(\forall t \in \mathcal{L}(G)/s : |t| \geq n)[\hat{X}_G(P(st)) \subseteq X_F]$.

Remark 1 For the sake of simplicity, our definition of diagnosability is based on the current-state estimate and the prespecified fault states. Diagnosability was originally defined by Reference 17 purely based on languages as follows:

$$(\forall s \in \Psi(\Sigma_F))(\exists n \in \mathbb{N})(\forall t \in \mathcal{L}(G)/s : |t| \geq n)$$
$$(\forall w \in P^{-1}(P(st)) \cap \mathcal{L}(G))[\Sigma_F \in w]$$

One can easily check that the language-based definition and the current-state-estimate-based definition are equivalent. Also, in general, the nonfault behavior can be described as a specification language rather than fault events; this formulation can also be transformed to our event-based fault setting by refining the state space of the system; see, for example, Reference 18.

The concept of diagnosability of DES was first introduced in Reference 19, where state-based faults are considered. In Reference 17, the authors introduced the language-based formulation of diagnosability. Since then, many variations of diagnosability have been studied in the literature. For example, model reduction for diagnosability was studied in Reference 20. Diagnosability of repeated/intermittent faults was studied in References 21–23. The reader is referred to the recent survey (24) for more references on diagnosability analysis.

In some applications, we may want to *predict* the occurrence of fault before it actually occurs. This problem is referred to as the *fault prognosis problem*. Still, we assume that $\Sigma_F \subseteq \Sigma$ is the set of fault events and $X_N \subseteq X$ is the set of nonfault states. However, in the fault prognosis problem, a fault event need not be unobservable. To formulate the problem, we define the following two sets of states:

- boundary states, $\partial(G) = \{x \in X_N : \exists e_f \in \Sigma_F \text{ s.t. } \delta(x, e_f)!\}$ and
- indicator states, $\mathfrak{I}(G) = \{x \in X_N : \exists n \in \mathbb{N}, \forall s \in \mathcal{L}(G, x) \text{ s.t. } |s| > n \Rightarrow \Sigma_F \in s\}$.

Intuitively, a boundary state is a nonfault state from which a fault event *may* occur in the next step, and an indicator state is a state from which a fault event will occur *for sure* within a finite number of steps. With the help of indicator states, we can also use current-state-estimation algorithm to perform online fault prognosis. Specifically, for any observation $\alpha \in P(\mathcal{L}(G))$, we know that

- the fault will occur for sure in a finite number of steps if $\hat{X}_G(\alpha) \subseteq \mathfrak{I}(G)$ and
- the fault is not guaranteed to occur within any finite number of steps if $\hat{X}_G(\alpha) \nsubseteq \mathfrak{I}(G)$.

Similarly, the natural question arises as to can we successfully predict the occurrence of fault in the sense that: (i) there is no missed fault and (ii) there is no false alarm? This is captured by the notion of prognosability as follows.

Definition 7 (Prognosability) System G is said to be prognosable w.r.t. Σ_o and Σ_F if $(\forall s \in \mathcal{L}(G) : \delta(s) \cap \partial(G) \neq \emptyset)(\exists t \in \overline{\{s\}})[\hat{X}_G(P(t)) \subseteq \mathfrak{I}(G)]$.

Prognosability is also referred to as predictability in the literature. Intuitively, prognosability requires that, for any string that reaches a boundary state, it has a prefix for which we can claim unambiguously that fault will occur for sure in the future, that is, we can issue a fault alarm. Still, for the sake of simplicity, here we characterize prognosability using current-state estimate, boundary states, and indicator states. Our definition is also equivalent to the language-based definition in References 25 and 26, where predictability was originally introduced. Prognosability also has several variations in the literature. For example, in Reference 27, two performance bounds were proposed to characterize how early a fault alarm can be issued and when a fault is guaranteed to occur once an alarm is issued.

4.3. State Disambiguation and Observability

In some applications, the purpose of state estimation is to distinguish some states. This problem is referred to as the *state disambiguation problem* (28–30). Formally, the specification of this problem is defined as a set of state pairs $T_{\text{spec}} \subseteq X \times X$, and we want to make sure that we can always distinguish between every pair of states in T_{spec}.

Definition 8 (Distinguishability) System G is said to be distinguishable w.r.t. Σ_o and $T_{\text{spec}} \subseteq X \times X$ if $(\forall \alpha \in P(\mathcal{L}(G)))[(\hat{X}_G(\alpha) \times \hat{X}_G(\alpha)) \cap T_{\text{spec}} = \emptyset]$.

Distinguishability is very useful as many important properties in the literature can be formulated as distinguishability. For example, one can check that prognosability can actually be rewritten as distinguishability with specification $T_{\text{spec}} = \partial(G) \times (X_N \backslash \mathfrak{I}(G))$. Observability is another important property in partially observed DES, which together with controllability provide the necessary and sufficient conditions for the existence of a supervisor achieving a desired language. The reader is referred to References 31 and 32 for formal definition of observability. This property is also a special case of distinguishability (possibly after state-space refinement) as it essentially requires that we can distinguish two states at which different control actions are needed; see, for example, Reference 28.

4.4. Opacity

Finally, state estimation is also useful in information-flow security analysis, which is an important topic in cyber-physical systems. In this setting, we assume that the system is also monitored by a passive intruder (eavesdropper) that can observe the occurrences of events in Σ_o. Furthermore, we assume that the system has a "secret" that does not want to be revealed to the intruder. In general, what is a secret is problem dependent. Here, we consider a simple scenario where the secret is modeled as a set of secret states $X_S \subseteq X$. Then, we use the notion of opacity to characterize whether or not the secret can be revealed to the intruder.

Definition 9 (Opacity (2, 3, 33, 35)) Let G be a DES with $\Sigma_o \subseteq \Sigma$ and secret states $X_S \subseteq X$. Then, G is said to be

- current-state opaque if, for any $\alpha \in P(\mathcal{L}(G))$, we have $\hat{X}_G(\alpha) \not\subseteq X_S$;
- initial-state opaque if, for any $\alpha \in P(\mathcal{L}(G))$, we have $\hat{X}_{0,G}(\alpha) \not\subseteq X_S$; and
- infinite-step opaque if, for any $\alpha\beta \in P(\mathcal{L}(G))$, we have $\hat{X}_G(\alpha \mid \alpha\beta) \not\subseteq X_S$.

Essentially, opacity is a confidentiality property capturing the plausible deniability of the system's "secret" in the presence of an outside observer that is potentially malicious. More specifically, current-state opacity (respectively, initial-state opacity) requires that the intruder should never know for sure that the system is currently at (respectively, initially from) a secret state. The system is said to be infinite-step opaque if the intruder can never determine for sure that the system was at a secret state for any specific instant even based on the future information.

Opacity was originally introduced in the computer science literature (36). Then, it was introduced to the framework of DES by References 37–39. There are also many variations of opacity studied in the literature. For example, in Reference 40, Lin formulated opacity using a language-based framework, where the notions of strong opacity and weak opacity are proposed. In Reference 2, a concept called initial-and-final-state opacity was provided; the authors also studied the transformations among several notions of opacity. When one is only allowed to use a bounded delayed information to improve the state estimate for a previous instant (or we do not care about the secret anymore after some delays), infinite-step opacity becomes to K-step opacity (41), where K is a nonnegative integer capturing the delay bound. Opacity is also closely related to another two information-flow security properties called anonymity (42) and noninterference (43, 44). The reader is referred to the survey (45) for more references on opacity.

5. VERIFICATION TECHNIQUES

In this section, we provide techniques for verifying observational properties introduced in the previous section. First, we show that most of the properties in partially observed DES can be verified using the observer structure, and some important properties can be more efficiently verified using the twin-plant technique in polynomial time.

5.1. Observer-Based Verification

Verification of Detectability. First, we study the verification of current-state detectability. Recall that,

current-state detectability requires that $(\exists n \in \mathbb{N})(\forall \alpha \in P(\mathcal{L}(G)) : |\alpha| \geq n)[|\hat{X}_G(\alpha)| = 1]$. In other words, a system is *not* current-state detectable if there is an arbitrarily long observation string such that the current-state estimate is *not always* a singleton starting from any instant. Since $P(\mathcal{L}(G))$ is a regular language, due to the Pumping lemma, the existence of such an arbitrarily long string is equivalent to the existence of a cycle in Obs(G) in which a state is not a singleton. This immediately suggests Algorithm 5 for the verification of current-state detectability. Initial-state detectability can also be checked in the same manner using G_{aug}. The only differences from Algorithm 5 are (i) we need to consider Obs(G_{aug}) rather than Obs(G) in line 1 and (ii) in line 2, we need to check the existence of a cycle $q_0 q_1 \ldots q_n$ in Obs(G_{aug}) such that $|I_0(q_i)| > 1$ for some $i = 1, \ldots, n$.

Algorithm 5 Cur-State-Dect-Ver-Obs

Inputs: G
Output: Current-State Detectable or Not
1: Build Obs(G) = $(X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0})$
2: **if** there exists a cycle $q_0 q_1 \ldots q_n$ in Obs(G) such that $|q_i| > 1$ for some $i = 1, \ldots, n$ **then**
3: **return** G is not current-state detectable
4: **else**
5: **return** G is current-state detectable
6: **end if**

To check delayed detectability (for parameters $k_1, k_2 \in \mathbb{N}$), first we recall that the delayed-state estimate can be computed by $\hat{X}_G(\alpha \mid \alpha\beta) = \hat{X}_G(\alpha) \cap \hat{X}_{G_{\text{R}}}(\beta_R)$. Therefore, delayed detectability can be checked by the following steps:

- First, we compute all possible current-state estimate $\hat{X}_G(\alpha)$ reachable via some string α whose length is greater than or equal to k_1;
- Then, we compute all possible current-state estimate of $\hat{X}_{G_{\text{R}}}(\beta_R)$ in G_{R} reachable via some string β_R whose length is greater than or equal to k_2;
- Finally, we test whether or not there exist such $\hat{X}_G(\alpha)$ and $\hat{X}_{G_{\text{R}}}(\beta_R)$ such that $|\hat{X}_G(\alpha) \cap \hat{X}_{G_{\text{R}}}(\beta_R)| > 1$. If so, then it means that some instant after k_1 steps cannot be determined within k_2 steps of delay, that is, the system is not delayed detectable.

This procedure is formalized by Algorithm 6, where states in lines 3 and 4 can be computed by a simple depth-first search or a breath-first search.

Verification of Diagnosability. The verification of diagnosability is very similar to the case of current-state detectability. Specifically, a system is *not* diagnosable if there is an arbitrarily long string *after the occurrence of fault* such that the current-state estimate is *not* a subset of X_N starting from any instant. Therefore, the general idea is to replaced condition $|q_i| > 1$ in Algorithm 5 by $q_i \not\subseteq X_F$. However, we need to do a little bit more here since we

Algorithm 6 Delay-State-Dect-Ver-Obs

Inputs: G
Output: Delayed Detectable or Not
1: Build Obs(G) = $(X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0})$
2: Build Obs(G_{R}) = $(X^R_{\text{obs}}, \Sigma_o, \delta^R_{\text{obs}}, X)$
3: **for all** $q_1 \in X_{\text{obs}}$ that be can reached by a string longer than k_1 in Obs(G) **do**
4: **for all** $q_2 \in X^R_{\text{obs}}$ that be can reached by a string longer than k_2 in Obs(G_{R}) **do**
5: **if** $|q_1 \cap q_2| > 1$ **then**
6: **return** G is not delayed detectable
7: **end if**
8: **end for**
9: **end for**
10: **return** G is delayed detectable

are only interested in arbitrarily long uncertain strings *after the occurrence of fault*; this is also referred to as an *indeterminate cycles* in the literature (17). In other words, the existence of an arbitrarily long uncertain but nonfault string does not necessarily violate diagnosability.

To this end, we need to compose Obs(G) with the dynamic of the original system G. Specifically, let Obs(G) be the observer of G. We define

$$\widetilde{\text{Obs}}(G) = (X_{\text{obs}}, \Sigma, \tilde{\delta}_{\text{obs}}, x_{\text{obs},0}) \tag{14}$$

as the DFA by adding self-loops of all unobservable events at each state in X_{obs}. One can easily check that $\mathcal{L}(G) = \mathcal{L}(G \times \widetilde{\text{Obs}}(G))$. Then, by looking at the first component of each state in $G \times \widetilde{\text{Obs}}(G)$, we know whether a fault event has occurred or not. Therefore, we can check diagnosability based on $G \times \widetilde{\text{Obs}}(G)$ by Algorithm 7.

Algorithm 7 Diag-Ver-Obs

Inputs: G and Σ_F
Output: Diagnosable or Not
1: Build Obs(G) = $(X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0})$
2: Augment Obs(G) as $\widetilde{\text{Obs}}(G) = (X_{\text{obs}}, \Sigma, \tilde{\delta}_{\text{obs}}, x_{\text{obs},0})$ by adding unobservable self-loops
3: Compute $G \times \widetilde{\text{Obs}}(G)$
4: **if** there exists a cycle $(x_0, q_0)(x_1, q_1) \ldots (x_n, q_n)$ in $G \times \widetilde{\text{Obs}}(G)$ such that $x_i \in X_F$ and $q_i \notin X_F$ for some $i = 1, \ldots, n$ **then**
5: **return** G is not diagnosable
6: **else**
7: **return** G is diagnosable
8: **end if**

Verification of Distinguishability and Opacity. The verification of distinguishability is straightforward using the observer. Specifically, we just need to check whether or not there exists a state $q \in X_{\text{obs}}$ in Obs(G) such that $(q \times q) \cap$

$T_{\text{spec}} \neq \emptyset$. If so, the system is not distinguishable; otherwise, it is distinguishable. Since we have discussed that prognosability is a special case of distinguishability, it can also be checked in the same manner.

Similarly, current-state opacity can be verified by checking whether or not there exists a state $q \in X_{\text{obs}}$ in Obs(G) such that $q \subseteq X_S$. If so, the system is not opaque; otherwise, it is opaque. To check initial-state opacity, we need to construct Obs(G_{R}), and to check whether or not there exists a state $q \in X^R_{\text{obs}}$ in Obs(G_{R}) such that $q \cap X_0 \subseteq X_S$.

The verification of infinite-step opacity is similar to the case of delayed detectability; both involve delayed-state estimate that can be computed according to Proposition 3. The verification procedure for infinite-step opacity is provided in Algorithm 8 with complexity $O(|\Sigma|4^{|X|})$.

Algorithm 8 Inf-Opa-Ver-Obs

Inputs: G
Output: Infinite-Step Opaque or Not
1: Build Obs(G) = $(X_{\text{obs}}, \Sigma_o, \delta_{\text{obs}}, x_{\text{obs},0})$
2: Build Obs(G_{R}) = $(X^R_{\text{obs}}, \Sigma_o, \delta^R_{\text{obs}}, X)$
3: **for all** $q_1 \in X_{\text{obs}}$ **do**
4: **for all** $q_2 \in X^R_{\text{obs}}$ **do**
5: **if** $\emptyset \neq q_1 \cap q_2 \subseteq X_S$ **then**
6: **return** G is not infinite-step opaque
7: **end if**
8: **end for**
9: **end for**
10: **return** G is infinite-step opaque

5.2. Twin-Plant-Based Verification

In the previous section, we have shown that the observer can be used for the verification of all properties introduced. However, the size of the observer is exponential in the size of the system. Then, the natural question arises as to can we find polynomial-time algorithms for verifying these properties? Unfortunately, it has been shown in Reference 46 that deciding opacity is PSPACE complete; hence, no polynomial-time algorithm exists. However, it is indeed possible to check diagnosability, detectability, and distinguishability in polynomial time using the *twin-plant* structure. This structure was originally proposed in Reference 47 for the verification of observability; later on, it has been used for verifying diagnosability (under the name of "verifier") by References 48 and 49 and for verifying detectability (under the name of "detector") by Reference 50.

Definition 10 (Twin-Plant) Given system $G = (X, \Sigma, \delta, X_0)$ with observable events $\Sigma_o \subseteq \Sigma$, the twin plant is a new NFA $V(G) = (X_V, \Sigma_V, \delta_V, X_{0,V})$, where

- $X_V \subseteq X \times X$ is the set of states;
- $\Sigma_V = (\Sigma_o \times \Sigma_o) \cup (\Sigma_{\text{uo}} \times \{\epsilon\}) \cup (\{\epsilon\} \times \Sigma_{\text{uo}})$ is the set of events;
- $X_{0,V} = X_0 \times X_0$ is the set of initial states; and
- $\delta_V : X_V \times \Sigma_V \to 2^{X_V}$ is the partial transition function defined by: for any state $(x_1, x_2) \in X_V$ and event $\sigma \in \Sigma$

(a) If $\sigma \in \Sigma_o$, then the following transition is defined:

$$\delta_V((x_1, x_2), (\sigma, \sigma)) = \delta(x_1, \sigma) \times \delta(x_2, \sigma) \tag{15}$$

(a) If $\sigma \in \Sigma_{uo}$, then the following transitions are defined:

$$\delta_V((x_1, x_2), (\sigma, \epsilon)) = \delta(x_1, \sigma) \times \{x_2\} \tag{16}$$

$$\delta_V((x_1, x_2), (\epsilon, \sigma)) = \{x_1\} \times \delta(x_2, \sigma) \tag{17}$$

Hereafter, we only consider the accessible part of $V(G)$.

Intuitively, the twin plant V tracks all pairs of observation equivalent strings in G. Specifically, if $s_1, s_2 \in \mathcal{L}(G)$ are two strings in G such that $P(s_1) = P(s_2)$, then there exists a string $s \in \mathcal{L}(V(G))$ in $V(G)$ such that its first and second components are s_1 and s_2, respectively. On the other hand, for any string $s = (s_1, s_2) \in \mathcal{L}(V(G))$ in $V(G)$, we have that $P(s_1) = P(s_2)$.

The twin plant can be applied directly for the verification of distinguishability. Specifically, we need to check if $V(G)$ contains a pair of states in the specification. This procedure is presented in Algorithm 9.

Algorithm 9 Dist-Ver-TP

Inputs: G and T_{spec}
Output: Distinguishable or Not
1: Build $V(G) = (X_V, \Sigma_V, \delta_V, X_{0,V})$
2: **if** there exists a state $(x_1, x_2) \in X_V$ in $V(G)$ such that $(x_1, x_2) \in T_{\text{spec}}$ **then**
3: **return** G is not distinguishable
4: **else**
5: **return** G is distinguishable
6: **end if**

According to the definition of detectability, we know that the system is not detectable if and only if there exist two arbitrarily long strings having the same observation, such that these two strings lead to two different states. As we discussed above, all such string pairs can be captured by the twin plant. This suggests Algorithm 10 for the verification of current-state detectability.

The case of diagnosability is similar. Specifically, a system is not diagnosable if there exist an arbitrarily long fault string and a nonfault string such that they have the same observation. This condition can be checked by Algorithm 11. Note that, we have already assumed that there is no unobservable cycle in G. Otherwise, we need to add the following condition to the "if condition" in line 2 of Algorithm 11 to obtain an arbitrarily long fault string:

$$\exists j = 0, \ldots, n-1, \exists \sigma_V \notin \Sigma_{uo} \times \{\epsilon\} : (x^1_{j+1}, x^2_{j+1}) \in \delta_V((x^1_j, x^2_j), \sigma_V)$$

Algorithm 10 Cur-State-Dect-Ver-TP

Inputs: G and Σ_F
Output: Current-State Detectable or Not
1: Build $V(G) = (X_V, \Sigma_V, \delta_V, X_{0,V})$
2: **if** there exists a cycle $(x_0^1, x_0^2)(x_1^1, x_1^2) \dots (x_n^1, q_n^2)$ in $V(G)$ such that $\exists i = 0, \dots, n : x_i^1 \neq x_i^2$ **then**
3: **return** G is not current-state detectable
4: **else**
5: **return** G is current-state detectable
6: **end if**

Algorithm 11 Diag-Ver-TP

Inputs: G and Σ_F
Output: Diagnosable or Not
1: Build $V(G) = (X_V, \Sigma_V, \delta_V, X_{0,V})$
2: **if** there exists a cycle $(x_0^1, x_0^2)(x_1^1, x_1^2) \dots (x_n^1, q_n^2)$ in $V(G)$ such that
3: $\forall i = 0, \dots, n : x_i^1 \in X_N \wedge x_i^2 \in X_F$ **then**
4: **return** G is not diagnosable
5: **else**
6: **return** G is diagnosable
7: **end if**

Since the size of the twin plant is only quadratic in the size of G, distinguishability, detectability, and diagnosability can all be checked in polynomial time, which is better than the observer-based approach.

6. RELATED PROBLEMS AND FURTHER READINGS

6.1. State Estimation under General Observation Models

Throughout this article, we assume that the observation of the system is modeled as a natural projection. The natural projection mapping is essentially static in the sense that an event is always either observable or unobservable. One related topic is the *sensor selection problem* for static observations (51–55), that is, we want to decide which events should be observable by placing with sensors such that a given observational property is fulfilled. Another related topic in the static observation setting is the *robust state estimation* problem when the observation is unreliable. This problem has been investigated in the context of detectability analysis (56), diagnosability analysis (57–60), and supervisory control (61–66) for partially observed DES.

In many situations, due to information communications and acquisitions, the observation mapping may be *dynamic*, that is, whether or not an event is observable depends on the trajectory of the system. One example is the dynamic sensor activation problem, where we can decide to turn sensors on/off dynamically online based on the observation history. In References 67–70, the fault diagnosis problem is studied under the dynamic observation setting. In References 71 and 72, observability is studied under the dynamic observation setting. In References 50 and 73, the authors investigated the verification and synthesis of detectability under dynamic observations. Opacity under dynamic observations is also studied in References 46, 74, 75. A general approach for dynamic sensor activation for property enforcement is proposed by Reference 76. In References 77 and 78, it has been shown that (co)diagnosability and (co)observability can be mapped from one to the other in the general dynamic observation setting. The reader is referred to the recent survey (79) for more references on state-estimation problem under dynamic observations.

6.2. State Estimation in Coordinated, Distributed, and Modular Systems

In the setting of this article, we only consider the scenario where the system is monitored by a single observer, which is referred to as the *centralized state estimation*. In many large-scale systems, sensors can be physically distributed, and the system can be monitored by multiple local observers that have incomparable information. Each local observer can perform local state estimation and send it to a coordinator. Then, the coordinator will fuse all local information according to some prespecified protocol in order to obtain a global estimation decision. This is referred to as the *decentralized estimation and decision-making problem under coordinated architecture*.

In the context of DES, the decentralized estimation and decision-making problem was first studied by References 32, 80, 81 in the context of decentralized supervisory control, where the notion of coobservability is proposed. In References 82, 83, the problem of decentralized fault diagnosis problem was studied, and a corresponding property called codiagnosability was proposed; the verification codiagnosability has also been studied in the literature (84, 85). The decentralized fault prognosis has also been studied in the literature; see, for example, References 27, 86–88. Note that, in decentralized decision-making problems, one important issue is the underlying coordinated architecture. For example, References 80 and 85 only consider simple binary architectures for control and diagnosis, respectively; more complicated architectures can be found in References 27, 89–96.

When each local observer is allowed to communicate and exchange information with each other, the state-estimation problem is referred to as the *distributed*-estimation problem. Works on distributed state estimation and property verification can be found in References 97–100. Finally, state estimation and verification of partially observed *modular* DES have also been considered in the literature (12, 15, 101–104), where a modular system is composed of a set of local modules in the form of $G = G_1 \times \cdots \times G_n$.

6.3. Estimation and Verification of Petri Nets and Stochastic DES

In this article, we focus on DES modeled as finite-state automata. Petri nets, another important class of DES

models, are widely used to model many classes of concurrent systems. In particular, Petri nets provide a compact model without enumerating the entire state space, and it is well known that Petri net languages are more expressive than regular languages. The problems of state estimation and property verification have also drawn many attentions in the context of Petri nets. For example, state (marking) estimation algorithms for Petri nets have been proposed in References 105–108. Decidability and verification procedures for diagnosability (109–113), detectability (114, 115), prognosability (116), and opacity (117, 118) are also studied in the literature for Petri nets.

Another important generalization of the finite-state automata model is the stochastic DES (or labeled Markov chains). Stochastic DES can not only characterize whether or not a system can reach a state, it can also capture the possibility of reaching a state. Hence, it provides a model for the quantitative analysis and verification of DES. State estimation and verification of many important properties have also been extended to the stochastic DES setting; this includes, for example, diagnosability (119–121), detectability (11, 14, 56, 122), prognosability (123), and opacity (124–126).

6.4. Control Synthesis of Partially Observed DES

So far, we have only discussed property verification problems in partially observed DES. In many applications, when the answer to the verification problem is negative, it is important to *synthesize* a supervisor or controller that provably enforces the property by restricting the system behavior but as permissive as possible. This control synthesis problem has been studied in the literature in the framework of the *supervisory control theory* initiated by Ramadge and Wonham (127). The reader is referred to the textbooks (1–128) for more details on supervisory control of DES. Supervisory control under partial observation was originally investigated by References 31 and 32. Since then, many control synthesis algorithms for partial-observation supervisors have been proposed (129–133). In particular, Reference 134 solves the synthesis problem for maximally permissive nonblocking supervisors in the partial-observation setting.

In the context of enforcement of observational properties, in Reference 135, an approach was proposed for designing a supervisor that enforces diagnosability. Control synthesis algorithms have also been proposed in the literature for enforcing opacity; see, for example, References 136–138. In Referene 139, the authors studied the problem of synthesizing supervisors that enforce detectability. A uniform approach for control synthesis for enforcing a wide class of properties was recently proposed by Yin and Lafortune in a series of papers (134, 140–142).

RELATED ARTICLES

Discrete Event Systems; Discrete Event Dynamical Systems

BIBLIOGRAPHY

1. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*, 2nd ed.; Springer, 2008.
2. Y.-C. Wu and S. Lafortune. *Discrete Event Dyn. S.* **2013**, *23* (3), pp 307–339.
3. X. Yin and S. Lafortune. *Automatica*. **2017**, *80*, pp 162–171.
4. S. Shu, F. Lin, and H. Ying. *IEEE T. Automat. Contr.* **2007**, *52* (12), pp 2356–2359.
5. S. Shu and F. Lin. *IEEE T Autom. Sci. Eng.* **2013**, *10* (1), pp 187–196.
6. S. Shu and F. Lin. *IEEE T. Automat. Contr.* **2013**, *58* (4), pp 862–875.
7. C. M. Ozveren and A. S. Willsky. *IEEE T. Automat. Contr.* **1990**, *35* (7), pp 797–806.
8. C. N. Hadjicostis and C. Seatzu. K-Detectability in Discrete Event Systems, in 55th IEEE Conference on Decision and Control, pp 420–425, 2016.
9. S. Shu and F. Lin. *Syst. Control Lett.* **2011**, *60* (5), pp 310–317.
10. Y. Sasi and F. Lin. *Discrete Event Dyn. S.* **2018**, *28* (3), pp 449–470.
11. C. Keroglou and C. N. Hadjicostis. *Automatica*. **2017**, *86*, pp 192–198.
12. T. Masopust and X. Yin. *Automatica*. **2019**, *101*, pp 290–295.
13. T. Masopust. *Automatica*. **2018**, *93*, pp 257–261.
14. S. Shu, F. Lin, H. Ying, and X. Chen. *Automatica*. **2008**, *44* (12), pp 3054–3060.
15. X. Yin and S. Lafortune. *Automatica*. **2017**, *83*, pp 199–205.
16. K. Zhang. *Automatica*. **2017**, *81*, pp 217–220.
17. M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. *IEEE T. Automat. Contr.* **1995**, *40* (9), pp 1555–1575.
18. T.-S. Yoo and H. E. Garcia. *Syst. Control Lett.* **2008**, *57* (12), pp 1023–1029.
19. F. Lin. *Discrete Event Dyn. S.* **1994**, *4* (2), pp 197–212.
20. S. H. Zad, R. H. Kwong, and W. M. Wonham. *IEEE T. Automat. Contr.* **2003**, *48* (7), pp 1199–1212.
21. S. Jiang, R. Kumar, and H. E. Garcia. *IEEE T. Robotic. Autom.* **2003**, *19* (2), pp 310–323.
22. O. Contant, S. Lafortune, and D. Teneketzis. *Discrete Event Dyn. S.* **2004**, *14* (2), pp 171–202.
23. E. Fabre, L. Hélouët, E. Lefaucheux, and H. Marchand. *Discrete Event Dyn. S.* **2018**, *28* (2), pp 183–213.
24. J. Zaytoon and S. Lafortune. *Annu. Rev. Control*. **2013**, *37* (2), pp 308–320.
25. T. Jéron, H. Marchand, S. Genc, and S. Lafortune. *IFAC P. Vol.* **2008**, *41* (2), pp 537–543.
26. S. Genc and S. Lafortune. *Automatica*. **2009**, *45* (2), pp 301–311.
27. X. Yin and Z. Li. *Automatica*. **2016**, *69*, pp 375–379.
28. W. Wang, S. Lafortune, and F. Lin. *Syst. Control Lett.* **2007**, *56* (9–10), pp 656–661.
29. D. Sears and K. Rudie. On Computing Indistinguishable States of Nondeterministic Finite Automata with Partially Observable Transitions, in 53rd IEEE Conference on Decision and Control, pp 6731–6736, 2014.
30. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2018**, *63* (11), pp 3705–3718.
31. F. Lin and W. M. Wonham. *Inform. Sci*. **1988**, *44* (3), pp 173–198.

32. R. Cieslak, C. Desclaux, A. S. Fawaz, and P. Varaiya. *IEEE T. Automat. Contr.* **1988**, *33* (3), pp 249–260.
33. A. Saboori and C. N. Hadjicostis. *IEEE T. Automat. Contr.* **2012**, *57* (5), pp 1265–1269.
34. A. Saboori and C. N. Hadjicostis. *Inform. Sciences*. **2013**, *246*, pp 115–132.
35. K. Zhang, X. Yin, and M. Zamani. *IEEE T. Automat. Contr.* **2019**. DOI: 10.1109/TAC.2019.2908726.
36. L. Mazaré. Using Unication for Opacity Properties, in *Proceedings of WITS*, *4*, pp 165–176, 2004.
37. E. Badouel, M. Bednarczyk, A. Borzyszkowski, B. Caillaud, and P. Darondeau. *Discrete Event Dyn. S.* **2007**, *17* (4), pp 425–446.
38. J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan. *Int. J. Inf. Secur.* **2008**, *7* (6), pp 421–435.
39. A. Saboori and C. N. Hadjicostis. Notions of Security and Opacity in Discrete Event Systems, in *46th IEEE Conference on Decision and Control*, IEEE, pp 5056–5061, 2007.
40. F. Lin. *Automatica*. **2011**, *47* (3), pp 496–503.
41. A. Saboori and C. N. Hadjicostis. *IEEE T Autom. Sci. Eng.* **2011**, *8* (3), pp 549–559.
42. L. Sweeney. *Int. J. Uncertain. Fuzz.* **2002**, *10* (05), pp 557–570.
43. N. Ben Hadj-Alouane, S. Lafrance, F. Lin, J. Mullins, and M. Yeddes. *IEEE T. Syst. Man Cy.B*. **2005**, *35* (5), pp 948–958.
44. N. Ben Hadj-Alouane, S. Lafrance, F. Lin, J. Mullins, and M. Yeddes. *IEEE T. Automat. Contr.* **2005**, *50* (6), pp 920–925.
45. R. Jacob, J.-J. Lesage, and J.-M. Faure. *Annu. Rev. Control*. **2016**, *41*, pp 135–146.
46. F. Cassez, J. Dubreil, and H. Marchand. *Form. Method. Syst. Des.* **2012**, *40* (1), pp 88–115.
47. J. N. Tsitsiklis, *Math. Control Signal.* **1989**, *2* (2), pp 95–107.
48. S. Jiang, Z. Huang, V. Chandra, and R. Kumar. *IEEE T. Automat. Contr.* **2001**, *46* (8), pp 1318–1321.
49. T.-S. Yoo and S. Lafortune. *IEEE T. Automat. Contr.* **2002**, *47* (9), pp 1491–1495.
50. S. Shu and F. Lin. *Syst. Control Lett.* **2010**, *59* (1), pp 9–17.
51. A. Haji-Valizadeh and K. Loparo. *IEEE T. Automat. Contr.* **1996**, *41* (11), pp 1579–1593.
52. T.-S. Yoo and S. Lafortune. *IEEE T. Automat. Contr.* **2002**, *47* (9), pp 1495–1499.
53. S. Jiang, R. Kumar, and H. E. Garcia. *IEEE T. Automat. Contr.* **2003**, *48*(3), pp 369–381.
54. Y. Ru and C. N. Hadjicostis. *IEEE T. Automat. Contr.* **2010**, *55* (8), pp 1751–1764.
55. M. P. Cabasino, S. Lafortune, and C. Seatzu. *Automatica*. **2013**, *49* (8), pp 2373–2383.
56. X. Yin. *Automatica*. **2017**, *80*, pp 127–134.
57. D. Thorsley, T.-S. Yoo, and H. E. Garcia. Diagnosability of Stochastic Discrete-Event Systems under Unreliable Observations, in *American Control Conference*, pp 1158–1165, 2008.
58. S. Takai and T. Ushio. *IEEE T. Automat. Contr.* **2012**, *57* (3), pp 798–804.
59. L. K. Carvalho, J. C. Basilio, and M. V. Moreira. *Automatica*. **2012**, *48* (9), pp 2068–2078.
60. L. K. Carvalho, M. V. Moreira, J. C. Basilio, and S. Lafortune. *Automatica*. **2013**, *49* (1), pp 223–231.
61. K. Rohloff. Sensor Failure Tolerant Supervisory Control, in *44th IEEE Conf. Decision and Control*, pp 3493–3498, 2005.
62. A. M. Sánchez and F. J. Montoya. *Discrete Event Dyn. S.* **2006**, *16* (4), pp 493–525.
63. F. Lin. *SIAM J. Control Optim.* **2014**, *52* (2), pp 1276–1298.
64. M. V. S. Alves, J. C. Basilio, A. E. Carrilho da Cunha, L. K. Carvalho, and M. V. Moreira. Robust Supervisory Control Against Intermittent Loss of Observations, in *12th International Workshop on Discrete Event Systems*, *12*, pp 294–299, 2014.
65. T. Ushio and S. Takai. *IEEE T. Automat. Contr.* **2016**, *61* (3), pp 799–804.
66. X. Yin. *IEEE T. Automat. Contr.* **2017**, *62* (5), pp 2576–2581.
67. D. Thorsley and D. Teneketzis. *Discrete Event Dyn. S.* **2007**, *17* (4), pp 531–583.
68. F. Cassez and S. Tripakis. *Fundamental Informaticae*. **2008**, *88* (4), 497–540.
69. W. Wang, S. Lafortune, A. R. Girard, and F. Lin. *Automatica*. **2010**, *46* (7), pp 1165–1175.
70. E. Dallal and S. Lafortune. *IEEE T. Automat. Contr.* **2014**, *59* (4) pp 966–981.
71. Y. Huang, K. Rudie, and F. Lin. *IEEE T. Automat. Contr.* **2008**, *53* (1), pp 384–388.
72. W. Wang, S. Lafortune, F. Lin, and A. R. Girard. *IEEE T. Automat. Contr.* **2010**, *55* (11), pp 2447–2461.
73. S. Shu, Z. Huang, and F. Lin. *IEEE Trans. Autom. Sci. Eng.* **2013**, *10* (2), pp 457–461.
74. Y. Ji, X. Yin, and S. Lafortune. *IEEE T. Automat. Contr.* **2019**, *64* (10), pp 4369–4376.
75. B. Zhang, S. Shu, and F. Lin. *IEEE T Autom. Sci. Eng.* **2015**, *12* (4), pp 1067–1079.
76. X. Yin and S. Lafortune. *Automatica*. **2019**, *105*, pp 376–383.
77. W. Wang, A. R. Girard, S. Lafortune, and F. Lin. *IEEE T. Automat. Contr.* **2011**, *56* (7), pp 1551–1566.
78. X. Yin and S. Lafortune. *Automatica*. **2015**, *61*, pp 241–252.
79. D. Sears and K. Rudie. *Discrete Event Dyn. S.* **2016**, *26* (2), pp 295–349.
80. K. Rudie and W. M. Wonham. *IEEE T. Automat. Contr.* **1992**, *37* (11), pp 1692–1708.
81. K. Rudie and J. C. Willems. *IEEE T. Automat. Contr.* **1995**, *40* (7), pp 1313–1319.
82. R. Debouk, S. Lafortune, and D. Teneketzis. *Discrete Event Dyn. S.* **2000**, *10* (1–2), pp 33–86.
83. Y. Wang, T.-S. Yoo, and S. Lafortune. *Discrete Event Dyn. S.* **2007**, *17* (2), pp 233–263.
84. M. V. Moreira, T. C. Jesus, and J. C. Basilio. *IEEE T. Automat. Contr.* **2011**, *56* (7), pp 1679–1684.
85. W. Qiu and R. Kumar. *IEEE T. Syst. Man Cy. A*. **2006**, *36* (2), pp 384–395.
86. R. Kumar and S. Takai. *IEEE T. Automat. Contr.* **2010**, *55* (1), pp 48–59.
87. A. Khoumsi and H. Chakib. *IEEE T Autom. Sci. Eng.* **2012**, *9* (2), pp 412–417.
88. X. Yin and Z. Li. *IEEE T. Cy.* **2019**, *49* (4), pp 1302–1313.
89. T.-S. Yoo and S. Lafortune. *Discrete Event Dyn. S.* **2002**, *12* (3), pp 335–377.
90. T.-S. Yoo and S. Lafortune. *IEEE T. Automat. Contr.* **2004**, *49* (11), pp 1886–1904.
91. R. Kumar and S. Takai. *IEEE T Autom. Sci. Eng.* **2009**, *6* (3), pp 479–491.
92. R. Kumar and S. Takai. *IEEE T. Automat. Contr.* **2007**, *52* (10), pp 1783–1794.

93. S. Takai and R. Kumar. *IEEE T. Automat. Contr.* **2011**, *56* (1), pp 165–171.
94. H. Chakib and A. Khoumsi. *IEEE T. Automat. Contr.* **2011**, *56* (11), pp 2608–2622.
95. S. L. Ricker and K. Rudie. *IEEE T. Automat. Contr.* **2007**, *52* (3), pp 428–441.
96. A. Khoumsi and H. Chakib. *IEEE T. Automat. Contr.* **2018**, *63* (12), pp 4278–4285.
97. A. Benveniste, E. Fabre, S. Haar, and C. Jard. *IEEE T. Automat. Contr.* **2003**, *48* (5), pp 714–727.
98. R. Su and W. M. Wonham. *IEEE T. Automat. Contr.* **2005**, *50* (12), pp 1923–1935.
99. S. Genc and S. Lafortune. *IEEE T Autom. Sci. Eng.* **2007**, *4* (2), pp 206–219.
100. S. Takai and R. Kumar. *IEEE T. Automat. Contr.* **2012**, *57* (5), pp 1259–1265.
101. K. Rohloff and S. Lafortune. *Discrete Event Dyn. S.* **2005**, *15* (2), pp 145–167.
102. O. Contant, S. Lafortune, and D. Teneketzis. *Discrete Event Dyn. S.* **2006**, *16* (1), pp 9–37.
103. A. Saboori and C. N. Hadjicostis. *IFAC P. Vol.* **2010**, *43* (12), pp 78–83.
104. K. W. Schmidt. *IEEE T. Syst. Man Cy.: Syst.* **2013**, *43* (5), pp 1130–1140.
105. A. Giua, C. Seatzu, and D. Corona. *IEEE T. Automat. Contr.* **2007**, *52* (9), pp 1695–1699.
106. L. Li and C. N. Hadjicostis. *IEEE T. Automat. Contr.* **2013**, *58* (1), pp 198–203.
107. P. Bonhomme. *IEEE T. Syst. Man Cy.: Syst.* **2015**, *45* (3), pp 508–518.
108. F. Basile, M. P. Cabasino, and C. Seatzu. *IEEE T. Automat. Contr.* **2015**, *60* (4), pp 997–1009.
109. B. Bérard, S. Haar, S. Schmitz, and S. Schwoon. *Fund. Inform.* **2018**, *161* (4), pp 317–349.
110. N. Ran, H. Su, A. Giua, and C. Seatzu. *IEEE T. Automat. Contr.* **2018**, *63* (4), pp 1192–1199.
111. F. Basile, P. Chiacchio, and G. De Tommasi. *Automatica*. **2012**, *48* (9), pp 2047–2058.
112. M. P. Cabasino, A. Giua, S. Lafortune, and C. Seatzu. *IEEE T. Automat. Contr.* **2012**, *57* (12), pp 3104–3117.
113. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2017**, *62* (11), pp 5931–5938.
114. T. Masopust and X. Yin. *Automatica*. **2019**, *104*, pp 238–241.
115. A. Giua and C. Seatzu. *IEEE T. Automat. Contr.* **2002**, *47* (9), pp 1424–1437.
116. X. Yin. *IEEE T. Automat. Contr.* **2018**, *63* (6), pp 1828–1834.
117. Y. Tong, Z. Li, C. Seatzu, and A. Giua. *IEEE T. Automat. Contr.* **2017**, *62* (6), pp 2823–2837.
118. Y. Tong, Z. Li, C. Seatzu, and A. Giua. *Automatica*. **2017**, *80*, pp 48–53.
119. X. Yin, J. Chen, Z. Li, and S. Li. *IEEE T. Automat. Contr.* **2019**, *64* (10), pp 4237–4244.
120. F. Liu, D. Qiu, H. Xing, and Z. Fan. *IEEE T. Automat. Contr.* **2008**, *53* (2), pp 535–546.
121. D. Thorsley and D. Teneketzis. *IEEE T. Automat. Contr.* **2005**, *50* (4), pp 476–492.
122. P. Zhao, S. Shu, F. Lin, and B. Zhang. *IEEE T. Automat. Contr.* **2019**, *64* (1), pp 433–439.
123. J. Chen and R. Kumar. *IEEE T. Automat. Contr.* **2015**, *60* (6), pp 1570–1581.
124. A. Saboori and C. N. Hadjicostis. *IEEE T. Automat. Contr.* **2014**, *59* (1), pp 120–133.
125. B. Bérard, K. Chatterjee, and N. Sznajder. *Inform. Process. Lett.* **2015**, *115* (1), pp 52–59.
126. X. Yin, Z. Li, W. Wang, and S. Li. *Automatica*. **2019**, *99*, pp 266–274.
127. P. J. Ramadge and W. M. Wonham. *SIAM J. Control Optim.* **1987**, *25* (1), pp 206–230.
128. W. M. Wonham and K. Cai. *Supervisory Control of Discrete-Event Systems*. Springer, 2019.
129. H. Cho and S. I. Marcus. *Math. Syst. Theory*. **1989**, *22* (1), pp 177–211.
130. R. Kumar, V. Garg, and S. I. Marcus. *Syst. Control Lett.* **1991**, *17* (3), pp 157–168.
131. S. Takai and T. Ushio. *Syst. Control Lett.* **2003**, *49* (3), pp 191–200.
132. T.-S. Yoo and S. Lafortune. *Discrete Event Dyn. S.* **2006**, *16* (4), pp 527–553.
133. K. Cai, R. Zhang, and W. M. Wonham. *IEEE T. Automat. Contr.* **2015**, *60* (3), pp 659–670.
134. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2016**, *61* (5), pp 1239–1254.
135. M. Sampath, S. Lafortune, and D. Teneketzis. *IEEE T. Automat. Contr.* **1998**, *43* (7), pp 908–929.
136. J. Dubreil, P. Darondeau, and H. Marchand. *IEEE T. Automat. Contr.* **2010**, *55* (5), pp 1089–1100.
137. Y. Tong, Z. Li, C. Seatzu, and A. Giua. *Discrete Event Dyn. S.* **2018**, *28* (2), pp 161–182.
138. A. Saboori and C. N. Hadjicostis. *IEEE T. Automat. Contr.* **2012**, *57* (5), pp 1155–1165.
139. S. Shu and F. Lin. *IEEE T. Automat. Contr.* **2013**, *58* (8), pp 2125–2130.
140. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2018**, *63* (12), pp 4435–4441.
141. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2017**, *62* (8), pp 3914–3929.
142. X. Yin and S. Lafortune. *IEEE T. Automat. Contr.* **2016**, *61* (8), pp 2140–2154.

XIANG YIN

Shanghai Jiao Tong University, Shanghai, China