# An Incremental Approach for Pattern Diagnosability in Distributed Discrete Event Systems

Lina YE, Philippe DAGUE
LRI, Univ. Paris-Sud 11
CNRS / INRIA Saclay, Ile-de-France
4 rue Jacques Monod, bat G
Orsay, F-91893, France
lina.ye@lri.fr, philippe.dague@lri.fr

Yuhong YAN
Concordia University
1455 de Maisonneuve Blvd.
West, EV3.227 Montreal, Quebec
H3G 1M8, Canada
yuhong@encs.concordia.ca

## Abstract

*Diagnosability is a crucial property that determines at design stage how accurate any diagnosis algorithm can be on a partially observable system. Recent work on diagnosability has generalized fault event case to pattern case, which can describe more general objectives for diagnosis problem, but based on global model and global twin plant construction. In this paper, we propose an original framework to solve pattern diagnosability in a distributed way to avoid calculating global objects. We first show how to incrementally accomplish pattern recognition without building global model by propagating only diagnosability relative information between components. Then an efficient way to construct pattern verifier is proposed, which is inspired from the classical twin plant method but with smaller state space, to search for partial critical paths, whose global consistency is subsequently checked. Meanwhile we prove that the result obtained from our distributed approach is on an equality with that from the centralized one but the evaluation result shows that our search state space exploited is only a small subpart of the global twin plant, whose construction is unavoidable in the centralized approach.*

## 1. Introduction

Fault diagnosis is a crucial and challenging task in the automatic control of large complex systems. Generally speaking, diagnosis reasoning is to detect possible faults that can explain the observations, whose correctness depends on the diagnosability of the system. Diagnosability is an important property that determines at design stage how accurate any diagnosis algorithm can be on a partially observable system and thus has significant economic impact on the improvement of performance and reliability of complex systems. The diagnosability analysis problem has received considerable attention in the literature. [6] introduced the first definition of diagnosability for discrete event systems and proposed a necessary and sufficient condition for testing it. The major drawback is that the complexity is exponential in the number of system states. Then [4] and [9] proposed new algorithms with polynomial complexity in the number of system states, which introduce the classical twin plant method.

However, all above work analyzes diagnosability in a centralized way. Real systems, e.g. telecommunication networks, water distribution networks, transportation systems, are steadily growing in terms of size, complexity and interactions. The centralized diagnosability approach requires a unrealistic combinatorial explosion of the search space. It is why very recently the distributed approach for diagnosability began to be investigated, relying on local models and local twin plants ([5], [7] and [8]). All their approaches have been done only for simple fault events occurring locally into one component. On the other hand, recent work has generalized the property usually checked in diagnosability, i.e. the occurrence of a fault event, to the recognition of a pattern that can represent more general objectives such as multiple faults, ordered occurrence of significant events, multiple occurrences of the same fault, but which is analyzed in a centralized way ([3]).

In this paper, we propose an original distributed framework for pattern diagnosability verification. The idea is to find an equivalent alternative to the centralized pattern diagnosability checking that is more efficient in order to improve the scalability of the problem. Our approach contributes to the pattern diagnosability problem in several aspects. Firstly, we extend pattern diagnosability problem from the centralized framework to the distributed one. Secondly, pattern recognition can be checked by incrementally constructing pattern recognizers, which often concerns sev-

eral components. Thirdly, we propose a way to build pattern verifier that is inspired from the classical twin plant construction but with smaller search space and containing all necessary diagnosability information to search for partial critical paths, which are corresponding to critical paths in the global twin plant when they are globally consistent. Finally some key information about the reasons why the system is non diagnosable is provided by our algorithm when the system is non diagnosable, which can help the designer to improve the diagnosable level of the system by rearranging sensor placement.

The paper is organized as follows. In section 2, we model a distributed system as a set of finite state machines (FSM) and recall pattern and pattern diagnosability. Section 3 describes the pattern recognition and diagnosability analysis in a distributed way and then section 4 presents the formal algorithm. The result of our distributed approach compared to that obtained from the centralized one is evaluated in section 5.

## 2. Background

### 2.1. System model

We consider a distributed discrete event system $G$ composed of a set of components that can communicate with each other by communication events. The local model of a component is defined as below.

**Definition 1** *(Local Model). The local model of the component $i$ is a FSM $G_i = (Q_i, \Sigma_i, \delta_i, q_i^0)$, where*

- *$Q_i$ is the set of states*
- *$\Sigma_i$ is the set of events*
- *$\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the set of transitions*
- *$q_i^0$ is the initial state*

The set of events $\Sigma_i$ is divided into three disjoint parts: $\Sigma_{i_o}$ is the set of local observable events, $\Sigma_{i_u}$ is the set of local unobservable events and $\Sigma_{i_c}$ is the set of unobservable communication events that are shared by at least one other component. For any pair of distinct local components $G_i$ and $G_j$, we have $\Sigma_{i_o} \cap \Sigma_{j_o} = \emptyset$ and $\Sigma_{i_u} \cap \Sigma_{j_u} = \emptyset$, which means that any two different local components can only share communication events. For the transition set, it is easy to extend $\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ to $\delta_i \subseteq Q_i \times \Sigma_i^* \times Q_i$ by the following way: 1) $(q, \epsilon, q) \in \delta_i$, where $\epsilon$ is the null event; 2) $(q, se, q1) \in \delta_i$ if $\exists q\prime \in Q_i, (q, s, q\prime) \in \delta_i$ and $(q\prime, e, q1) \in \delta_i$, where $s \in \Sigma_i^*, e \in \Sigma_i$.

Figure 1 depicts a distributed system composed of three components $G_1$(top), $G_2$(bottom left) and $G_3$(bottom right). For $G_1$, we have $\Sigma_1 = \{c1, c2, u1, u2, o1, o2\}, \Sigma_{1_o} = \{o1, o2\}, \Sigma_{1_u} = \{u1, u2\}, \Sigma_{1_c} = \{c1, c2\}$. In the same way, we can get the event sets for $G_2$, $G_3$

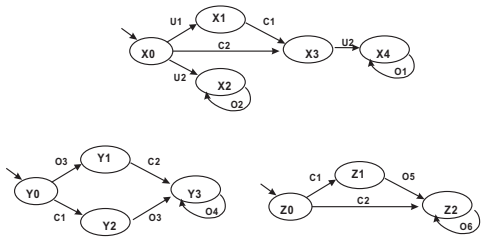


**Figure 1. A distributed system with three components $G_1$(top), $G_2$(bottom left) and $G_3$(bottom right).**

and for the whole distributed system as well. Now we define some useful operations. In the following, we have $G_1 = (Q_1, \Sigma_1, \delta_1, q_1^0)$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_2^0)$.

**Definition 2** *(Synchronization). Given two FSMs $G_1$ and $G_2$, their synchronization based on $\Sigma_s$, where $\Sigma_s$ is the set of synchronized events, is $G_1 \|_{\Sigma_s} G_2$=$(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1\|2}, (q_1^0, q_2^0))$, where $\delta_{1\|2}$ is defined as the following:*

- *$\delta_{1\|2}((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$ if $\sigma \in \Sigma_1 \wedge \sigma \in \Sigma_2 \wedge \sigma \in \Sigma_s$*
- *$\delta_{1\|2}((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), q_2)$ if $\sigma \in \Sigma_1 \wedge \sigma \notin \Sigma_2 \wedge \sigma \notin \Sigma_s$*
- *$\delta_{1\|2}((q_1, q_2), \sigma) = (q_1, \delta_2(q_2, \sigma))$ if $\sigma \notin \Sigma_1 \wedge \sigma \in \Sigma_2 \wedge \sigma \notin \Sigma_s$*

**Definition 3** *(Product). Given two FSMs $G_1$ and $G_2$, their product is $G_1 \times G_2 = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta_{1\times2}, (q_1^0, q_2^0))$, where $\delta_{1\times2}((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$ if both $\delta_1(q_1, \sigma)$ and $\delta_2(q_2, \sigma)$ are defined in $G_1, G_2$ respectively. Otherwise, $\delta_{1\times2}((q_1, q_2), \sigma)$ is undefined in $G_1 \times G_2$.*

**Definition 4** *(Delay Closure). Given a FSM $G$ $(Q, \Sigma, \delta, q^0)$, its delay closure with respect to $\Sigma_d$, where $\Sigma_d \subseteq \Sigma$, is $\mathbb{C}_{\Sigma_d}(G) = (Q, \Sigma_d, \delta_d, q^0)$, where $\delta_d(q, \sigma) = q\prime$, if $\exists \sigma \in \Sigma_d$, we have $\delta(q, s\sigma) = q\prime$ in G and $\forall \sigma\prime \in s, \sigma\prime \notin \Sigma_d$.*

For the synchronization, any synchronized event always occurs simultaneously in all concerned components. Different from the synchronization, the operation of product does not allow the case that some components stay in their own internal state whereas other components may change their state. As for the operation of delay closure with respect to $\Sigma_d$, we preserve the information about the events in $\Sigma_d$ while abstracting away irrelevant parts. The synchronization can be easily generalized to a set of FSMs using its associativity property. The global model of the distributed system in figure 1 is obtained by synchronizing its three local models: $G = \{Q, \Sigma, \delta, q^0\} = \|_{\Sigma_c}(G_i), i \in \{1, 2, 3\}$.

Given the global model $G$ of a distributed system, its prefix-closed language $L(G)$ describes the behaviors of the system, where $L(G) \subseteq \Sigma^*$. Formally, the language $L(G)$ is the set of words produced from $G$: $L(G) = \{s \in \Sigma^* | \exists q \in Q, (q^0, s, q) \in \delta\}$. In the following, we call a word from $G$ a trajectory in $G$ and a sequence $q_0\sigma_0 q_1\sigma_1...$ a path in $G$, where $\sigma_0\sigma_1...$ is a trajectory in G and $\forall i$, we have $(q_i, \sigma_i, q_{i+1}) \in \delta$. We call a FSM $G\prime$ a subsystem of $G$ if $L(G\prime) \subseteq L(G^s)$, where $G^s$ is the synchronized FSM on any non-empty set $\{G_{i_1}, ...G_{i_m}\}$ and $G_{i_j}, j \in \{1, ..., m\}$ could be any component of the system. If $\rho\prime$ is the projection of a trajectory (path) $\rho$ in $G$ on the subsystem $G\prime$, then we call $\rho\prime$ the subpart of $\rho$ in $G\prime$. If there is a set of final states $F$ in the system, we denote the marked language generated by $G$ by: $L_m(G) = \{s \in L(G) | \exists q \in F, (q^0, s, q) \in \delta\}$. In our paper, both the language and the marked language are assumed to be live observable, which means that there is no loop containing only unobservable events. Given $s \in L$, we denote the post-language of $L$ after $s$ by $L/s$ and denote the projection of the trajectory $s$ to observable events by $P(s)$. Then the inverse projection of an observation sequence $s$, denoted by $P^{-1}(s)$, returns the set of all trajectories whose observable projection is $s$.

## 2.2. Pattern diagnosability

In this section, we recall the notion of pattern for diagnosis problem and pattern diagnosability definition.

**Definition 5** *(Pattern). A pattern is a deterministic, complete FSM with a stable set of final states $F_\Omega$, $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q^0_\Omega, F_\Omega)$, where $F_\Omega \subseteq Q_\Omega$.*

Since $F_\Omega$ is stable, the marked language generated by $\Omega$ is "extension-closed", formally described as: $\forall s \in L_m(\Omega), \forall s\prime \in \Sigma^*_\Omega, ss\prime \in L_m(\Omega)$. So once $\Omega$ arrives in a final state, it will be always in a final state in the future. Note that $\forall s \in L_m(\Omega), \exists e \in s$ such that $e$ is unobservable. Otherwise, the diagnosability problem would be trivial.

In a pattern $\Omega$, we call an event $\sigma$ a significant event of $\Omega$ if $\exists (q, \sigma, q\prime) \in \delta_\Omega$ with $q \neq q\prime$. We use $\Theta_\Omega$ to denote the set of significant events of $\Omega$ and $\widehat{\varpi_q}$ to denote the set of events in $\Omega$ such that $\forall \sigma \in \widehat{\varpi_q}, \exists (q, \sigma, q\prime) \in \delta_\Omega, q \neq q\prime$. Thus $\widehat{\varpi_q}$ is actually the set of significant events of $\Omega$ whose source state in $\Omega$ is the state $q$.

**Definition 6** *(Simple Pattern). A pattern $\Omega$ is a simple pattern if $\forall q_\Omega, q_\Omega \in Q_\Omega \wedge q_\Omega \notin F_\Omega$, we have $|\widehat{\varpi_{q_\Omega}}| = 1$.*

For the sake of simplicity, we illustrate our approach by dealing with simple patterns, which can be easily extended to general ones. Even with simple patterns, the diagnosis problem can be generalized from detecting single fault event to recognizing event sequences that can describe more general objectives, such as ordered occurrence of significant events, multiple occurrences of the same fault [3]. Actually the fault event case is a special one of the pattern case.

Due to $F_\Omega$ being stable, we can merge all final states in one beforehand, which has no impact on our diagnosability analysis. In a simple pattern, the significant event whose source state is $q$ is denoted by $\varpi^s_q$. If $q$ is the final state, $\varpi^s_q = \epsilon$. Figure 2 depicts a pattern that describes the ordered occurrence of two significant events $\Theta_\Omega = \{u1, o3\}$. For this pattern, we have $Q_\Omega = \{p0, p1, p2\}$, $\Sigma_\Omega = \{c1, c2, u1, u2, o1, o2, o3, o4, o5, o6\}$, $\Sigma_{\Omega_o} = \{o1, o2, o3, o4, o5, o6\}$, $q^0_\Omega = p0, F_\Omega = \{p2\}$. Given a system G and a pattern $\Omega$, we assume $\Sigma = \Sigma_\Omega$,

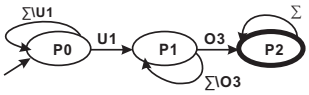


**Figure 2. Pattern $\Omega$.**

$\Sigma_o = \Sigma_{\Omega_o}$, $\Sigma_u = \Sigma_{\Omega_u}$ and $\forall \sigma \in \Theta_\Omega, \forall \sigma\prime \in \Sigma_c, \sigma\prime \neq \sigma$, where $\Sigma_c$ is the set of communication events in G. In other words, any significant event of the pattern is not a communication event, which has meaning in the sense that communication events function only for exchanging messages between components. A trajectory $s \in L(G)$ is recognized by $\Omega$ iff $s \in L_m(\Omega)$. The property of pattern diagnosability concerns the ability of a system to detect the trajectory recognized by a pattern with certainty, based on a sequence of observations.

**Definition 7** *(Pattern Diagnosability). A system G is $\Omega$-diagnosable if*

$$\exists n \in N, \forall s \in L(G) \cap L_m(\Omega), \forall t \in L(G)/s,$$
$$\text{if } |t| \geq n, \text{ then } P^{-1}P(s.t) \subseteq L_m(\Omega).$$

If a system $G$ is $\Omega$-diagnosable, then whenever a trajectory $s$ in $G$ is recognized by the pattern, for its any extension $t$ with enough finite events, any trajectory with the same observations as $s.t$ is also recognized by the pattern. In other words, $G$ is not $\Omega$-diagnosable if there exist two trajectories $p$ and $p\prime$ in $L(G)$ satisfying the following conditions: 1) $p \in L_m(\Omega)$ and $p\prime \notin L_m(\Omega)$; 2) $p$ is of arbitrarily long length after pattern recognition; 3) $P(p) = P(p\prime)$. A pair of such trajectories is called **a critical pair**. Note that all existing work for pattern recognition and diagnosability verification employs the global model of the system and is based on global twin plant construction.

The basic idea of the twin plant, described in [4], is to build a FSM that compares every pair of trajectories with the same observations to search for critical pairs. For centralized pattern diagnosability analysis [3], given a global model $G$ and the considered pattern $\Omega$, pattern recognition

is performed through the product operation on $G$ and $\Omega$. Then the global twin plant, shortly $GTP$, is constructed first by constructing its observer through delay closure and then by synchronizing the observer with itself. Then we have $GTP = \complement_{\Sigma_o}(G \times \Omega)\|_{\Sigma_s}\complement_{\Sigma_o}(G \times \Omega)$, where $\Sigma_s$ is the set of observable events. Obviously, each state of $GTP$, labeled with two pairs of system states with the associated pattern states, provides two possible pattern recognitions for the same observations. For a state $T_G$ in $GTP$, if its one associated pattern state is the final state of $\Omega$ while the other is not, then $T_G$ is an ambiguous state with respect to $\Omega$ since the recognition of $\Omega$ cannot be certain given the observations up to $T_G$. In $GTP$, if a path contains an ambiguous state cycle that includes at least one observable event, then it is called **a critical path**. It was proved that a critical path in $GTP$ corresponds to a critical pair in the system. Thus diagnosability verification can be performed by searching for critical paths in $GTP$.

## 3. Theoretical distributed framework

Now we show how to distribute the pattern recognition and its diagnosability verification on subsystems without computing the global objects. We begin the pattern recognition from the component with the event $\omega^s_{q^0_\Omega}$, where $q^0_\Omega$ is the initial state of the pattern. This component, also called the initial subsystem, contains the significant event of $\Omega$ whose source state in $\Omega$ is its initial state. If the pattern cannot be completely recognized in this component, the subsystem will be extended by propagating only diagnosability relative part to next selected component for next recognizer construction. And thus we continue to extend the subsystem until the recognition is completed. In this incremental way what we obtain is often a small subpart of the global model because the propagated part is often much smaller than current subsystem and that pattern concerned components are normally a subset of system components.

The idea of centralized pattern diagnosability approach is to check the existence of a critical path in $GTP$. Our distributed approach is to avoid $GTP$ calculation by computing pattern verifier for the subsystem where the pattern recognition is completed, which is inspired from twin plant construction but with smaller search space. Then we demonstrate that a partial critical path in this verifier corresponds to one in $GTP$ when it does not disappear after global consistency checking. In this distributed way, the final obtained state machine for diagnosability verification is normally a quite small subpart of $GTP$.

### 3.1. Pattern recognizer construction

Pattern recognition in a subsystem is performed by constructing the pattern recognizer, which is defined as below.

**Definition 8** *(Pattern Recognizer). Given a subsystem* $G\prime = (Q\prime, \Sigma\prime, \delta\prime, q\prime^0)$ *and a pattern* $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q^0_\Omega, F_\Omega)$*, then the pattern recognizer of* $G\prime$ *is* $R_{G\prime} = (Q_{R_{G\prime}}, \Sigma_{R_{G\prime}}, \delta_{R_{G\prime}}, q^0_{R_{G\prime}}, F_{R_{G\prime}}) = G\prime \times \Omega$*, where the initial state is* $q^0_{R_{G\prime}} = (q\prime^0, q^0_\Omega)$*,* $F_{R_{G\prime}} = (Q\prime \times F_\Omega) \cap Q_{R_{G\prime}}$ *is the set of final states.*

Since $\Omega$ is a complete FSM, we have $L(\Omega) = \Sigma^*$ and thus $L(R_{G\prime}) = L(G\prime) \cap L(\Omega) = L(G\prime)$. So the pattern recognizer can show which part of the pattern can be recognized after any trajectory in the subsystem. Given the initial recognizer state $q^0_{R_{G\prime}} = (q\prime^0, q^0_\Omega)$, a state $(q, q_\Omega)$ is called a suspicious state of $R_{G\prime}$ if $q^0_\Omega \neq q_\Omega$. Given the set of all suspicious states in $R_{G\prime}$, $\omega = \{(q_1, q^1_\Omega),...(q_n, q^n_\Omega)\}$, a state $(q_i, q^i_\Omega) \in \omega$ is called a Target Suspicious State(TSS) of $R_{G\prime}$ if in $\Omega$, an event sequence through which $q_f$ can be reached from $q^i_\Omega$ contains the minimal number of significant events compared to all $q^k_\Omega$, where $k \in \{1, ...n\}$ and $q_f$ is the final state of $\Omega$. TSS is thus defined to show which part of the pattern can be recognized in the current subsystem. If $F_{R_{G\prime}} \neq \emptyset$, then the pattern is recognized in $G\prime$ and $R_{G\prime}$ is called the complete recognizer, denoted by $R_c$. On the other hand, if $F_{R_{G\prime}} = \emptyset$, then the pattern is not yet recognized in $G\prime$ and we should choose next component to extend the subsystem for building the next recognizer. As said before, the initial subsystem is the component that contains the event $\varpi^s_{q^0_\Omega}$. Suppose that $q^c$ is the pattern state contained in a TSS of the pattern recognizer of the current subsystem, then the next component to be selected for subsystem extension is the one with the event $\omega^s_{q^c}$, which is the significant event whose source state in $\Omega$ is $q^c$. Consider that any significant event is not a communication event, this selected component is always unique.

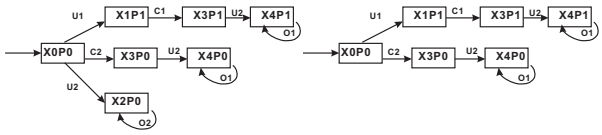


**Figure 3. The pattern recognizer $R_{G\prime}$(left) and its part $R^\Omega_{G\prime}$(right)**

For our example, the pattern recognizer of the initial subsystem, denoted by $R_{G\prime}$, is depicted by the left part of figure 3. Actually the initial subsystem is the component $G_1$, which contains the event $\varpi^s_{q^0_\Omega} = \varpi^s_{p0} = u1$. In this recognizer, the states whose pattern state part is $p1$ are TSSs, which means that in the initial subsystem, the recognized pattern part is the part from the initial state $p0$ to the state $p1$. Then the next selected component should contain the event $\omega^s_{p1} = o3$.

### 3.2. Diagnosability information propagation

A non complete recognizer normally can only recognize a part of the pattern. The recognition should be completed by incrementally extending the subsystem. Given a non complete recognizer $R_{G\prime}$, we reduce it to $R^{\Omega}_{G\prime}$, denoting the part where all the states are either on a path containing at least one TSS of $R_{G\prime}$ or on a path with the same observations as one containing a TSS. Since $L(R_{G\prime}) = L(G\prime)$ and $L(R^{\Omega}_{G\prime}) \subseteq L(R_{G\prime})$, then we have $L(R^{\Omega}_{G\prime}) \subseteq L(G\prime)$. To recognize the next part of the pattern, we extend the current subsystem by synchronizing $R^{\Omega}_{G\prime}$ with $G_j$, $R^{\Omega}_{G\prime} \| _{\Sigma_s} G_j$, where $G_j$ is the next selected component and $\Sigma_s$ is the set of shared events of all involved components. Note that in a real complex system, $R^{\Omega}_{G\prime}$ is normally a small subpart of $G\prime$, $L(R^{\Omega}_{G\prime}) \subset L(G\prime)$, and the propagated part is actually quite limited. Thus the synchronized state machine contains smaller state space compared to that obtained by synchronizing the whole subsystem with the next component. If the pattern can be recognized in the system, then the recognizer of the extended subsystem can achieve next recognition of the pattern.

There are two intentions of diagnosability information propagation. One is to continue the recognition process. Another is to retain the information about critical pairs to facilitate the diagnosability analysis.

For our example, the right part of figure 3 depicts the diagnosability relative part $R^{\Omega}_{G\prime}$. There is one trajectory in $R_{G\prime}$ that is not in $R^{\Omega}_{G\prime}$, which can never be recognized by the pattern when it synchronizes with other components and can never be the one with the same observations as those recognized by the pattern. Thus it is relative to neither the pattern recognition nor the pattern diagnosability verification. The top part of figure 4 presents a part of $R^{\Omega}_{G\prime} \| _{\Sigma_s} G_2$, $\Sigma_s = \{c1, c2\}$. Then we construct the pattern recognizer of this extended subsystem, partly depicted by the bottom part of figure 4. It is actually the complete recognizer since it contains final states and thus the pattern can be recognized in the current subsystem.

### 3.3. Pattern verifier construction

Next we will show how to distribute diagnosability verification based on pattern verifier construction as well as its global consistency checking, which normally has much smaller search state space than $GTP$. The result will be analyzed in section 5. Now we provide some lemmas that are the basis of proving the correctness of our distributed method.

From non intersection of local observable events between components, we can directly get the following lemma.

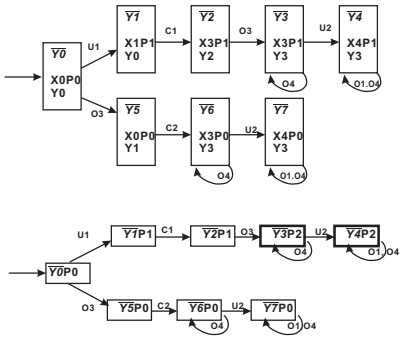


**Figure 4. Part of the synchronization $R^{\Omega}_{G\prime} \| _{\Sigma_s} G_2$(top) and part of the complete recognizer $R_c$ (bottom).**

**Lemma 1** *Any critical pair in the global model has the same observable projection on each component.*

Then we can obtain the next lemma.

**Lemma 2** *The complete recognizer $R_c$ contains the subparts in the corresponding subsystem of all critical pairs in the global model.*

**Proof** :
Let $G_i$ be the initial subsystem. Due to the fact that the only shared events between components are communication events and any significant event of the pattern is not a communication event, then any significant event of the pattern is contained in only one component. In the pattern recognizer $R_{G_i}$, the paths not containing a TSS of $R_{G_i}$ are never be the subparts of those that can be recognized by the pattern because the significant event leading to the pattern state contained in a TSS cannot be in any other component. In other words, the set of paths in $R_{G_i}$ with a TSS must contain the subparts in $G_i$ of all those that can be recognized by the pattern. From lemma 1, we get that any critical pair has the same observable projection on $G_i$. Thus since in $R^{\Omega}_{G_i}$, all the paths having the same observations as those containing a TSS are also retained, we know that $R^{\Omega}_{G_i}$ contains the subparts in the initial subsystem $G_i$ of all critical pairs. Then after synchronizing $R^{\Omega}_{G_i}$ with the next selected component, suppose $G_j$, again from lemma 1 and in the same way as above, we can deduce that the recognizer of the extended subsystem contains the subparts in the extended subsystem of all critical pairs. We repeat the above steps until the subsystem corresponding to the complete recognizer is obtained. It can be proved that the complete recognizer must contain the subparts in the corresponding subsystem of all critical pairs in the global model.

Before pattern verifier construction, we refine the complete recognizer by the delay closure with respect to $\Sigma_d$,

where $\Sigma_d$ is the set of communication events and observable events, $R_r = \mathtt{C}_{\Sigma_d}(R_c)$. We obtain left instance of $R_r$, denoted by $R_r^l$, by prefixing the communication events with $L$ and then by retaining only the paths with final states of the recognizer. Then we get the right instance of $R_r$, denoted by $R_r^r$, by prefixing the communication events with $R$ and then by retaining the paths without final states. The pattern verifier can be constructed by synchronizing the left instance and the right instance based on all observable events in $R_r$.

**Definition 9** *(Pattern Verifier). Given the refined recognizer $R_r$, its corresponding pattern verifier is $V = R_r^l \| _{\Sigma_s} R_r^r$, where $\Sigma_s$ is the set of observable events in $R_r$.*

The main difference between the pattern twin plant and the pattern verifier is that twin plant construction is based on the synchronization of the recognizer with itself (left instance and right instance are actually identical and non-reduced), but the pattern verifier can simplify this synchronization by reducing left and right instances as above. In the pattern verifier, if a path $\varrho$ contains an ambiguous state cycle that contains at least one local observable event for all involved components, then $\varrho$ is called a partial critical path. Any partial critical path is corresponding to a critical pair in the global model if it is globally consistent. From the fact that $R_r^l$ contains all paths with final states of the recognizer and $R_r^r$ contains all paths without final states, and from lemma 2 and the way to construct the pattern verifier, we can prove the following lemma in a straightforward way.

**Lemma 3** *If there is no partial critical path in the pattern verifier, then there is no critical path in $GTP$.*

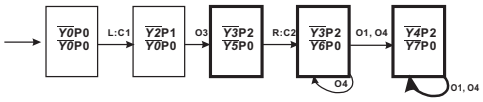


**Figure 5. part of the pattern verifier $V$.**

For our example, figure 5 partly depicts the pattern verifier. This part is actually a partial critical path since it contains an ambiguous state cycle with one local observable event for both involved components.

### 3.4. Global consistency checking

From lemma 3, the existence of critical paths in $GTP$ implies the existence of partial critical paths in the pattern verifier but not each obtained partial critical path is corresponding to a critical path in $GTP$. The reason is that for now we do not take into account the communication with the neighborhood in the whole system. To solve this, we first define the local twin checker of a component, which is to obtain all possible pairs of local trajectories with the same observations in the component, and then synchronize the partial critical paths with the local twin checkers of the connected components to check their global consistency. Obviously, a partial critical path is corresponding to a critical path in $GTP$ when it is globally consistent.

The local twin checker of a component is obtained first by operating delay closure with respect to the set of communication events and observable events on its local model and then by synchronizing the obtained local model with itself (left and right instances) based on its observable events. We distinguish non-synchronized events by prefixing them with $L$ and $R$.

**Definition 10** *(Local Twin Checker). The local twin checker of $G_i$ is $C_i = (\mathtt{C}_{\Sigma_d}(G_i))^l \| _{\Sigma_s} (\mathtt{C}_{\Sigma_d}(G_i))^r$, where $\Sigma_d = \Sigma_{i_o} \cup \Sigma_{i_c}$ and $\Sigma_s = \Sigma_{i_o}$.*

**Algorithm 1** Global Consistency Checking

```
1:  INPUT: component models G_1, ..., G_n of the system G;
    pattern verifier V
2:  V ← Reduce(V)
3:  while ∃ a connected component G_j not involved in V
    do
4:      C_j ← ConstructLTC(G_j)
5:      V ← V ||_{Σ_s} C_j
6:      V ← Reduce(V)
7:  if V ≠ ∅ then
8:      return V
9:  else
10:     return "The system is diagnosable."
```

Algorithm 1 provides the pseudo-code of global consistency checking for the pattern verifier. Firstly, the pattern verifier $V$ is reduced by only retaining all partial critical paths (line 2). Secondly, when there is a component $G_j$ non involved in $V$ that contains at least one communication event contained also in reduced $V$, then repeat the following steps: 1) construct its local twin checker $C_j$ to obtain $V \| _{\Sigma_s} C_j$, where $\Sigma_s$ is the set of common events of $V$ and $C_j$ (left communication events, right communication events). The states in the synchronized state machine that contain an ambiguous state from the pattern verifier are also called ambiguous states; 2) reduce the obtained state machine by only retaining all partial critical paths. Here the partial critical paths are those having an ambiguous state cycle containing at least one local observable event for all involved components (line 3-6). Finally, when there is no other connected component, any partial critical path obtained in the final state machine is globally consistent since all its communication events are validated. If there is at least one such path, which means that the final state machine is not empty,

then it is returned to provide the non-diagnosability information. Otherwise, the diagnosable information is returned (line 7-10).

**Lemma 4** *A partial critical path in the pattern verifier is globally consistent iff it is a subpart in the corresponding subsystem of a critical path in $GTP$.*

**Proof** :
($\Rightarrow$) Suppose that a partial critical path $\rho$ is globally consistent and that $\rho$ is not a subpart of a critical path in $GTP$. Since $\rho$ is not corresponding to any critical path in $GTP$, from the way to construct $GTP$, described in section 2.2 and from the way to check global consistency, it is easy to know that $\rho$ will disappear after global consistency checking, which means that $\rho$ is not globally consistent and thus contradicts the assumption.
($\Leftarrow$) Suppose now that a partial critical path $\rho$ is not globally consistent and that it is a subpart of a critical path in $GTP$. From the non global consistency of $\rho$ it follows that it will disappear after global consistency checking, which means that not all communication events in $\rho$ are valid. However, all communication events in any critical path in $GTP$ are valid because $GTP$ is constructed from the global model. This implies that $\rho$ is not a subpart of a critical path in $GTP$, which contradicts the assumption.

Inspired from the result of the centralized approach for pattern diagnosability [3], we get the following theorem.

**Theorem 1** *Given a pattern $\Omega$ and the global model of a system G, G is $\Omega$-diagnosable iff there is no critical path in $GTP$.*

Then from lemma 3, lemma 4 and theorem 1, we can directly obtain the following theorem to verify pattern diagnosability in a distributed way.

**Theorem 2** *A distributed system G is $\Omega$-diagnosable iff there is no partial critical path in the pattern verifier that is globally consistent.*

For our example, after global consistency checking, the partial critical path depicted in figure 5 will disappear due to its synchronization with the local twin checker of the component $G_3$, which means that there is no corresponding critical path in $GTP$. In the same way, after checking all the rest part of the pattern verifier, we know that there is no partial critical path that is globally consistent. Thus from theorem 2, the system is verified to be $\Omega$-diagnosable.

## 4. Algorithm

This section presents the algorithm of pattern diagnosability verification in a distributed way without calculating $GTP$. Algorithm 2 gives the pseudo-code of the verification procedure. After the initialization of the parameters, when the current recognizer is not the complete one, which means that for the moment the pattern is not yet recognized in the current subsystem and we need to exploit the other components, the algorithm repeatedly performs the following steps:

**Algorithm 2** Pattern Diagnosability Verification Algorithm for Distributed Systems

```
1: INPUT: component models G_1, ..., G_n of the system G; the pattern Ω to be diagnosed in G
2: Initializations: q^c ← q^0_Ω (currently recognized pattern state); R ← ∅ (current recognizer); V ← ∅ (pattern verifier)
3: while R is not the complete recognizer do
4:    R ← REDUCE(R)
5:    G_i ← SelectCom(G_1, ...G_n, Ω, q^c)
6:    G_i ← G_i ||_{Σ_s} R, where Σ_s is the set of common events of all involved components including G_i
7:    R ← ConstructPR(G_i, Ω)
8:    if q^c = q', where q' is the pattern state part contained in a TSS of R then
9:       return "Ω cannot be recognized in G."
10:   else
11:      q^c ← q'
12: R ← Refine(R)
13: V ← ConstructPV(R)
14: CheckGlobalConsistency(G, V)
```

1. The current recognizer is reduced as described in section 3.2, doing nothing for the current recognizer being empty (for the first time), and then the next component $G_i$ is selected for extending the subsystem. The selection strategy is described in section 3.1. (line 4, 5)
2. The reduced recognizer is synchronized with the selected component and the pattern recognizer of this synchronized one is constructed again. (line 6, 7)
3. If the pattern state part in a TSS of the current recognizer is the same as the value in $q^c$, which means that the current subsystem can recognize the same part of the pattern as the previous subsystem. From the way to choose the component for synchronization and its uniqueness, we can deduce that the next part of the pattern cannot be recognized in the system and thus the whole pattern cannot be recognized in G. Otherwise, we update $q^c$ by assigning the pattern state in a TSS of the current recognizer. (line 8-11)

When the complete recognizer is obtained, we check pattern diagnosability first by refining this recognizer through the delay closure as described in section 3.3 and then by constructing the corresponding pattern verifier for global

consistency checking (line 12-14). There are several causes that can stop this algorithm: 1) pattern cannot be recognized in the system, then the algorithm returns non-recognizability information; 2) pattern can be recognized but not diagnosable, then the returned state machine of function CheckGlobalConsistency(G, T), which is described in algorithm 1, contains the information about the undistinguishable behaviors that cause the non-diagnosability of the pattern; 3) pattern can be recognized and diagnosable, then the function CheckGlobalConsistency(G, T) returns diagnosability information.

## 5. Result

In the centralized approach ([3]), all components are first synchronized to get the global model and then the global pattern recognizer is constructed before calculating $GTP$. It is too expensive to apply the centralized approach for real complex systems because the size of state space of the global model risks an exponential growing with the number of system components. Our distributed method avoids calculating global objects in the following way: 1) Let $G^m$ be the $m^{th}$ obtained subsystem. We have $G^1 = G_{i_1}$, where $G_{i_1}$ is the initial subsystem, and $G^k = R^{\Omega}_{G^{k-1}} \| _{\Sigma_s} G_{i_k}$, where $1 < k \leq n$, $\Sigma_s$ is the set of common events of all involved components and $G_{i_k}$ is the $k^{th}$ selected component for extending the subsystem. As analyzed in section 3.2, normally we have $L(R^{\Omega}_{G^{k-1}}) \subset L(G^{k-1})$. While for the centralized approach, the pattern concerned part of the global model should be $\| _{\Sigma_s}(G_{i_k})$, where the synchronized events $\Sigma_s$ are their common events and $1 \leq k \leq n$. In other words, begin from the initial subsystem, each time for extending subsystem, we only propagate the diagnosability relative part, which is actually a subpart of current subsystem, to the next selected component and in this way the subsystem obtained corresponding to the complete recognizer is only a small subpart of concerned part in the global model. 2) To construct pattern verifier, we synchronize the reduced left instance and the reduced right instance of the complete recognizer, which keeps all necessary diagnosability information but makes the search state space smaller compared to the twin plant adopted by the centralized approach, where the non-reduced left instance and non-reduced right instance are synchronized. 3) If the pattern relative components do not include all components $\{G_{i_1}, ...G_{i_k}\} \subset \{G_1, ...G_n\}$ and there are other connected components, we minimize the search state space in final state machine by retaining only partial critical paths every time before synchronizing with the connected local twin checker since what we are interested in is the existence of the globally consistent partial critical paths. In this way, what we finally obtain is a reasonably small and necessary portion of $GTP$, where there is even no information about those components that are not connected to the partial critical paths.

## 6. Conclusion

Our paper proposes an original theoretical framework to solve pattern diagnosability considering the distributed nature of complex systems. We incrementally recognize a pattern by propagating the diagnosability relative information to avoid building the global model. Then inspired from twin plant method, we construct pattern verifier that has smaller search space to find partial critical paths before checking their global consistency. A new distributed verification algorithm is thus proposed. Our approach can be extended to general patterns in a straightforward way by checking the existence of a critical pair of trajectories such that exactly one of them is recognized by a given general pattern. The perspective of this work can be the extension to predictability analysis of patterns in a distributed way, which is a stronger property because predictability is a sufficient but not necessary condition of diagnosability ([2]).

## References


[1] C. G. Cassandras and S. Laforture. *Introduction to Discrete Event Systems*. Springer-Verlag, New York Inc., 2007.

[2] T. Jéron, H. Marchand, S. Genc, and S. Laforture. Predictability of sequence patterns in discrete event systems. *Proceedings of the 17th World Congress*, July 2008.

[3] T. Jéron, H. Marchand, S. Pinchinat, and M. O. Cordier. Supervision patterns in discrete event systems diagnosis. *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 262–268, July 2006.

[4] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control*, 46:1318–1321, August 2001.

[5] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. *Proceedings of European Conference on Artificial Intelligent*, pages 43–47, August 2004.

[6] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete event system. *IEEE Transactions on Automatic Control*, 40:1555–1575, 1995.

[7] A. Schumann and J. Huang. A scalable jointree algorithm for diagnosability. *Proceedings of AAAI Conference on Artificial Intelligence*, page 535C540, July 2008.

[8] A. Schumann and Y. Pencolé. Scalable diagnosability checking of event-driven systems. *Proceedings of International Joint Conference on Artificial Intelligence*, pages 575–580, January 2007.

[9] T. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *IEEE Transactions on Automatic Control*, 47:1491–1495, September 2002.