A general algorithm for pattern diagnosability of distributed discrete event systems

Lina Ye, Philippe Dague

A General Algorithm for Pattern Diagnosability of Distributed Discrete Event Systems

Lina YE and Philippe DAGUE
LRI, UMR8623, Univ. Paris-Sud, CNRS, Orsay, F-91405
lina.ye@lri.fr, philippe.dague@lri.fr

***Abstract*—Diagnosability is an important system property that determines at design stage how accurate any diagnostic reasoning can be on a partially observed system. A fault in a discrete-event system is diagnosable iff its occurrence can always be deduced from enough observations. It is well known that centralized diagnosability approaches lead to combinatorial explosion of the search space since they assume the existence of a monolithic model of the system. This is why very recently the distributed approaches for diagnosability began to be investigated, relying on local objects. On the other hand, diagnosis objectives are generalized from fault event to fault pattern that can represent multiple faults, repeating fault, sequences of significant events, repair of faults, etc. For pattern case, most existing approaches are centralized. In this paper, we propose a new distributed framework for pattern diagnosability. We first show how to recognize patterns by incrementally constructing local pattern recognizers through extended subsystems. Then we propose a structure called regional pattern verifier that is constructed from the subsystem where the pattern is completely recognized before showing how to abstract just the necessary and sufficient diagnosability information to further save the search space. Then the global consistency checking is based on another local structure called abstracted local twin checker to analyze pattern diagnosability. In this way, we avoid constructing global objects both for pattern recognition and for pattern diagnosability. The correctness of our distributed algorithm is theoretically proved and its efficiency experimentally demonstrated by the results of the implementation.**

I. Introduction

Fault diagnosis is a crucial and challenging task in the automatic control of large complex systems. However, diagnosis decision could be necessarily ambiguous, in which case running a diagnosis engine may get a wrong decision. So it is very important to decide at design stage how accurate any diagnosis algorithm can be on a given partially observable system. This problem is called diagnosability analysis and is the basic question that underlies diagnosis.

The diagnosability analysis problem has received considerable attention in the literature. Some existing works analyze diagnosability in a centralized way ([1], [2] and [3]), i.e., the knowledge of the monolithic model of a given system is hypothesized, which is a very powerful information for diagnosability analysis. However, real systems are steadily growing in terms of size, complexity and interactions. The centralized diagnosability approaches lead to combinatorial explosion of the search space. This is why very recently the distributed approaches for diagnosability began to be investigated ([4], [5], [6], etc.), relying on local objects. More precisely, original diagnosability information can be obtained from the component where the fault may occur and then the global decision is calculated by checking its global consistency. All above approaches assume that the fault is a predefined event resulting in unexpected system behavior. However, sometimes the fault can be a sequence of some important events while any single one of them is not the fault by itself. A new proposal in the centralized case is provided by [7], who formally introduce the notion of supervision pattern, simply called pattern, that is general enough to cover an important class of diagnosis objectives, e.g. diagnosing multiple faults, repeating faults, sequences of significant events, repair of faults, etc. A fault event is a special case of pattern.

In this paper, we propose a new and efficient distributed method for pattern diagnosability analysis of discrete event systems. First we extend pattern diagnosability problem from centralized framework to distributed one. Then we show how to recognize patterns by incrementally constructing local pattern recognizers for extended subsystems. More precisely, the subsystem is extended by synchronizing the diagnosability relative part with next selected component. In this way we can avoid global model. Next we propose a structure called regional pattern verifier that is constructed from the subsystem where the pattern is completely recognized before showing how to abstract just the necessary and sufficient diagnosability information to further save the search space. Then the global consistency checking of the retained part is based on another local structure called abstracted local twin checker to check pattern diagnosability. In this way, we avoid constructing global objects both for pattern recognition and for pattern diagnosability verification. Our 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.

II. Preliminaries

In this section, we define the system model, recall pattern diagnosability of discrete event systems as well as the centralized approach.

A. System Model

We consider a distributed discrete event system G composed of a set of components $G_1, ..., G_n$ that communicate with each other by communication events. Each component is modeled by a finite state machine (FSM), denoted by

$G_i = (Q_i, \Sigma_i, \delta_i, q_i^0)$, where Q_i is the set of states, Σ_i is the set of events, $\delta_i \subseteq Q_i \times \Sigma_i \times Q_i$ is the set of transitions (the same notation will be kept for its natural extension to words of Σ_i^*) and q_i^0 is the initial state. The set of events Σ_i is divided into three disjoint parts: Σ_{i_o} the set of observable events, Σ_{i_u} the set of unobservable events and Σ_{i_c} the set of unobservable communication events that are shared by at least one other component (communication events are assumed to be unobservable because we target general systems with a mixture of observable and unobservable communication events but, for sake of simplicity, we deal here with the case where all communication events are unobservable, as the observable case is easy and so the general case can be easily derived from the existing work ([4] and [6]). 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 components only share communication events and their observable events and unobservable events are disjoint.

It is important to notice that, if the model is distributed, the observations are centralized, i.e. accessible to one global observer: the case with several partial observers is completely different for diagnosability analysis, which becomes in particular undecidable ([8]).

Two composition operations are defined as follows. For the sake of simplicity, they are presented for two FSMs but it is easy to generalize them for a set of FSMs using the associativity properties [9].

Definition 1: (Synchronization). Given two FSMs $G_1 = (Q_1, \Sigma_1, \delta_1, q_1^0)$ and $G_2 = (Q_2, \Sigma_2, \delta_2, q_2^0)$, their synchronization 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 $\Sigma_s = \Sigma_1 \cap \Sigma_2$ is the set of shared events, which can be omitted when there is no ambiguity in the context, and $\delta_{1\|2}$ is defined as follows:

- $((q_1, q_2), \sigma, (q_1', q_2')) \in \delta_{1\|2}$, if $\sigma \in \Sigma_s, (q_1, \sigma, q_1') \in \delta_1$ and $(q_2, \sigma, q_2') \in \delta_2$;
- $((q_1, q_2), \sigma, (q_1', q_2)) \in \delta_{1\|2}$, if $\sigma \in \Sigma_1 \backslash \Sigma_s$ and $(q_1, \sigma, q_1') \in \delta_1$;
- $((q_1, q_2), \sigma, (q_1, q_2')) \in \delta_{1\|2}$, if $\sigma \in \Sigma_2 \backslash \Sigma_s$ and $(q_2, \sigma, q_2') \in \delta_2$.

Definition 2: (Product). Given two FSMs G_1 and G_2, their product is $G_1 \times G_2 = (Q_1 \times Q_2,\ \Sigma_1 \cup \Sigma_2,\ \delta_{1\times 2},\ (q_1^0, q_2^0))$, where $((q_1, q_2), \sigma, (q_1', q_2')) \in \delta_{1\times 2}$ iff $(q_1, \sigma, q_1') \in \delta_1$ and $(q_2, \sigma, q_2') \in \delta_2$.

The operation of product is sometimes called complete synchronization. The main difference between the two operations is how the private events, i.e., the events not in $\Sigma_1 \cap \Sigma_2$, are handled. In the product, the transitions of the two FSMs must always be synchronized on a shared event, $\sigma \in \Sigma_1 \cap \Sigma_2$. In other words, an event in the product occurs iff it occurs in both FSMs. In the synchronization, the two FSMs are still synchronized on the shared events but the private events can independently be executed whenever possible. We denote the synchronized FSM of components $G_1, ..., G_n$ as $\|(G_1, ..., G_n)$, which is actually the monolithic model of the entire system with $\Sigma = \cup_i \Sigma_i$ and $\Sigma_o = \cup_i \Sigma_{i_o}$, also called the global model in the following. Then we define the operation called delay closure with respect to a set of events, that preserves all information about this set by abstracting away irrelevant parts.

Definition 3: (Delay Closure). Given a FSM G = (Q, Σ, δ, q^0), its delay closure with respect to a set of events $\Sigma_d \subseteq \Sigma$ is $\mathsf{C}_{\Sigma_d}(G) = (Q, \Sigma_d, \delta_d, q^0)$ where $(q, \sigma, q') \in \delta_d$ iff $\exists s \in (\Sigma \backslash \Sigma_d)^*, (q, s\sigma, q') \in \delta$.

Figure 1 presents a simple distributed system composed of three components, where observable events are denoted by Oi, unobservable events by Ui and unobservable communication events by Ci. The global model, denoted by G, is implicitly defined as the synchronization of the three components, where the set of synchronized events are communication events, denoted by $G = \|(G_1, G_2, G_3)$.

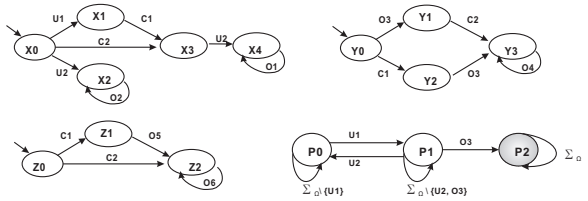

Fig. 1. A distributed system composed of three components: G_1 (top left), G_2 (top right), G_3 (bottom left), and a pattern Ω to be diagnosed (bottom right).

Given a system model G, the prefix-closed language $L(G) \subseteq \Sigma^*$ of words produced by the FSM G describes the normal and faulty behaviors of the system. Formally, $L(G) = \{s \in \Sigma^* | \exists q \in Q, (q^0, s, q) \in \delta\}$. If there is a set F of final states in the FSM, then 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 the following, we call a word of $L(G)$ a **trajectory** in the system G and a sequence $q_0\sigma_0 q_1 \sigma_1 ...$ a **path** in G, where $\sigma_0\sigma_1 ...$ is a trajectory and, for all i, $(q_i, \sigma_i, q_{i+1}) \in \delta$. Given $s \in L(G)$, we denote the post-language of $L(G)$ after s by $L(G)/s$ and denote the projection of the trajectory s to observable events by $P(s)$. In our approach, we adopt the following assumption: the language of each component is observable live, i.e. it is live and there is no cycle with only unobservable events.

B. Pattern Diagnosability of Discrete Event Systems

Now we recall the notion of pattern for diagnosis problem and pattern diagnosability of discrete event systems ([7]).

Definition 4: (Pattern). A pattern is a deterministic, complete FSM Ω with a stable set F_Ω of final states, $\Omega = (Q_\Omega, \Sigma_\Omega, \delta_\Omega, q_\Omega^0, F_\Omega)$.

Since F_Ω is stable, the marked language generated by Ω 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 Ω arrives in a final state, it will always be in a final state in the future. With pattern definition, the diagnosis problem can be generalized from detecting fault events to recognizing event sequences that can describe more general objectives, like ordered occurrence of significant events, multiple occurrences of the same fault, the

repair of a fault, etc ([7]). The fault event case is a special one of the pattern case.

In a given pattern Ω, we call an event σ a significant event of Ω if $\exists(q,\sigma,q\prime) \in \delta_\Omega$ with $q \neq q\prime$, i.e., any event that can change pattern state. We use Θ_Ω to denote the set of significant events of Ω and $\widehat{\varpi_q}$ to denote the set of events $\sigma \in \Sigma$ such that $\exists(q,\sigma,q\prime) \in \delta_\Omega, q \neq q\prime$. Thus $\widehat{\varpi_q}$ is actually the set of significant events of Ω that change the state q.

Given a system G and a pattern Ω, it is assumed that $\Sigma = \Sigma_\Omega$, $\Sigma_o = \Sigma_{\Omega_o}$, $\Sigma_u = \Sigma_{\Omega_u}$. We say that Ω is recognized by a trajectory in the system $s \in L(G)$ iff $s \in L_m(\Omega)$. The property of pattern diagnosability concerns the ability of a system to detect a trajectory recognizing the pattern with certainty, based on a sequence of observations. We assume that $\exists\sigma \in \Theta_\Omega$ such that σ is unobservable, which means that at least one significant event is unobservable: otherwise, the diagnosability problem would be trivial. For example, figure 1 depicts an example of such a system and a pattern. Here in the pattern (bottom right) $\Sigma_\Omega = \{C1, C2, U1, U2, O1, O2, O3, O4, O5, O6\}$, which is the same set of events as that of the system. And the final state set of the pattern is $\{P2\}$. We can see that the recognition of this pattern requires the ordered occurrences of the significant events $U1$, $O3$, where any event except $U2$ is allowed between them and any event is allowed before and after them.

Definition 5: (Pattern Diagnosability). A pattern Ω is diagnosable in a system G (we say that G is Ω-diagnosable) iff

$$\exists n \in N, \forall s \in L(G) \cap L_m(\Omega), \forall t \in L(G)/s, \text{ if } |t| \geq n, \text{ then}$$
$$\forall p \in L(G), P(p) = P(s.t) \Rightarrow p \in L_m(\Omega).$$

A system G is Ω-diagnosable iff for any trajectory s in G recognizing the pattern and for any extension t of s with enough events, any trajectory with the same observations as $s.t$ also recognizes the pattern. A pair of trajectories $p, p\prime$ satisfying the following conditions is called a critical pair, also a **global critical pair** in the following considering that it relates to the global model: 1) $p \in L_m(\Omega)$ and $p\prime \notin L_m(\Omega)$; 2) p is of arbitrarily long length after pattern recognition; 3) $P(p) = P(p\prime)$. The existence of such a global critical pair witnesses non Ω-diagnosability of the system. Thus pattern diagnosability checking is to search for global critical pairs.

C. Centralized Method

A centralized method for pattern diagnosability checking is proposed in [7], where the existence of the global model is assumed. Then the pattern recognition is based on the construction of global pattern recognizer.

Definition 6: (Global Pattern Recognizer). Given a global model $G = (Q, \Sigma, \delta, q^0)$ and a pattern Ω, then the global pattern recognizer is $R_G = G \times \Omega$, where the initial state is (q^0, q^0_Ω) and the set F_{R_G} of final states is $(Q \times F_\Omega) \cap Q_{R_G}$ (Q_{R_G} is the set of states in R_G).

Since Ω is a complete deterministic FSM, we have $L(\Omega) = \Sigma^*$ and $L(R_G) = L(G) \cap L(\Omega) = L(G)$. So the global pattern recognizer shows which part of the pattern can be recognized by any trajectory in the system. If $F_{R_G} \neq \emptyset$, we say that the pattern can be recognized in the system. After pattern recognition, pattern diagnosability is analyzed based on the structure called global pattern verifier.

Definition 7: (Global Pattern Verifier). The global pattern verifier for a given global pattern recognizer R_G, denoted by V_G, can be obtained by $V_G = \mathsf{C}_{\Sigma_o}(R_G) \| _{\Sigma_o} \mathsf{C}_{\Sigma_o}(R_G)$.

To construct the global pattern verifier, the delay closure with respect to the set of observable events is first performed on the global pattern recognizer and then the resulted FSM is synchronized with itself based on the set of observable events. The idea is to obtain all pairs of trajectories with the same observations to search for global critical pairs. In V_G, each state is a pair of recognizer states that provide two possible pattern recognitions. Given a verifier state composed of two recognizer states, if only one of them is a final state, which means that the recognition of the pattern is not certain up to this state with the same observations, this verifier state is called an ambiguous state. An ambiguous state cycle is a cycle containing only ambiguous states. As any event in V_G is observable, then a path in V_G containing an ambiguous state cycle corresponds to a global critical pair, which is called a **global critical path** in the following. So pattern diagnosability testing consists in checking the existence of global critical paths in V_G. Thus we have the following theorem.

Theorem 1: A pattern Ω is diagnosable in a system G iff there is no global critical path in the global pattern verifier.

III. Distributed Framework

As said before, the centralized approach is impractical due to its assumption of the monolithic model. In this section, we show how to recognize a given pattern by incrementally extending a subsystem to avoid the global model and then show how to analyze pattern diagnosability through global consistency checking of the diagnosability relative parts to avoid the global pattern verifier.

A. Pattern Recognition

Before showing how to recognize a given pattern, for the sake of simplicity, we first modify final states in the pattern as follows: $\forall\rho$, where ρ is a path in Ω, such that $\rho = q \xrightarrow{\sigma_1} q_1 ... \xrightarrow{\sigma_n} q_n \xrightarrow{\sigma\prime} q\prime$, where $\{q_1, ..., q_n, q\prime\} \subseteq F_\Omega$, $q \notin F_\Omega$, it is modified as a path $\rho\prime = q \xrightarrow{\sigma_1} q_1 \xrightarrow{\Sigma_\Omega} q_1$. This means that we replace the stable final states set F_Ω with its transitions by the stable final states set $F'_\Omega = \{(q, \Sigma_\Omega, q) \mid q \in F_\Omega \text{ and } \exists(q\prime, \sigma, q), q\prime \notin F_\Omega\}$. Since F_Ω is stable, this operation has no impact on the correctness of the diagnosability algorithm, except to make it more simpler.

Given a subsystem, following the same idea as in definition 6, we construct the local pattern recognizer as follows.

Definition 8: (Local Pattern Recognizer). Given a subsystem $G_S = (Q_S, \Sigma_S, \delta_S, q^0_S)$ and a pattern Ω, then the local pattern recognizer of G_S is $R_{G_S} = G_S \times \Omega$, where the initial state is (q^0_S, q^0_Ω) and the set $F_{R_{G_S}}$ of final states is $(Q_S \times F_\Omega) \cap Q_{R_{G_S}}$ ($Q_{R_{G_S}}$ is the set of states in R_{G_S}).

Similar to the global pattern recognizer, the local pattern recognizer shows which part of the pattern is recognized by any trajectory in the subsystem. If the pattern cannot be completely

recognized, we need to choose another component to extend the subsystem for further recognition. Before choosing the next component, we first define the following notations.

Definition 9: (Recognition Relative Path and Diagnosability Relative Path).

- Given a path ρ in the pattern recognizer R_{G_S} of the subsystem G_S, if ρ contains at least one state $q_r = (q, q_\Omega)$, such that either q_Ω is a final state of the pattern Ω or $\exists \sigma \in \widehat{\varpi_{q_\Omega}}$ such that $\sigma \in \Sigma \backslash \Sigma_S$ (where Σ_S is the set of events of G_S and Σ is that of the entire system G), then ρ is called a recognition relative path. And σ, if any, is called a next recognizable event with respect to the subsystem G_S. The set of next recognizable events with respect to G_S is denoted by Λ_{G_S}.
- Given a path ρ in R_{G_S}, if it is a recognition relative path or it has the same observations as some recognition relative path of R_{G_S}, then it is called a diagnosability relative path.

A recognition relative path contains either at least one final state of the recognizer or at least one state that is the source state of a significant event in the pattern such that this significant event is contained outside of the current subsystem, which is called a next recognizable event. Only such kind of paths can possibly recognize the rest of the pattern after synchronization with other components. Then the next component to be chosen should contain at least one next recognizable event. The set of recognition relative paths contain the projections on the subsystem of all the global trajectories that recognize the pattern (called the corresponding subparts of these trajectories in the subsystem). The local pattern recognizer R_{G_S} can then be reduced by only retaining its diagnosability relative paths. Let denote $R^{\Omega}_{G_S}$ this reduced recognizer.

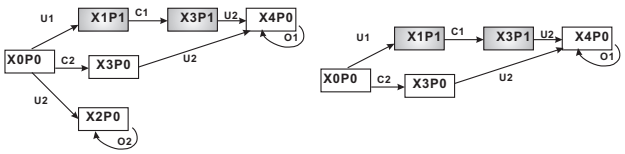

Fig. 2. The local pattern recognizer R_{G_S} for the initial subsystem, i.e., G_1 (left), and the reduced one $R^{\Omega}_{G_S}$ (right).

Figure 2 shows the local pattern recognizer R_{G_S}, where G_S is the component G_1 (left part) and the reduced recognizer $R^{\Omega}_{G_S}$ (right part), where the gray nodes represent the recognizer states whose pattern state is the source state of a next recognizable event in the pattern. Here we have only one next recognizable event $O3$ with respect to G_S. Thus we get the reduced part $R^{\Omega}_{G_S}$ by deleting one path which concerns neither pattern recognition nor pattern diagnosability. So it is easy to prove the following lemma since the set of diagnosability relative paths includes not only recognition relative paths but also all the paths with the same observations as some recognition relative path.

Lemma 1: The reduced pattern recognizer $R^{\Omega}_{G_S}$ contains the corresponding subpart in the subsystem G_S (projection on G_S) of all global critical pairs.

We define a complete recognizer as a local reduced pattern recognizer with at least one final state and without next recognizable event with respect to its corresponding subsystem. The existence of a complete recognizer implies that the pattern can be recognized in the current subsystem and there is no other component that should be further exploited for next recognition.

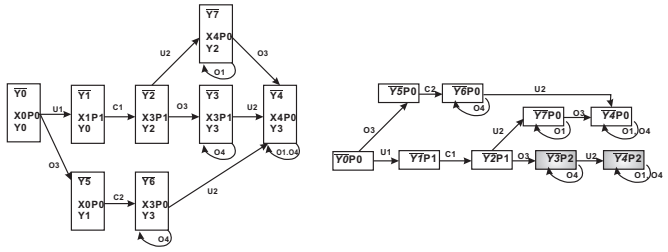

Fig. 3. Part of the extended subsystem $R^{\Omega}_{G_S} \| G_2$ (left) and part of its corresponding reduced pattern recognizer (right).

If a local reduced recognizer $R^{\Omega}_{G_S}$ is not a complete recognizer, then there could be three cases:

1) there is no final state in $R^{\Omega}_{G_S}$ and the set of next recognizable events is not empty $\Lambda_{G_S} \neq \emptyset$;
2) there exists at least one final state in $R^{\Omega}_{G_S}$ and the set of next recognizable events is not empty $\Lambda_{G_S} \neq \emptyset$;
3) there is no final state in $R^{\Omega}_{G_S}$ and the set of next recognizable events is empty $\Lambda_{G_S} = \emptyset$.

In case 1, the pattern is not yet recognized in the current subsystem and there exists at least one next recognizable event. And in case 2, at least one sequence of ordered significant events of the pattern is recognized in the current subsystem but there exists at least one next recognizable event for next recognition (considering there could be not only one such sequence in the pattern to be recognized). So in the first two cases, there exists at least one component G_j such that $\Sigma_j \cap \Lambda_{G_S} \neq \emptyset$ and thus we select G_j that contains at least one next recognizable event for next recognition. Then we extend G_S by G_j through constructing $R^{\Omega}_{G_S} \|_{\Sigma_c} G_j$, where synchronization is based on the shared communication events, and we extend accordingly the pattern recognizer for this extended subsystem by product with Ω and we reduce it. Note that the number of events in Λ_{G_S} is not necessarily only one, so G_j is not unique in general but the order of selection is not influential for pattern recognition. Case 3 means that the pattern is not recognized in the current subsystem and there is no next recognizable event. In other words, case 3 implies that the pattern cannot be recognized in the whole system.

In figure 2 (right), $R^{\Omega}_{G_S}$ is in case 1 with $\Lambda_{G_S} = \{O3\}$. As $O3 \in \Sigma_2$, G_2 is selected for extension. Figure 3 depicts this extension of G_S by G_2 through synchronization based on the shared communication events (left part) and the according reduced extended pattern recognizer (right part) which is actually a complete recognizer since it contains final states

and there is no next recognizable event. Here we use gray nodes to show final states of the recognizer.

B. Pattern Diagnosability Verification

Once a complete recognizer is calculated, then we construct the regional pattern verifier based on this complete recognizer. Since unobservable events do not intersect between components and there is no cycle of unobservable events, the information about unobservable events is not useful during global consistency checking. But, different from the global pattern verifier defined in definition 7, for regional version we need, to check global consistency, to retain not only observable events but also communication events. So first we refine the complete recognizer by the delay closure with respect to the set of communication events and observable events. The refined recognizer is denoted as R_r. We obtain left instance of R_r, denoted by R_r^l, by prefixing the communication events with L. Then we get the right instance of R_r, denoted by R_r^r, by prefixing the communication events with R. This is because we need to keep track of the origin (left or right instance) of the communication events for further synchronization, which are non synchronized events when constructing regional pattern verifier. The regional pattern verifier is thus constructed by synchronizing the left instance with the right instance based on all observable events in R_r. The idea is to obtain all pairs of trajectories with the same observations in the subsystem.

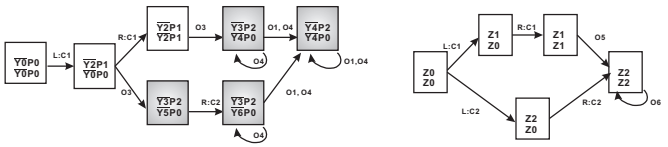

Fig. 4. Part of the regional pattern verifier for $\{G_1, G_2\}$ (left) and part of the local twin checker for G_3 (right).

Definition 10: (Regional Pattern Verifier). Given the refined recognizer R_r, the regional pattern verifier is $V = R_r^l \|_{\Sigma_{r_o}} R_r^r$, where Σ_{r_o} is the set of observable events in R_r.

In the regional pattern verifier, any path containing an ambiguous state is called a partial critical path. A part of the regional pattern verifier based on the complete recognizer partly depicted in the right part of figure 3 is shown in the left part of figure 4, where gray nodes are used for ambiguous states. Here we have partial critical paths since they contain ambiguous states.

Global consistency checking consists in verifying whether a partial critical path corresponds to a global critical path. Recall that a global critical path is a path in the global pattern verifier containing an ambiguous state cycle with at least one observable event. As this presence of an observable event is guaranteed from our assumption of observable liveness for each component, the information in a partial critical path important for global consistency checking is all the ambiguous states. And this consistency checking consists in verifying whether there exist ambiguous state cycles in the global path after synchronization process. Since synchronization between components is based on common communication events, then it is also necessary to retain communication information. In other words, the regional pattern verifier can be abstracted by only retaining information about all its ambiguous states as well as its communication events.

Definition 11: (Abstracted Pattern Verifier). A given regional pattern verifier V is abstracted in V^a, called abstracted pattern verifier, by the following steps, where $\{G_{s_1}, ...G_{s_m}\}$ are the components involved in V:

1) (Retaining only communication events) Delay closure with respect to the set of communication events is operated on the pattern verifier V, $V^a = \mathsf{C}_{\Sigma_c}(V)$.
2) (Adding qualitative description of the ambiguous state cycles without communication event, which have been lost in step 1) If there exists a path in V: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$, where q_0 is the initial state of V, $\exists j \in \{0, ..., n-1\}$, $q_j = q_n$, $\forall p \in \{j, ..., n\}$, q_p is an ambiguous state, $\forall k \in \{j+1, ..., n\}, e_k \in \Sigma_o$, i.e. all events in this ambiguous state cycle are observable events, then the corresponding path in V^a $p = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m$ is modified as $p\prime = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m \xrightarrow{obs} q^\Omega \xrightarrow{obs} q^\Omega$, where obs represents the existence of observable events of the subsystem corresponding to V and q^Ω represents a verifier state that is ambiguous with respect to the pattern Ω.
3) For each communication transition $(q_i \xrightarrow{e_1} q_j)$ in V^a such that only one of the two states is ambiguous, we check its corresponding part in V. If q_i is ambiguous and in this corresponding part of V, there is an ambiguous state after e_1, then this transition in V^a becomes $(q_i \xrightarrow{e_1} q^\Omega \xrightarrow{\sigma} q_j)$. And if q_j is ambiguous and in this corresponding part, there is an ambiguous state before e_1, then this transition in V^a becomes $(q_i \xrightarrow{\sigma} q^\Omega \xrightarrow{e_1} q_j)$, where σ represents an event of the system but which one is not important.

The abstracted pattern verifier retains the corresponding parts of all partial critical paths in the regional pattern verifier, which are also called partial critical paths in the following for the sake of simplicity. All ambiguous state cycles are kept in a qualitative way: those with both observable and communication events are kept with their only communication events in the first step of definition 11 while those with only observable events, which are lost in this first step, are recuperated by the second step. And the third step is to recuperate ambiguous states before or after communication events lost in the first step but not recovered in the second step. The left part of figure 5 shows a part of the abstracted pattern verifier obtained from the regional verifier depicted in figure 4, where ambiguous states are represented by gray nodes.

Each global critical path corresponds to a partial critical path in the abstracted pattern verifier but the inverse is not true. The reason is that up to now we did not take into account the communication of partial critical paths with their neighborhood in the whole system. In other words, a partial

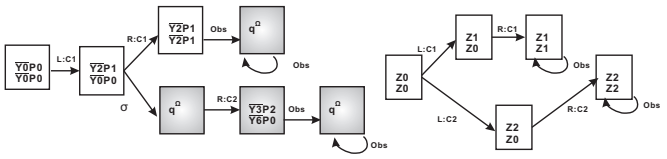

Fig. 5. Part of the abstracted pattern verifier for $\{G_1, G_2\}$ (left) and part of the ALTC for G_3 (right).

critical path is not necessarily extensible into a global critical path during synchronization. To check the global consistency, given a component, we define as follows a structure called local twin checker, which aims at getting all pairs of local trajectories with the same observations.

Definition 12: (Local Twin Checker). The local twin checker of G_i is $C_i = (\complement_{\Sigma_d}(G_i))^l \| _{\Sigma_{i_o}} (\complement_{\Sigma_d}(G_i))^r$, where $\Sigma_d = \Sigma_{i_o} \cup \Sigma_{i_c}$.

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 distinguishing non-synchronized communication events with the prefix L and R (left and right instances) before synchronizing the resulted local model with itself based on the observable events. The right part of figure 4 shows part of the local twin checker for the component G_3.

The local twin checker is used to check whether a partial critical path can be extended into a global critical path through synchronization with the abstracted pattern verifier. Considering that ambiguous state cycles can be blocked by communication events and that any ambiguous state can possibly be extended into an ambiguous state cycle after synchronizing with some cycle in the local twin checkers, then only communication events and all cycles need to be kept in the local twin checker. Thus the abstracted local twin checker is defined as follows.

Definition 13: (Abstracted Local Twin Checker-ALTC). A given local twin checker C_i is abstracted in C_i^a, called abstracted local twin checker (ALTC), by operating delay closure with respect to the set of communication events on C_i, $C_i^a = \complement_{\Sigma_{ic}}(C_i)$ and then by the following step: if there exists a local path in C_i: $q_0 \xrightarrow{e_1} q_1 ... \xrightarrow{e_n} q_n$, where q_0 is the initial state of C_i, $\exists j \in \{0, ..., n-1\}$, $q_j = q_n$, $\forall k \in \{j+1, ..., n\}$, e_k is an observable event, suppose that the corresponding local path in C_i^a is $p = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m$, then it is modified as $p\prime = q'_0 \xrightarrow{c_1} q'_1 ... \xrightarrow{c_m} q'_m \xrightarrow{obs} q'_m$.

Thus ALTC preserves all communication events as well as all cycles in a qualitative way. The idea is to check whether the partial critical paths can be extended into global critical paths after synchronization with all other ALTCs.

During the pattern recognition, by keeping only diagnosability relative paths during the subsystem extension, the search space has been already reduced. Now we will further save space by checking the global consistency of only partial critical paths through synchronization of the abstracted pattern verifier with the ALTCs to avoid building the global pattern verifier.

To check their global consistency, the partial critical paths are synchronized with the ALTCs of the connected components, i.e., those components which are not involved in the subsystem G_S corresponding to the abstracted pattern verifier but are neighboring with G_S (contain at least one shared communication event with G_S). Now we define the global consistency of a partial critical path as follows.

Definition 14: (Global Consistency). A partial critical path is globally consistent if after synchronization with the ALTCs of all connected components, it either contains an ambiguous state cycle or contains an ambiguous state and there exists at least one independent component, i.e., non connected component.

Algorithm 1 Global Consistency Checking

```
1:  INPUT: component models G_1, ..., G_n of the system G;
    the abstracted pattern verifier V^a; the current subsystem
    G_S, which is initially the subsystem corresponding to V^a
2:  V^a ← Reduce(V^a)
3:  while V^a ≠ ∅ and ConnectComp(G_S) ≠ ∅ do
4:      G_j ← Select(ConnectComp(G_S))
5:      C_j^a ← ConstructALTC(G_j)
6:      V^a ← V^a ‖ C_j^a
7:      V^a ← Reduce(V^a)
8:      G_S ← Add(G_S, G_j)
9:  if (there exist ambiguous cycles in V^a) or (V^a ≠ ∅ and
    G_S ≠ G) then
10:     return V^a
11: else
12:     return G_S
```

Algorithm 1 describes the procedure to check the global consistency, given the set of components of the system, abstracted pattern verifier V^a with its corresponding subsystem G_S. V^a is first reduced by only retaining all its partial critical paths (line 2). When the reduced V^a is not empty and there is at least one connected component to the subsystem G_S (line 3), i.e., the component shares at least one common communication event with G_S, then global consistency checking repeatedly performs the following steps.

1) Select one connected component G_j and construct its ALTC C_j^a (line 4-5).
2) Synchronize C_j^a with the reduced V^a, where the set of synchronized events is the set of common left and right communication events of G_S and G_j (line 6).
3) Reduce the newly obtained abstracted pattern verifier by only retaining all its partial critical paths before updating the current subsystem G_S by adding the component G_j (line 7-8).

If there is no other connected component, any path obtained in the final FSM that either contains an ambiguous state cycle or contains an ambiguous state with the existence of non connected components is globally consistent. If there is at least one such path, then this FSM is returned to provide the non-

diagnosability information (line 9-10). Otherwise, if there is no such path, then the current subsystem G_S is a Ω-diagnosable subsystem, which is returned by our algorithm (line 11-12).

Now we are ready to state and prove the following lemma.

Lemma 2: A partial critical path is globally consistent iff it corresponds to (i.e., is the projection of) a global critical path.

Proof:

($\Rightarrow$) Suppose that a partial critical path ρ is globally consistent. After synchronization with the ALTCs of all connected components, if it contains an ambiguous state cycle, then it is easy to deduce that, when ρ is synchronized with all ALTCs, it must contains an ambiguous state cycle. Otherwise, if it contains an ambiguous state and there exists at least one independent component, then after synchronizing with ALTCs of independent components, we get ambiguous cycles. From the assumption of observable liveness of each component, there must exist in this cycle at least one observable event. This means that ρ can be extended into a global critical path. ($\Leftarrow$) Suppose that a partial critical path ρ corresponds to a global critical path $\rho\prime$, i.e., the projection of $\rho\prime$ on the abstracted pattern verifier is ρ. Since global critical path contains an ambiguous state cycle and ρ contains at least one ambiguous state, then it follows that after synchronization with the ALTCs of all connected components, it either contains an ambiguous state cycle or contains an ambiguous state with at least one independent component. which means that ρ is globally consistent. ■

From lemma 2 and theorem 1, we can directly obtain the following theorem to verify pattern diagnosability in a distributed way.

Theorem 2: A pattern Ω is diagnosable in a system G iff there is no partial critical path that is globally consistent.

For our example, after global consistency checking of the partial critical paths of the abstracted pattern verifier for $\{G_1, G_2\}$ (see figure 5, left), i.e., after synchronizing these paths with the ALTC for G_3 (see figure 5, right), at least one partial critical path does not disappear and contains an ambiguous state cycle. In other words, there does exist a globally consistent partial critical path. Thus this system is not Ω-diagnosable.

C. Distributed Algorithm

The algorithm 2 describes the procedure of the distributed diagnosability verification for a given pattern. With the set of component models and the pattern under consideration as input, the parameters of the algorithm are initialized. As long as there exists at least one next recognizable event with respect to the current subsystem (line 3), which means that there are other components that should be further exploited for next pattern recognition, the following steps are repeatedly performed.

1) The current pattern recognizer is reduced by only retaining its diagnosability relative paths (for the first time doing nothing since the current recognizer is empty) and then one component containing at least one next recognizable event is selected (line 4-5).

Algorithm 2 Pattern Diagnosability Algorithm for Distributed System

1: INPUT: component models $G_1, ..., G_n$ of the system G: $G = \{G_1, ...G_n\}$; the pattern Ω to be diagnosed in G
2: Initializations: $G_S \leftarrow \emptyset$ (the current subsystem, initially empty); $R \leftarrow \emptyset$ (the current pattern recognizer, initially empty); $\Lambda_{G_S} \leftarrow \widehat{\varpi_{q_\Omega^0}}$ (the set of next recognizable events with respect to the current subsystem, initially the set of significant events of Ω that change its initial state q_Ω^0);
3: **while** $\Lambda_{G_S} \neq \emptyset$ **do**
4: $R \leftarrow REDUCE(R)$
5: $G_i \leftarrow SelectComp(\Lambda_{G_S}, G)$
6: $G_i \leftarrow G_i \| R$, where the synchronized events are the common communication events of G_S and G_i
7: $R \leftarrow ConstructLPR(G_i, \Omega)$
8: $G_S \leftarrow Add(G_S, G_i)$
9: $\Lambda_{G_S} \leftarrow CollectNRE(R, G_S, G, \Omega)$
10: **if** R is not a complete recognizer **then**
11: return "Ω cannot be recognized in G"
12: **else**
13: $R \leftarrow Refine(R)$
14: $V^a \leftarrow ConstructAPV(R)$
15: $CheckGlobalConsistency(G, V^a, G_S)$

2) The reduced recognizer is then synchronized with the selected component, based on the set of common communication events of the current subsystem and the selected component (for the first time doing nothing since the current subsystem is empty), and then the local pattern recognizer for this synchronized FSM is again constructed (line 6-7).
3) The current subsystem is now updated by adding the selected component, and then the set of next recognizable events with respect to this current subsystem is updated as described in definition 9 (line 8-9).

When there is no more next recognizable event and the current pattern recognizer is not a complete one, which means that there is no final state in this recognizer, it can be deduced that the pattern can never be recognized in the system. In this case, our algorithm returns the information about non recognizability of the pattern (line 10-11). Otherwise (line 12), i.e., the current pattern recognizer is a complete one, based on which we construct the abstracted pattern verifier (line 13-14) before checking its global consistency (line 15) whose procedure described by algorithm 1.

IV. Implementation

To experimentally illustrate the correctness of our proposed distributed algorithm and demonstrate its efficiency, we have compared it to the centralized algorithm of [7] after having implemented both. Our results emphasize that the search space of our distributed algorithm is much smaller than that of the centralized one in most cases. Considering that if the number of faults is high we will face a significant increase in

complexity, it is better to check the diagnosability individually for each fault, as it is usually done.

The test case that we adopt is a simple example of an office system composed of several components (please see more details about the example in [8]). Then the figure 6 shows the growth of states number and of transitions number in the global pattern recognizer, the global pattern verifier, the distributed complete pattern recognizer and the distributed pattern verifier when the system is extended by adding more independent components. We can see that, since the added components are independent, then for our distributed approach, the search spaces of the complete pattern recognizer (distributed PR) and of the distributed pattern verifier (distributed PV) never increase, while for the centralized approach, the search spaces of the global pattern recognizer (global PR) and of the global pattern verifier (global PV) dramatically increase. This is obviously the case the most favorable to distributed approaches and further experiments to evaluate the gain for intermediate scenarios, depending in particular of the degree of connectedness of the components, are planned.

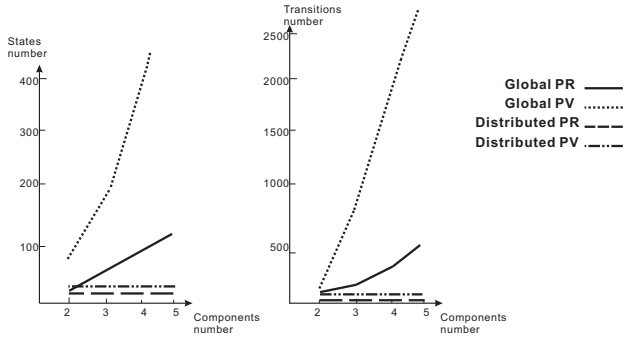

Fig. 6. The search space growth when adding simple independent components.

Furthermore, we also compared two distributed approaches for several small systems where all components are connected: normal distributed approach without abstraction and distributed approach with abstraction as described in this paper. The idea was to check how much space can be saved with the abstraction. And the results show that for all these examples (which will have to be completed by larger ones), we can save space between thirty and sixty percentage.

V. Conclusion

In this paper we proposed a distributed approach for pattern diagnosability. We first showed how to incrementally recognize the pattern by synchronizing the diagnosability relative paths with other components to avoid building a global model. Then, for pattern diagnosability checking, we abstracted just the necessary and sufficient information from local objects to further save the search space. The approach described in [10] is close to this work, whose subject is also about how to verifier pattern diagnosability of distributed discrete event systems. However, their algorithm is limited to deal with only what called simple pattern, i.e., one sequence of events. Moreover, in [10], different from the approach shown in this paper, there is no abstraction technique to further save search space. Our approach in this paper is general enough to be applicable to any pattern, i.e., both to only one sequence of events or several, even infinite, sequences of events. The correctness and efficiency of our algorithm have been not only theoretically but also practically proved through implementation. To the sake of comparison, we also implemented the centralized algorithm. Our results emphasize that our proposed distributed approach leads to the same conclusions as the centralized one but with a much smaller search space. To the best of our knowledge, these are the first experimental results for pattern diagnosability, even in the centralized case. We have several perspectives for this work. For example, when a diagnosable subsystem is returned by our approach, it is possible to investigate whether the observations (i.e., observable events) in this subsystem can be reduced while keeping it diagnosable and, if yes, how to reduce them ([11]). Our approach can also be extended to analyze the predictability of distributed systems that concerns the ability of these systems to predict a fault before its occurrence, and thus trigger actions to avoid it ([12]).

References

[1] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis, "Diagnosability of discrete event system," in *IEEE Transactions on Automatic Control*, 1995, pp. 40(9):1555–1575.

[2] S. Jiang, Z. Huang, V. Chandra, and R. Kumar, "A polynomial time algorithm for diagnosability of discrete event systems," in *IEEE Transactions on Automatic Control*, 2001, pp. 46(8):1318–1321.

[3] A. Cimatti, C. Pecheur, and R. Cavada, "Formal verification of diagnosability via symbolic model checking," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, Acapulco, Mexico, 2003, pp. 363–369.

[4] Y. Pencolé, "Diagnosability analysis of distributed discrete event systems," in *Proceedings of European Conference on Articifial Intelligent ECAI'04*, 2004, pp. 43–47.

[5] A. Schumann and Y. Pencolé, "Scalable diagnosability checking of event-driven systems," in *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, Hyderabad, India, 2007, pp. 575–580.

[6] A. Schumann and J. Huang, "A scalable jointree algorithm for diagnosability," in *23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, Chicago, USA, July 2008.

[7] T. Jéron, H. Marchand, S. Pinchinat, and M. Cordier, "Supervision patterns in discrete event systems diagnosis," in *Proceedings of the 8th International Workshop on Discrete Event Systems*, New York, July 2006.

[8] L. Ye, *Optimized Diagnosability of Distributed Discrete Event Systems Through Abstraction*. Thesis, Université Paris-Sud, 2011.

[9] C. G. Cassandras and S. Lafortune, *Introduction To Discrete Event Systems Second Edition*. Springer, 2008.

[10] L. Ye, P. Dague, and Y. Yan, "An incremental approach for pattern diagnosability in distributed discrete event systems," in *21st International Conference on Tools with Artificial Intelligence (ICTAI-09)*, Newark, NJ, USA, November 2009, pp. 123–130.

[11] L. B. Briones, A. Lazovik, and P. Dague, "Optimizing the system observability level for diagnosability," in *3rd International Symposium on Leveraging Applications of Formal Methods, Verification and Validation*, October 2008, pp. 815–830.

[12] T. Jéron, H. Marchand, S.Genc, and S. Laforture, "Predictability of sequence patterns in discrete event systems," in *Proceedings of the 17th World Congress*, Seoul, Korea, July 2008, pp. 537–543.