A Scalable Jointree Algorithm for Diagnosability*

Anika Schumann
Advanced Computing Research Centre
University of South Australia
Mawson Lakes, SA 5095, Australia
anika.schumann@cs.unisa.edu.au

Jinbo Huang
National ICT Australia and
The Australian National University
Canberra, ACT 0200, Australia
jinbo.huang@nicta.com.au

Abstract

Diagnosability is an essential property that determines how accurate any diagnostic reasoning can be on a system given any sequence of observations. An unobservable fault event in a discrete-event system is diagnosable iff its occurrence can always be deduced once sufficiently many subsequent observable events have occurred. A classical approach to diagnosability checking constructs a finite state machine known as a *twin plant* for the system, which has a *critical path* iff some fault event is not diagnosable. Recent work attempts to avoid the often impractical construction of the global twin plant by exploiting system structure. Specifically, local twin plants are constructed for components of the system, and synchronized with each other until diagnosability is decided. Unfortunately, synchronization of twin plants can remain a bottleneck for large systems; in the worst case, in particular, all local twin plants would be synchronized, again producing the global twin plant. We solve the diagnosability problem in a way that exploits the distributed nature of realistic systems. In our algorithm consistency among twin plants is achieved by message passing on a *jointree*. Scalability is significantly improved as the messages computed are generally much smaller than the synchronized product of the twin plants involved. Moreover we use an iterative procedure to search for a subset of the jointree that is sufficient to decide diagnosability. Finally, our algorithm is scalable in practice: it provides an approximate and useful solution if the computational resources are not sufficient.

Introduction

Automated fault diagnosis has significant practical impact by improving reliability and facilitating maintenance of systems. Given a monitor continuously receiving observations from a dynamic event-driven system, diagnostic algorithms detect possible fault events that explain the observations. For many applications, it is not sufficient to identify what faults *could* have occurred; rather one wishes to know what faults have *definitely* occurred. Computing the latter in general requires *diagnosability* of the system, that is, the guarantee that the occurrence of a fault can be detected with certainty after a finite number of subsequent observations (Sampath et al. 1995). Consequently, diagnosability analysis of the system should be performed before any diagnostic reasoning. The diagnosability results then help in choosing the type of diagnostic algorithm that can be performed and provide some information of how to change the system to make it more diagnosable.

In this paper, we propose a formal framework for checking diagnosability on event-driven systems which is mainly motivated by two facts. On the one hand, checking diagnosability means determining the existence of two behaviors in the system that are not *distinguishable*. However, in realistic systems, there is a combinatorial explosion of the search space that forbids the practical use of classical and centralized diagnosability checking methods (Sampath et al. 1995) like the twin plant method (Jiang et al. 2001; Yoo and Lafortune 2002).

Our proposal makes several contributions to the diagnosability problem. The first one is the definition of a new theoretical framework where the classical diagnosability problem is described as a distributed search problem. Instead of searching for indistinguishable behaviors in a global twin plant, we propose to distribute the search based on local twin plants, represented as finite state machines (FSMs). Specifically, we exploit the modularity of the system by organizing the system components into a special tree structure, known as a *jointree*, where each node of the tree is assigned a subset of the local twin plants, whose collective set of events has a size bounded by the *treewidth* of the system. Once the jointree is constructed we need only synchronize the twin plants in each jointree node, and all further computation takes the form of message passing along the edges of the jointree. Using the jointree properties we show that after two messages per edge, the FSMs at all nodes are collectively consistent. This allows us to decide diagnosability by considering these FSMs in sequence instead of the large global twin plant.

We describe how messages, which are themselves FSMs, are computed, based on projecting a FSM onto a subset of its events, and how diagnosability information can be propagated along with the messages. Then we employ a systematic iterative procedure so that only a subset of the jointree is considered at a time and the loop terminates as soon as the current subset is sufficient for deciding diagnosability.

Since diagnosability analysis is a complex problem, we also consider the usefulness of our algorithm where it cannot run to completion due to lack of computational resources. The distributed nature of our search ensures that in such cases it is able to provide an approximate solution to the

*This work was supported by NICTA's SuperCom project. The first author is currently funded by ARC grant DP0560183.
Copyright © 2008, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

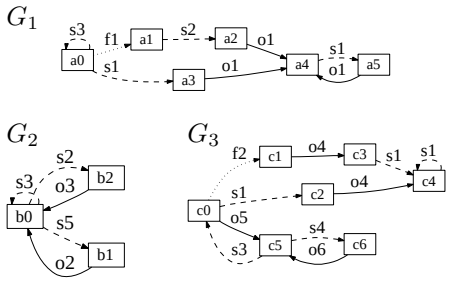

Figure 1: Three components of a system modeled as FSMs. Solid, dashed, and dotted lines denote observable, shared, and failure transitions, respectively.

diagnosability problem. Specifically, it returns a maximal subsystem for which the existence of indistinguishable behaviors has been decided, but not yet verified against the rest of the system due to the limited computational resources. In case indistinguishable behaviors have been found, then the larger this subsystem is, naturally, the more likely the whole system is not diagnosable; otherwise the reverse is true. Such an approximate solution is also useful in that it tells the user that on-line monitoring of this particular subsystem will not be sufficient to detect occurrences of the fault.

Background

In this section we review the definition of diagnosability and the twin plant approach to diagnosability checking, and give a short introduction to jointrees.

Diagnosability of discrete-event systems

As in (Sampath et al. 1995), we consider a discrete-event system G consisting of components $G_1, \ldots, G_n$. Each component is a FSM $G_i = \langle X_i, \Sigma_i, x_{0_i}, T_i \rangle$ where X_i is the set of states, Σ_i is the set of events, x_{0_i} is the initial state, and T_i is the transition relation ($T_i \subseteq X_i \times \Sigma_i \times X_i$). The set of events Σ_i is divided into four disjoint subsets: observable events Σ_{o_i}, unobservable events Σ_{s_i} shared with other components, unobservable fault events Σ_{f_i}, and other unobservable events Σ_{u_i}.

Figure 1 depicts three components of a system modeled as FSMs. Note that a monolithic model for the entire system is implicitly defined as the synchronized product, $G = Sync(G_1, \ldots, G_n)$, of all component models. The result of $Sync$ is a FSM whose state space is the Cartesian product of the state spaces of the components, and whose transitions are synchronized in that any shared event always occurs simultaneously in all components that define it.

A fault $F \in \bigcup_i \Sigma_{f_i}$ of the system is *diagnosable* iff its (unobservable) occurrence can always be deduced after finite delay (Sampath et al. 1995). In other words, a fault is not diagnosable if there exist two infinite paths from the initial state which contain the same infinite sequence of observable events but exactly one of which contains the fault.

More formally, let p_F denote a path starting from the initial state of the system and ending with the occurrence of a fault F in a state x_F, let s_F denote a finite path starting from x_F, and let $obs(p)$ denote the sequence of observable events in a path p. As in (Sampath et al. 1995), we assume that (i) the system is *live* (there is a transition from every state), and (ii) the observable behavior of the system is *live* ($obs(p)$ is infinite for any infinite path p of the system). We have:

Definition 1 (Diagnosability) *F is diagnosable iff*

$$\exists d \in \mathbb{N}, \forall p_F s_F, |obs(s_F)| > d \Rightarrow$$
$$(\forall p, obs(p) = obs(p_F s_F) \Rightarrow F \textit{ occurs in } p).$$

Diagnosability checking thus requires the search for two infinite paths p and p', i.e. paths containing a cycle, with $obs(p) = obs(p')$ such that F is in p but not in p'. The pair (p, p') is called a *critical pair* (Cimatti, Pecheur, & Cavada 2003). From here on we will write *path* to mean a path that starts from the initial state of the system.

Twin plant for diagnosability checking

The idea of the twin plant is to build a FSM that compares every pair of paths (p, p') in the system that are equivalent to the observer ($obs(p) = obs(p')$), and apply Definition 1 to determine diagnosability (Jiang et al. 2001).

We now present the twin plant method based on the component models. From them we compute the *interactive diagnoser* (Pencolé 2005), which gives the set of faults that can possibly have occurred for each sequence of observable and shared events. The interactive diagnoser of a component G_i is the nondeterministic finite state machine $\widetilde{G}_i = \langle \widetilde{X}_i, \widetilde{\Sigma}_i, \widetilde{x}_{0_i}, \widetilde{T}_i \rangle$ where $\widetilde{X}_i$ is the set of states ($\widetilde{X}_i \subseteq X_i \times \mathcal{F}$ with $\mathcal{F} \subseteq 2^{\Sigma_{f_i}}$), $\widetilde{\Sigma}_i$ is the set of events ($\widetilde{\Sigma}_i = \Sigma_{o_i} \cup \Sigma_{s_i}$), $\widetilde{x}_{0_i} = (x_{0_i}, \emptyset)$ is the initial state, and $\widetilde{T}_i \subseteq \widetilde{X}_i \times \widetilde{\Sigma}_i \times \widetilde{X}_i$ is the transition set $(x, \mathcal{F}) \xrightarrow{\sigma} (x', \mathcal{F}')$ such that there exists a transition sequence $x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_m} x_m \xrightarrow{\sigma} x'$ in G_i with $\Sigma'_i = \{\sigma_1, \ldots, \sigma_m\} \subseteq \Sigma_{f_i} \cup \Sigma_{u_i}$ and $\mathcal{F}' = \mathcal{F} \cup (\Sigma'_i \cap \Sigma_{f_i})$.

Figure 2 (top) depicts the interactive diagnoser for component G_1 in Figure 1. Following the transitions $s2, o1$ from the initial state of the diagnoser, for example, we arrive at state $(a4, \{f1\})$, meaning that the system contains a path to state $a4$ on which the sequence of observable and shared events is exactly $s2, o1$ and the set of faults is exactly $\{f1\}$.

The *local twin plant* is then constructed by synchronizing two instances $\widetilde{G}_i^l$ (left) and $\widetilde{G}_i^r$ (right) of the same interactive diagnoser based on the observable events $\Sigma_{o_i} = \Sigma_{o_i}^l = \Sigma_{o_i}^r$. Since only observable behaviors are compared, the shared events must be distinguished between the two instances: in $\widetilde{G}_i^l$ (resp. $\widetilde{G}_i^r$), any shared event $\sigma \in \Sigma_{s_i}$ from $\widetilde{G}_i$ is renamed l:$\sigma \in \Sigma_{s_i}^l$ (resp. r:$\sigma \in \Sigma_{s_i}^r$). This gives us the local twin plant $\hat{G}_i = Sync(\widetilde{G}_i^l, \widetilde{G}_i^r)$.

Figure 2 (bottom) depicts part of the twin plant for component G_1 in Figure 1. The top labels $x0, \ldots, x6$ of the states are their identifiers to which we will refer in subsequent figures. State labels are composed of a state in the left interactive diagnoser (middle label) and one in the right interactive diagnoser (bottom label). Each state of the twin plant is a pair $\hat{x} = ((x^l, \mathcal{F}^l), (x^r, \mathcal{F}^r))$ that represents two possible diagnoses given the same sequence of observable events. If some fault F belongs to $\mathcal{F}^l \cup \mathcal{F}^r$ but not to $\mathcal{F}^l \cap \mathcal{F}^r$, then the occurrence of F cannot be deduced in this state. In this case, the state $\hat{x}$ is called *F-nondiagnosable*;

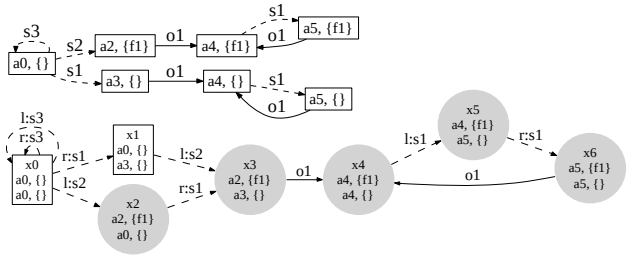

Figure 2: Diagnoser (top) and part of a twin plant (bottom).

otherwise it is called *F-diagnosable*. In Figure 2 the oval nodes represent f1-nondiagnosable states. By extension, a state $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_k)$ is *F-nondiagnosable* iff it is composed of one *F-nondiagnosable* state.

A fault F is diagnosable in system G iff, its *global twin plant* (GTP for short) $Sync(\hat{G}_1, \ldots, \hat{G}_n)$ has no path p with a cycle containing at least one observable event and one F-nondiagnosable state (Schumann and Pencolé 2007). Such a path p represents a *critical pair* (p_1, p_2), and is called a *critical path*. The oval nodes in Figure 2, for example, form part of a critical path.

The twin plant method searches for such a path in the GTP. In this paper, we propose a new algorithm that avoids building the global twin plant, which is impractical for systems with a large number of states. Instead, local twin plants are built for components G_i of the system. Since the existence of a critical path in a local twin plant does not imply nondiagnosability of the global system, we need to propagate information between local twin plants, which we accomplish by message passing on a *jointree*.

Jointrees

Jointrees have been a classical tool in probabilistic reasoning and constraint processing (Shenoy and Shafer 1986; Dechter 2003), and correspond to *tree decompositions* known in graph theory (Robertson and Seymour 1986). For our purposes, a jointree is a tree whose nodes are labeled with sets of events satisfying two special properties:

Definition 2 (Jointree) *Given a set of FSMs $G_1, \ldots, G_n$ defined over events $\Sigma_1, \ldots, \Sigma_n$ respectively, a jointree is a tree where each node is labeled with a subset of $\Sigma = \bigcup_i \Sigma_i$ such that*

- *every Σ_i is contained in at least one node, and*
- *if an event is in two distinct nodes, then it is in every node on the path between them.*

Figure 3 (left) depicts a jointree for the three local twin plants for the system in Figure 1. The label on each edge represents the intersection of the two neighboring nodes, known as the *separator*.

The size of the largest label on a node, minus 1, is known as the width of the jointree. Although finding a jointree of minimal width for a given graph is known to be NP-hard, in practice polynomial-time heuristics, such as min-fill, can produce good results (Dechter 2003).

Once a jointree is constructed, each FSM G_i is assigned to a node that contains its events Σ_i. Figure 3 (right) depicts such an assignment. Note that in general each node can have multiple FSMs assigned to it. Now we need only synchronize the FSMs in each node, and the properties of the jointree then guarantee that consistency among all the FSMs can be achieved by passing exactly two messages over each edge of the jointree, one in each direction.

A Jointree Algorithm for Diagnosability

The synchronization of all twin plants on a jointree would solve the diagnosability problem. However, for large systems this can easily be impractical. Therefore we will only synchronize the twin plants in each jointree node and all further computation takes the form of messages passing between nodes, in such a way that diagnosability can be decided in the end.

Jointrees admit a generic message passing method that achieves consistency among the nodes (Dechter 2003). In our case this translates into a method that achieves consistency of all FSMs labeling the jointree nodes. The messages passed on will themselves be FSMs. In this section we describe how these messages can be computed and passed and how the diagnosability information can be propagated correctly as part of the messages. This will then also enable us to present an iterative diagnosability algorithm.

Computing messages

Messages are exchanged to achieve consistency. A FSM G_i with events Σ_i is *globally consistent* with respect to FSMs $G_1, \ldots, G_n$ iff for every path p_i in it, there exists a path p in the synchronized product $Sync(G_1, \ldots, G_n)$ of all FSMs that has with respect to Σ_i the same event sequence as p_i. A FSM G_i^c is *complete* iff it contains all globally consistent paths of G_i.

We now describe how we can compute the messages that need to be passed over the jointree edges in order to obtain a complete and globally consistent FSM in every jointree node. Each edge of a jointree can be regarded as a two-way partition of the nodes of the tree, and a message sent over an edge will represent a summary of the collective behavior permitted by one side of the partition. A major advantage of this method is that this summary needs only to mention events given by the separator labeling the edge (the jointree properties ensure that this equals the intersection of the two sets of events across the partition).

Hence a message can be computed by *projecting* a FSM onto a subset of its events. The *projection* $\Pi_{\Sigma'}(G) = \langle X', \Sigma', x_0, T' \rangle$ of a FSM G on events $\Sigma' \subseteq \Sigma$ is obtained

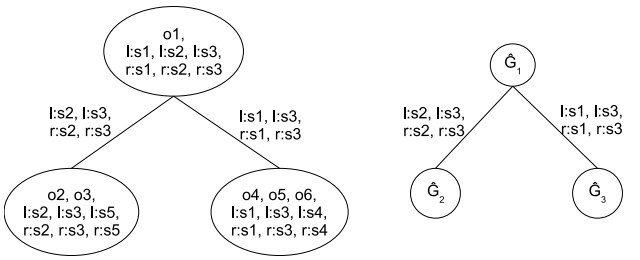

Figure 3: Jointree (left) and assignment of local twin plants to jointree nodes (right).

from G by first contracting all transitions not labeled by an event in Σ' and then removing all states (except the initial state x_0) that are not a target of any transition in the new set of transitions T'. More formally, T' is given as follows:

$$T' = \Big\{ x \xrightarrow{\sigma'} x' \mid x, x' \in X' \text{ and } \sigma' \in \Sigma' \text{ and } \exists\, x \xrightarrow{\sigma_1} x_1 \cdots \xrightarrow{\sigma_k} x_k \xrightarrow{\sigma'} x' \text{ in } G \text{ such that } \sigma_i \notin \Sigma' \ \forall i = 1, \ldots, k \Big\}.$$

Figure 4 shows the result of projecting the twin plant for G_1 on {l:s2,r:s2,l:s3,r:s3}.

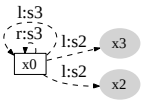

Figure 4: Projection $\Pi_{\{\text{l:s2,r:s2,l:s3,r:s3}\}}(\hat{G}_1)$.

Message passing

We now describe the message passing assuming that the FSMs in every node have been synchronized into a single FSM. To achieve consistency among the FSMs, each node of the jointree will in principle require a summary of the behavior permitted by FSMs residing in the rest of the tree. Given the jointree properties, all these summaries can be computed in only two passes over the jointree, one inward pass, in which the root "pulls" messages toward it from the rest of the tree and one outward pass, in which the root "pushes" messages away from it toward the leaves. Once all these messages have been sent, every FSM is updated based on all the messages it receives resulting in a complete and globally consistent FSM.

The process starts by designating any node of the tree as root. Then, in the first, inward pass, beginning with the leaves each node sends a message to its (unique) neighbor in the direction of the root. To compute this message, its FSM is synchronized with all messages it receives from its other neighbors (leaves do not have "other neighbors" and hence skip this step). The message it sends is then the projection of this FSM onto the separator between itself and the receiver of the message.

In the second, outward pass, each node (except the root) receives a message from its (unique) neighbor in the direction of the root. Again this message is computed by synchronizing its FSM with all messages it received from its other neighbors and by projecting the resulting FSM onto the separator between itself and the receiver of the message.

Finally, each node updates its own FSM by synchronizing it with messages from all its neighbors. Then every FSM G_i^c of a jointree node represents exactly the behavior that is complete and globally possible. Figure 5 illustrates the inward and outward propagation steps performed on the jointree of Figure 3 (right), resulting in the FSMs $\hat{G}_1^c$, $\hat{G}_2^c$ and $\hat{G}_3^c$. These propagations give us the following theorem:[1]

Theorem 1 *Every FSM G_i^c labeling a jointree node is complete and consistent with respect to all other FSMs $G_1, \ldots, G_n$ of the tree once it is synchronized with all messages it received.*

[1] Proofs of theorems in this paper are in (Schumann 2008).

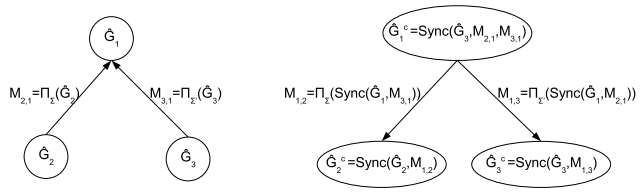

Figure 5: Inward (left) and outward (right) message propagation using jointrees, where $\Sigma = \{l{:}s2, r{:}s2, l{:}s3, r{:}s3\}$ and $\Sigma' = \{l{:}s1, r{:}s1, l{:}s3, r{:}s3\}$.

In particular this means that for every path p in $\hat{G}_i' = \Pi_{\Sigma_i}(\hat{G}_1, \ldots, \hat{G}_n)$ there is also an equivalent path p_i in $\hat{G}_i^c$, i.e., p_i is defined over the same event sequence as p, and vice versa. Now, for deciding diagnosability this simple equivalence is not sufficient. In addition we need to ensure that for every *critical* path p in G_i' there is also an equivalent *critical* path p_i in G_i^c. This requires the propagation of diagnosability information.

Propagation of diagnosability information

In the rest of the section we will assume that (i) the twin plants for components have been assigned to appropriate jointree nodes and synchronized within each node, (ii) G_F is the component defining the fault F whose diagnosability is to be checked, and (iii) the node containing the twin plant $\hat{G}_F$ is chosen as root.

Now, if any twin plant of a jointree node contains a consistent critical path, then the fault F is nondiagnosable. The root can already be searched for critical paths after the inward propagation, for two reasons: Firstly, the synchronization of the root with all its incoming messages results in a globally consistent twin plant, and secondly, since the fault F appears in the root the latter already contains diagnosability information, that is, the classification of states into diagnosable and nondiagnosable ones.

After the propagations are completed and every twin plant $\hat{G}_i$ is synchronized with all its incoming messages, every state of a twin plant is composed of a tuple $(\hat{x}_1, \ldots, \hat{x}_n)$, i.e., it is also composed of a state in $\hat{G}_F$. This state is "received" by $\hat{G}_i$ through its synchronization with the outward message, since the root is an ancestor of all nodes. When computing the outward messages we therefore need to ensure that we do not "lose" a path to a nondiagnosable state.

Recall that the projection operation, applied when computing the outward message, removes all states that are no longer a target state of a transition labeled by a separator event in Σ. This can lead to the removal of nondiagnosable states resulting in the incomplete propagation of diagnosability information. Consider for instance the twin plant $\hat{G}_u$ shown in Figure 6 (left). When computing the message $\mathcal{P}_u$ we remove the nondiagnosable state $u1$. This results in the consistent twin plant $\hat{G}_v^c$ which does not contain any critical paths although it should contain one as $\hat{G}_v^{c'}$ indicates.

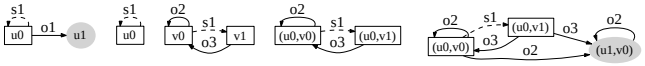

Figure 6: Twin plants $\hat{G}_u$, $\mathcal{P}_u = \Pi_{\{s1\}}(\hat{G}_u)$, $\hat{G}_v$, $\hat{G}_v^c = Sync(\Pi_{\{s1\}}(\hat{G}_u), \hat{G}_v)$, and $\hat{G}_v^{c'} = \Pi_{\Sigma_v}(Sync(\hat{G}_u, \hat{G}_v))$ (from left to right).

We therefore need to ensure that every message passed on from $\hat{G}$ to $\hat{G}'$ via the separator events Σ_{sep} will lead to a consistent twin plant $\hat{G}'^c$ that has a critical path iff $\Pi_{\Sigma_{sep}}(Sync(\hat{G}, \hat{G}'))$ has one. To achieve this we need to memorize for every diagnosable state $\hat{x}$ in $\mathcal{P}$ whether it has a *nondiagnosable local future*, that is, whether there is a transition sequence τ starting in $\hat{x}$ and leading to the non-diagnosable state $\hat{x}_k$ such that none of the transition events is kept in the projection. Formally this is the case iff the following condition is satisfied:

> There exists a transition sequence $\tau = \hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k$ in $\hat{G}$ such that $\hat{x}_k$ is nondiagnosable and none of the events $\sigma_1, \ldots, \sigma_k$ is in Σ_{sep}.

We do the memorization by adding for every diagnosable state $\hat{x}$ satisfying above condition a nondiagnosable *extended* terminal state $ext(\hat{x})$ and a terminal transition $\hat{x} \xrightarrow{ext} ext(\hat{x})$. The resulting FSM is the message which is passed.

Figure 7 illustrates the message $M_{1,3}$ sent from $\hat{G}_1$ (see Figure 2) to $\hat{G}_3$. Here the projection $\mathcal{P} = \Pi_{\{l:s1,r:s1,l:s3,r:s3\}}(\hat{G}_1)$ has the two diagnosable states $x0$ and $x1$, which both satisfy the above condition (see paths $x0 \xrightarrow{l:s2} x2$ and $x1 \xrightarrow{l:s2} x3$ in $\hat{G}_1$ in Figure 2). Thus $M_{1,3}$ contains two terminal transitions. On the other hand the message sent from $\hat{G}_1$ to $\hat{G}_2$ is the same as the projecting $\Pi_{\{l:s2,r:s2,l:s3,r:s3\}}(\hat{G}_1)$ shown in Figure 4 since there does not exist such a path in $\hat{G}_1$ for the only diagnosable state $x0$.

We are now ready to state the following major result:

Theorem 2 *Fault F is diagnosable in G iff after both passes of jointree propagation with diagnosability information, no FSM in a jointree node has a critical path.*

An iterative jointree algorithm

Rather than propagating messages over the entire jointree, we now describe how we can improve efficiency and scalability by searching for a subset of it that is sufficient to decide diagnosability.

The idea is that any critical path p in the global twin plant can be detected by looking only at those twin plants that define events appearing on p, since every other twin plant has

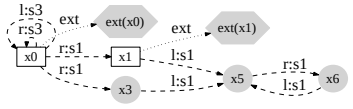

Figure 7: Message $M_{1,3}$. Grey states are nondiagnosable and hexagon shaped ones are extended states.

no impact on the behavior represented by p. Our aim is it to find a critical path defined over as few events as possible. The search for such a path will be done by iteratively increasing the set of jointree nodes (twin plants) $\hat{\mathbb{G}}$ under consideration, and looking for a critical path defined over only events Σ_{int} that are *internal* to $\hat{\mathbb{G}}$ (i.e., events that do not appear in the rest of the jointree). The detection of such a path establishes nondiagnosability and terminates the search.

Algorithm 1 CheckDiagnosability(jointree: J)

```
1:  Ĝ ← ∅                                          nodes in J being considered
2:  Σ_int ← ∅                                      events internal to Ĝ
3:  while Ĝ ≠ J and HasNondiagState(root) and
    SufficientMemory(Ĝ) do
4:      v ← PickNode(J, Ĝ)
5:      UpdateSets(v, Ĝ, Σ_int)
6:      Propagate(Ĝ)
7:      Ĝ_{Σ_int} ← GetAllPathsOverΣ_int(Ĝ)
8:      Propagate(Ĝ_{Σ_int})
9:      if ExistsTwinPlantWithCritPath(Ĝ_{Σ_int}) then
10:         return GetCritPath(Ĝ_{Σint})
11: if SufficientMemory(Ĝ) then
12:     return "F is diagnosable"
13: else
14:     ω ← set of components included in Ĝ
15:     if ExistsTwinPlantWithCritPath(Ĝ) then
16:         return "ω has a critical path"
17:     else
18:         return "ω has no critical path"
```

Algorithm 1 gives pseudo-code for this procedure. Our set of jointree nodes $\hat{\mathbb{G}}$ starts out containing just the root (*PickNode* on line 4 always returns the root the first time it is called), as the root is the only initial source of diagnosability information, without which no critical path can be detected in the other twin plants. At each iteration we select a new node that has a neighbor in $\hat{\mathbb{G}}$ (line 4), and add it to $\hat{\mathbb{G}}$ as well as update the set of internal events Σ_{int} (line 5). (We will discuss the node selection heuristic in the next subsection.)

Jointree propagation is then run twice:

- firstly on $\hat{\mathbb{G}}$ (line 6) to remove inconsistent paths in it. This can lead to the removal of nondiagnosable states which in turn might cause the function $HasNondiagState(root)$ to return false and thus verify diagnosability, and
- secondly on $\hat{\mathbb{G}}_{\Sigma_{int}}$ (line 8) which is obtained by removing from all twin plants in $\hat{\mathbb{G}}$ all transitions labeled by events not in Σ_{int}. This allows the detection of whether a twin plant $\hat{G} \in \hat{\mathbb{G}}$ has a critical path whose global consistency can be verified by considering only the twin plants in $\hat{\mathbb{G}}$ (since it does not contain any event that appears in the rest of the tree).

The following two conditions also terminate the algorithm:

- The entire jointree is considered but none of the twin plants contains a critical path; hence the diagnosability of the fault is verified (line 12).

- Due to limited computational resources, the algorithm terminates (lines 13–18). In this case it returns the maximal set of components ω for which the existence of critical paths has been decided (but not yet verified against the rest of the system). In case critical paths exist in ω, then the larger this subsystem is, naturally, the more likely the whole system is not diagnosable; otherwise the reverse is true. As mentioned earlier, such an approximate solution is also useful in that it tells the user that on-line monitoring of this particular subsystem will not be sufficient to detect occurrences of the fault.

Jointree node selection

The heuristic used to select a jointree node to explore next can have a considerable impact on the number of nodes necessary to decide diagnosability. Instead of directly choosing a node, we first consider choosing an event which the new node might bring into Σ_{int}.

Let Σ_p denote the set of shared events appearing on a critical path p in some twin plant $\hat{G} \in \hat{\mathbb{G}}$. A reasonable heuristic is to expand Σ_{int} with some new event in $\Sigma_p \setminus \Sigma_{int}$ in the hope that p may at some point evolve into a new critical path that contains only internal events. To further focus the search, we will only consider events in $\Sigma_p \setminus \Sigma_{int}$ for paths p for which $|\Sigma_p \setminus \Sigma_{int}|$ is minimal.

Among these "eligible" events, we then select one that appears in the fewest nodes outside $\hat{\mathbb{G}}$. The idea here is to minimize the number of nodes that need to be included in $\hat{\mathbb{G}}$ for that event to be internal. Finally, after choosing the event, we add to $\hat{\mathbb{G}}$ a neighboring node containing that event.

Related Work

The diagnosability problem of discrete-event systems was introduced in (Sampath et al. 1995) where the authors solved it by considering a deterministic diagnoser for the global system and a part of the global model. The main drawback of this method is its exponential space complexity in the number of system states.

Jiang et al. (2001) and Yoo and Lafortune (2002) then propose new algorithms which are only polynomial in the number of states in G and which introduce the twin plant method. The question of efficiency is raised in (Cimatti, Pecheur, & Cavada 2003) where the authors propose to use symbolic model-checking to test a restrictive diagnosability property by taking advantages of efficient model-checking tools. But, still the diagnosability problem is seen as a test on a system whose size is exponential in the number of components, even when encoded by means of binary decision diagrams as in (Cimatti, Pecheur, & Cavada 2003).

Some of the most recent work decides either diagnosability or nondiagnosability but not both. The work by Rintanen and Grastien (2007) shows how to search for critical paths using SAT thus verifying nondiagnosability. On the other hand, the decentralized approach of Schumann and Pencolé (2007) can only verify diagnosability.

The approach of Pencolé (2004) is the closest to ours. It is based on the assumption that the observable behavior of every component is live, which is more restrictive than the assumption in (Sampath et al. 1995) which we adopted, namely that the observable behavior of the system (but not necessarily that of individual components) is required to be live. This restriction implies that it is sufficient to only search for a critical path in the twin plant $\hat{G}_F$ containing the fault. In (Pencolé 2004) this is done by iteratively synchronizing $\hat{G}_F$ with other local twin plants until diagnosability can be decided. In comparison to our approach this corresponds to the synchronization of all twin plants $\hat{\mathbb{G}}$ considered at each iterative step of Algorithm 1. In contrast we do not require this synchronization but achieve consistency by propagating messages with bounded event sets. Note that if we would adopt the same liveliness restriction we would only need to search the jointree root for critical paths and hence no outward propagation with diagnosability information would be required.

Conclusion and Future Work

We have presented a new algorithm for the diagnosability problem that addresses the fundamental bottleneck of the classical twin plant method. Specifically, only a subsystem is considered at a time in an iterative fashion, and more importantly, the construction of the global twin plant and the synchronization of twin plants in a subsystem being examined are both avoided. Instead local twin plants are made consistent by message passing on a jointree. Even if the computational resources are not sufficient our algorithm provides an approximate solution. Future work includes extending this approach to allow for an analysis of a nondiagnosability result so that information can be provided to the user regarding possible ways to rectify the system.

References

Cimatti, A.; Pecheur, C.; and Cavada, R. 2003. Formal verification of diagnosability via symbolic model checking. In *IJCAI-03*, 363–369.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

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

Pencolé, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *ECAI-04*, 43–47.

Pencolé, Y. 2005. Assistance for the design of a diagnosable component-based system. In *ICTAI-05*, 549–556.

Rintanen, J., and Grastien, A. 2007. Diagnosability testing with satisfiability algorithms. In *IJCAI-07*, 532–537.

Robertson, N., and Seymour, P. D. 1986. Graph minors II: Algorithmic aspects of treewidth. *Journal of Algorithms* 7:309–322.

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

Schumann, A., and Pencolé, Y. 2007. Scalable diagnosability checking of event-driven systems. In *IJCAI-07*, 575–580.

Schumann, A. 2008. *Towards Efficiently Diagnosing Large Scale Discrete-Event Systems*. Ph.D. Dissertation, Computer Science Laboratory, The Australian National University.

Shenoy, P. P., and Shafer, G. 1986. Propagating belief functions with local computations. *IEEE Expert* 1(3):43–52.

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