A Jointree Algorithm for Diagnosability and its Application to the Verification of Distributed Software Systems[*]

Anika Schumann **Wolfgang Mayer** **Markus Stumptner**

The Australian National University
Canberra ACT 0200, Australia
anika.schumann@anu.edu.au
tel: +61 2 6267 6216
fax: +61 2 6125 8651

University of South Australia
Adelaide SA 5095, Australia
{mayer,mst}@cs.unisa.edu.au
tel: +61 8 8302 3965
fax: +61 8 8302 3988

Abstract

Diagnosability is an essential property that determines how accurate any diagnostic reasoning can be on a system. While diagnosability in a discrete event system can be decided by synchronising finite state machines representing ambiguous paths in individual subsystems, this synchronisation operation remains prohibitively complex.

We propose a novel algorithm that exploits structure and locality properties of a system to avoid expensive synchronisation operations. By propagating concise summary information reflecting diagnosability of small subsystems, diagnosability of the entire system is computed incrementally. As a result, we obtain an efficient algorithm that can not only decide (non)diagnosability but that is also applicable in scenarios where computational resources are limited. We also show how our algorithm can be applied to analyse distributed (software) systems.

1 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.

While diagnosability of existing systems has been actively researched, the issue has become relevant also in the *design and development* phase of new systems, where diagnosability is used to verify that faults can be detected and isolated easily. Given that considerable parts of modern complex systems are implemented in software, it has become desirable to apply similar techniques also to computer programs. In this context, the sensor placement problem to ensure diagnosability translates into the *debugging problem*, where specific faults in executable programs must be distinguished and isolated. While programs are generally more accessible than integrated physical systems, tight constraints on developer effort and limited computational resources on embedded devices in distributed settings do restrict monitoring and diagnostic solutions. Hence, establishing sufficient but cost-effective fault isolation frameworks in software design remains a vital issue that must be addressed as part of the overall system design phase. In particular, systems should be designed such that likely faults in software can be isolated quickly and unambiguously. Fortunately, the same analysis techniques can help in the design of both hard- and software systems: in physical systems, diagnosability amounts to deciding the sensor placement problem, while in the software domain possible probes and monitoring frameworks must be implemented at strategic interfaces between dependent subsystems. In the following presentation, both domains are used interchangeably.

In this paper, we propose a formal framework for checking diagnosability of event-driven systems which is mainly motivated by two facts. Checking diagnosability means determining the existence of two behaviours 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 centralised 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 solving 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 behaviours in a global model, we propose to distribute the search based on local twin plants [Pencolé, 2004], represented as finite state machines (FSMs). Specifically, we exploit modularity of a system by organising the system components into a tree structure, the *jointree*,

[*]This work was in part supported by the Australian Research Council under project DP0560183.

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 synchronise 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 exchanging two messages per edge, the FSMs in the tree are collectively consistent. This allows to decide diagnosability by considering these FSMs in sequence instead of the large global twin plant.

We describe how messages represented as FSMs are computed based on projections of a FSM onto a subset of its events, and how diagnosability information can be propagated along with the messages. We employ an iterative procedure such that only a subset of the jointree is considered at a time, terminating the algorithm once a subset sufficient for deciding diagnosability has been determined. Our approach to use selective message passing in a jointree improves upon previous work by Pencolé (2004) in that we relax some of the assumptions underlying earlier work and achieve greater scalability by employing more efficient synchronisation mechanisms.

Since diagnosability analysis is a complex problem, our algorithm explicitly accounts for the possibility that the available resources may not be sufficient to calculate the precise solution for a given problem. Our incremental analysis algorithm ensures that in such cases it is able to provide an approximate solution to the diagnosability problem. Specifically, a subsystem where the existence of indistinguishable behaviours has been established but not yet verified against the rest of the system is returned. While an approximate solution cannot be used to verify that a system is definitely (non)diagnosable, it is useful to show that on-line monitoring of this particular subsystem will not be sufficient to detect occurrences of the fault.

This paper is organised as follows: in Section 2 we summarise the Twin Plant method of addressing the diagnosability problem for discrete event systems and outline the basic principles underlying jointrees. Section 3 presents our approach to use jointrees for diagnosability analysis, discussing our message-passing scheme and iterative algorithm. Differences to related work are discussed in Section 4, followed by a summary of our contributions and possible future research directions.

2 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.

2.1 Diagnosability of Discrete Event Systems

Similar to the diagnosis of discrete-event systems we consider a system of distributed systems $G_1, \ldots, G_n$. The behaviour of each component can be represented as a finite state machine (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}, communication events Σ_{s_i} shared with other components, unobservable fault events Σ_{f_i}, and other unobservable events Σ_{u_i}. Without loss of generality the communication events are assumed to be unobservable and observable and fault events to be specific to a G_i.

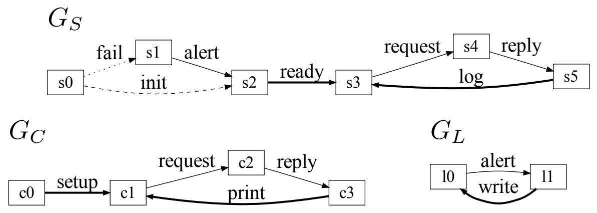

Figure 1: Three communicating processes modelled as FSMs. Bold, Solid, dashed, and dotted lines denote observable, shared, failure, and internal transitions, respectively.

Example 1 Assume a system of communicating processes is to be analysed to assess whether sufficient monitoring and logging capabilities have been put in place to detect particular types of faults in the system.

In our model, each process provides particular services to its peers and uses services provided by others by means of exchanging messages.[1] We abstract from concrete data contained in a message and represent each type of message as event. Hence, events and messages serve to coordinate the overall execution of processes in our system.

Figure 1 depicts a small distributed system of three subsystems represented as communicating FSMs: process G_S represents a server process that, once initialised, receives and executes requests from clients and returns the result. Subsystem G_C represents a client that communicates with the server process by sending requests and processing replies. Subsystem G_L implements a message archive where alerts are logged and stored for later analysis.

Interactions between subsystems are modelled as shared events; G_S and G_C communicate via events *request* and *reply*, while G_S sends alerts to G_L using an *alert* message.

Observable events in our model correspond to activity that can be perceived from a user's or administrator's point of view. In G_C, the completion of the initial setup stage, where parameters for the subsequent processing cycle are set, and the result of each request are directly visible. On the server side, event *ready* (denoting the completion of the initialisation stage) can be observed by the operator and a log file with entries for each request (event *log*) can be inspected. In G_L, the addition of a newly arrived message can be observed via event *write*.

In G_S, the initialisation step *init* cannot be observed directly. We further assume that the initialisation can fail (event *fail*); in this case, an alert event *alert* is sent to the logging subsystem. We assume that the system continues to execute, but some requests may not complete successfully due to failed initialisation of G_S.

We use this example to show how diagnosability analysis can help to assess whether the observable events are sufficient to infer the presence or absence of event *fail*.

[1]The jointree algorithm applied to a short example of interacting physical components is in [Schumann and Huang, 2008].

While the model in Figure 1 seems intuitively correct, we will show that the observable events are insufficient to decide the presence or absence of event *fail*. While the *alert* message is designed to make a failure observable in the logging component, G_L may delay its *write* operation indefinitely. Hence, this system becomes nondiagnosable. Results like these can be useful for system design, where requirements for different subsystems are laid out. By analysing a proposed design with respect to diagnosability of faults, requirements on logging and monitoring facilities can be verified.

Note that a monolithic model for the entire system is implicitly defined as the synchronised product, $G = Sync(G_1, \ldots, G_n)$[2] of all component models.

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 only one sequence contains a 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 behaviour of the system is *live* ($obs(p)$ is infinite for each infinite path p of the system). Then, diagnosability of a system with respect to F can be defined as follows:

Definition 1 (Diagnosability) F is diagnosable iff

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

Diagnosability checking thus requires the search for two infinite cyclic paths p and p' where F occurs in p but not in p', such that $obs(p) = obs(p')$. The pair (p, p') is called a *critical pair* [Cimatti, Pecheur, & Cavada, 2003]. Unless stated otherwise, we will use *path* to refer to a path that starts from the initial state of the system.

2.2 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]. From FSMs representing individual components, FSMs representing larger subsystems are constructed until diagnosability can be decided.

For each individual component model, the *interactive diagnoser* [Pencolé, 2005] is computed that returns the set of faults that could 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 each state corresponds to sets of possible faults that have occurred in a state in G_i: $\widetilde{X}_i \subseteq X_i \times \mathcal{F}$ with $\mathcal{F} \subseteq 2^{\Sigma_{f_i}}$; $\widetilde{\Sigma}_i = \Sigma_{o_i} \cup \Sigma_{s_i}$ is the set of shared and observable events; $\widetilde{x}_{0_i} = (x_{0_i}, \emptyset)$ represents the initial state where no faults are present; and $\widetilde{T}_i \subseteq \widetilde{X}_i \times \widetilde{\Sigma}_i \times \widetilde{X}_i$ denotes the set of transitions corresponding to observable and shared events. A transition $(x, \mathcal{F}) \xrightarrow{\sigma} (x', \mathcal{F}')$ exists if there is 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 $\mathcal{F}' = \mathcal{F} \cup (\Sigma'_i \cap \Sigma_{f_i})$.

Example 2 Figure 2 (top) depicts the interactive diagnoser for subsystem G_S in Figure 1. Following the transitions labelled $alert$ and $ready$ from the initial state of the diagnoser, we arrive at state $(s3, \{fail\})$, showing that the system contains a path to state $s3$ on which the sequence of observable and shared events is exactly $\langle alert, ready \rangle$ and the set of faults is exactly $\{fail\}$.

The *local twin plant* is then constructed by synchronising two instances $\widetilde{G}^l_i$ (left) and $\widetilde{G}^r_i$ (right) of the same interactive diagnoser based on the observable events $\Sigma_{o_i} = \Sigma^l_{o_i} = \Sigma^r_{o_i}$. Since only observable behaviours are compared, shared events must be distinguished between the two instances: in $\widetilde{G}^l_i$ (resp. $\widetilde{G}^r_i$), each shared event $\sigma \in \Sigma_{s_i}$ from $\widetilde{G}_i$ is renamed to l:$\sigma \in \Sigma^l_{s_i}$ (resp. r:$\sigma \in \Sigma^r_{s_i}$). As a result we obtain the local twin plant $\hat{G}_i = Sync(\widetilde{G}^l_i, \widetilde{G}^r_i)$.

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 a 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*; otherwise it is called *F-diagnosable*. 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.

Example 3 Figure 2 (bottom) depicts part of the twin plant for processes G_S in Figure 1. The top labels $x0, \ldots, x10$ 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). Oval nodes represent *fail*-nondiagnosable states. In this example, a part of the twin plans is shown where the fault cannot be deduced in all but the initial state.

A fault F is diagnosable in system G iff its *global twin plant (GTP)* $Sync(\hat{G}_1, \ldots, \hat{G}_n)$ 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. However, the GTP may be prohibitively large to perform this global analysis. In this paper, we propose a new algorithm that avoids building the global twin plant and operates on local twin plants instead. Since the existence of a critical path in a local twin plant does not imply nondiagnosability of the global system, the results of the local analysis must be propagated to other twin plants to decide diagnosability. As our main contribution in this paper we present a novel algorithm that ex-

[2]The result of *Sync* is a FSM whose state space is the Cartesian product of the state spaces of the processes, and whose transitions are synchronised such that any shared event always occurs simultaneously in all subsystems that communicate via it.

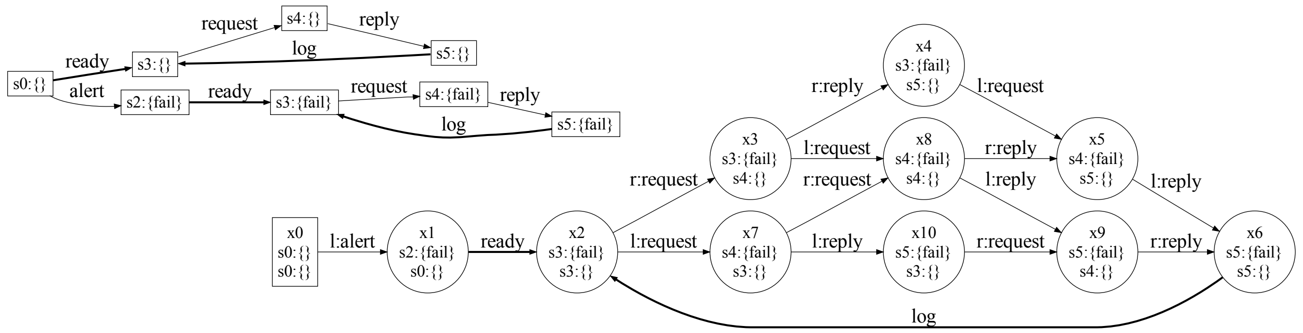

Figure 2: Diagnoser (top left) and part of a twin plant (bottom).

ploits a *jointree* to efficiently perform propagation and global diagnosability assessment.

2.3 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 labelled with sets of events satisfying two conditions:

Definition 2 (Jointree) Given a set of FSMs $G_1, \dots, G_n$ defined over events $\Sigma_1, \dots, \Sigma_n$ respectively, a jointree is a directed tree where each node is labelled 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 shared by two distinct nodes, then it also occurs in every node on the path connecting the nodes.

Example 4 Figure 3 (left) depicts a jointree for our three processes described in Figure 1. The label on each edge represents the intersection of the two neighbouring nodes, known as the *separator*.

Once a jointree has been 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 may be assigned multiple FSMs.

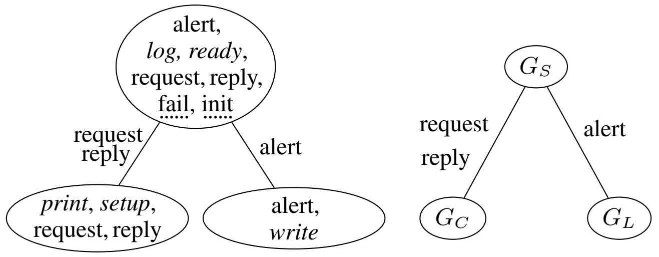

Figure 3: Jointree (left) and assignment of processes to jointree nodes (right). Observable events are set in italic font, failure events and internal events are underlined.

Once the FSMs have been assigned, the structure of the jointree can be exploited to guide our diagnosability assessment. In the following sections we show that it is sufficient to synchronise all FSMs assigned to the same node in the tree, followed by two message passing phases. The properties of the jointree then guarantee that consistency among all the FSMs has been achieved.

The efficiency of jointree propagation depends on the size of the FSMs computed. Hence it is desirable to minimise the number of FSMs assigned to a single node. For this purpose we can also use the well-known heuristics that minimise the number of events labelling a jointree node. The size of the largest label, minus 1, is known as the *width* of the jointree, and the minimum width among all possible jointrees for a given system is known as the *treewidth* of the system [Robertson and Seymour, 1986]. Efficient polynomial-time procedures, such as the *min-fill* heuristic, can be used to create jointrees of low width by exploiting system structure [Dechter, 2003]. These procedures also guarantee that the jointree nodes are labelled in such a way that no FSM is assigned to more than one node. Note further that nodes might be labelled with events that do not occur in any of the FSMs assigned to it. This is the case if FSMs interact in a cyclic way.

3 A Jointree Algorithm for Diagnosability

The synchronisation of all twin plants in a jointree would solve the diagnosability problem. However, for large systems this can be prohibitively expensive. This complexity can be avoided by synchronising only the twin plants in each jointree node, followed by message passing between adjacent tree nodes to propagate local results to a wider scope. After two cycles of message passing and synchronisation with local FSMs, global diagnosability can be decided.

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 labelling the jointree nodes. Here, the messages will themselves be FSMs. In the following we present an algorithm to compute these messages and show how diagnosability information can be propagated correctly between nodes. Subsequently, we detail our iterative algorithm that addresses the diagnosability problem.

3.1 Establishing consistency

While FSMs assigned to the same tree node are synchronised directly to obtain a local picture of the system behaviour, messages must be exchanged to achieve consistency between nodes.

Definition 3 (Global Consistency; Completeness) 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 G_i there exists a path p in the synchronised product $Sync(G_1, \ldots, G_n)$ that has with respect to Σ_i the same event sequence as p_i. A FSM G_i is *complete* iff it contains all globally consistent paths of G_i.

Each edge in a jointree partitions the tree into two subtrees, and a message sent over an edge represents a summary of the collective behaviour permitted by the sending side of the partition. A major advantage of this method is that this summary needs only to mention events given by the separator labelling the edge; the jointree construction ensures that this equals the intersection of the two sets of events across the partition.

A message can be computed by projecting a FSM onto a subset of its events.

Definition 4 (Projection) The *projection* $\Pi_{\Sigma'}(G) = \langle X', \Sigma', x_0, T' \rangle$ of a FSM G on events $\Sigma' \subseteq \Sigma$ is obtained from G by first contracting all transitions not labelled by an event in Σ' and then removing all states (except the initial state x_0) that are not the 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 G_c on its shared events.

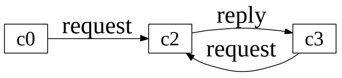

Figure 4: Projection $\Pi_{\{request, reply\}}(G_C)$

3.2 Message passing

To achieve consistency among the synchronised FSMs in a tree, each node requires a summary of the behaviour permitted by the FSMs in the remaining tree. Given the jointree properties, these summaries can be computed in only two passes over the jointree: one inward pass, in which the root "pulls" messages towards it from the rest of the tree, and one outward pass, in which the root "pushes" messages away from it towards the leaves. Once all messages have been exchanged, the FSM in each node is updated to reflect the information received from neighbouring nodes. As a result, all FSMs are complete and consistent.

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) neighbour n in the direction of the root. To compute this message, its FSM is synchronised with all messages it receives from its other neighbours (leaves do not have "other neighbours" and hence skip this step). The message that is subsequently sent to the node p closer to the tree root is the projection of this FSM onto the separator between n and p.

In the second, outward pass, each node (except the root) receives a message from its (unique) neighbour in the direction of the root. Again, a message is computed by synchronising a node's FSM with all messages it received from its other neighbours and by projecting the result onto the separator events between itself and the receiver of the message.

Finally, each node updates its associated FSM by synchronisation with messages received from its neighbours. As a result, each FSM G_i^c represents exactly the behaviour that is complete and possible in the global model implicitly defined by the jointree.

Example 5 Figure 5 illustrates the inward and outward propagation steps performed on the jointree of Figure 3 (right).

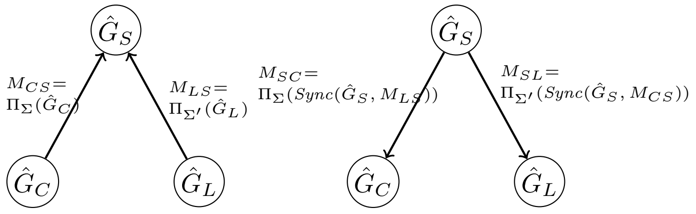
$\hat{G}_S$
$M_{CS}= \Pi_{\Sigma}(\hat{G}_C)$
$M_{LS}= \Pi_{\Sigma'}(\hat{G}_L)$
$\hat{G}_C$
$\hat{G}_L$
$\hat{G}_S$
$M_{SC}= \Pi_{\Sigma}(Sync(\hat{G}_S, M_{LS}))$
$M_{SL}= \Pi_{\Sigma'}(Sync(\hat{G}_S, M_{CS}))$
$\hat{G}_C$
$\hat{G}_L$

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

We have shown that after message propagation has completed, the FSMs in each node are consistent and complete:[3]

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

In particular it follows that for every path p in $\hat{G}_i' = \Pi_{\Sigma_i}(Sync(\hat{G}_1, \ldots, \hat{G}_n))$ there is also an equivalent path p_i in $\hat{G}_i^c$ defined over the same event sequence as p, and vice versa.

While Theorem 1 establishes desirable properties, it can be shown that it is insufficient to decide diagnosability, since some critical paths may be lost due to the projection operation.

In general, 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 in addition to the message passing algorithm outlined previously.

3.3 Propagation of diagnosability information

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

[3]Proofs of the theorems in this paper are in [Schumann, 2007].

It has been shown that if any twin plant of a jointree node contains a consistent critical path, then the fault F is nondiagnosable [Schumann, 2007]. Hence, diagnosability can be decided by searching for critical paths in the individual FSMs in the jointree after the message passing phase is completed.

The root can already be examined for critical paths after the inward propagation phase: (i) the synchronisation of the root with all its incoming messages results in a globally consistent twin plant, and (ii), since the fault F appears in the root, the FSM already contains diagnosability information, that is, the classification of states into diagnosable and nondiagnosable ones. If the root does not contain a nondiagnosable state, the entire system is known to be diagnosable. Otherwise, the outward propagation phase must be carried out to determine whether another jointree node has a critical path.

Once propagation is complete, every state of a twin plant comprises a tuple $(\hat{x}_1, \ldots, \hat{x}_n)$. In particular, each state contains a state from $\hat{G}_F$ that has been received and synchronised with the local FSM as part of the messages pushed from the root in the outward propagation phase. To ensure diagnosability information is preserved, we must ensure that no path to a nondiagnosable state is lost in this process.

Recall that the projection operation applied to compute the outward message removes all states that are no longer a target of a transition labelled by a separator event in Σ. This can lead to the removal of nondiagnosable states, resulting in the incomplete propagation of diagnosability information.

Example 6 Consider the twin plant $\hat{G}_u$ shown in Figure 6 (left). Assume a message $\mathcal{P}_u = \Pi_{\{s1\}}(\hat{G}_u)$ is to be computed with respect to event set $\{s_1\}$ and sent to $\hat{G}_v$. By projecting $\hat{G}_u$ onto $\{s_1\}$, the nondiagnosable state $u1$ is eliminated. This results in the consistent twin plant $\hat{G}_v^c = Sync(\Pi_{\{s1\}}(\hat{G}_u), \hat{G}_v)$ obtained by synchronisation of $\mathcal{P}_u$ with $\hat{G}_v$. However, $\hat{G}_v^c$ does not contain any critical paths, although it should contain one (as shown by the properly synchronised FSM $\hat{G}_v^{c\prime} = \Pi_{\Sigma_v}(Sync(\hat{G}_u, \hat{G}_v)))$.

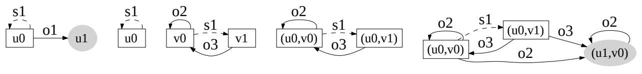

Figure 6: FSMs $\hat{G}_u$, $\mathcal{P}_u$, $\hat{G}_v$, $\hat{G}_v^c$, and $\hat{G}_v^{c\prime}$ (from left to right).

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. This guarantees that $\hat{G}'^c$ has a critical path iff $\Pi_{\Sigma'}(Sync(\hat{G}, \hat{G}'))$ has one, where Σ' is the event set labelling the jointree node of $\hat{G}'$.

To achieve this it is necessary to annotate every diagnosable state $\hat{x}$ in a message to capture whether it has a *nondiagnosable local future*, that is, whether there is a transition sequence starting in $\hat{x}$ and leading to a nondiagnosable state $\hat{x}_k$ such that none of the transition events is kept in the projection:

Definition 5 (Nondiagnosable Local Future) Let $\hat{G}$ and $\hat{G}'$ be two FSMs associated with adjacent nodes in a jointree connected by an edge labelled Σ_{sep}, and let $\hat{x}_k$ denote a nondiagnosable state in $\hat{G}$. Then, a diagnosable state $\hat{x} \in \hat{G}$ has a *nondiagnosable local future* iff there exists a transition sequence

$$\hat{x} \xrightarrow{\sigma_1} \hat{x}_1 \cdots \xrightarrow{\sigma_k} \hat{x}_k$$

in $\hat{G}$ such that none of the events $\sigma_1, \ldots, \sigma_k$ are in Σ_{sep}.

We capture this information by adding additional nondiagnosable subgraphs to the FSM $\Pi_{\Sigma_{sep}}(\hat{G})$ obtained by projection of $\hat{G}$: for every diagnosable state $\hat{x} \in \hat{G}$ that has a nondiagnosable local future w.r.t. Σ_{sep}, a nondiagnosable *extended* terminal state $ext(\hat{x})$ and a transition $\hat{x} \xrightarrow{ext} ext(\hat{x})$ are added to ensure that the critical path is not lost in the projection.

Example 7 The left part of Figure 7 illustrates the message M_{SL} sent from $\hat{G}_S$ (see Figure 2) to $\hat{G}_L$. The only diagnosable state $x0$ in $\hat{G}_S$ does not have a nondiagnosable local future, since all outgoing transitions are kept in the projection $\Pi_{\{alert\}}$, and the nondiagnosable state $x1$ is included in M_{SL}.

In contrast, the state $x0$ of the message M_{SC} shown on the right of Figure 7 does have a nondiagnosable local future w.r.t. $\{l : \textit{request}, r : \textit{request}, l : \textit{reply}, r : \textit{reply}\}$ according to Definition 5. The path $x0 \xrightarrow{l:alert} x1$ in $\hat{G}_S$ leads to the nondiagnosable state $x1$, but is eliminated by the projection $\Pi_{\{request, reply\}}$. Hence, an extended state $ext(x0)$ must be introduced in M_{SC} as depicted on the right of Figure 7.

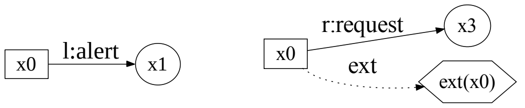

Figure 7: Message M_{SL} (left) and part of message M_{SC} (right). Circles denote nondiagnosable states and hexagon shapes extended states, respectively.

Note that there is no need to introduce artificial states for a *nondiagnosable* state $\hat{x}'$. This results from the fact that all states reachable from $\hat{x}'$ via transitions labelled by events not kept in the projection can only be part of a nondiagnosable cycle if there is also a nondiagnosable cycle with state $\hat{x}'$ (according to the synchronisation operation). Hence nondiagnosability can be verified correctly based only on the latter.

Using the extended messages diagnosability can be decided:

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.*

3.4 Iterative jointree propagation

Since the complexity of our approach results from the complexity of the message propagation (and not the jointree construction), the efficiency of the algorithm can further be improved by limiting the scope of propagation within the jointree. We propose an algorithm that iteratively extends propagation in the tree until a subtree sufficiently large to decide (non)diagnosability has been processed, or computational resources have been exhausted. In the latter case, our algorithm

Algorithm 1 CHECKDIAGNOSABILITY(jointree: J)

```
1:  $\hat{\mathbb{G}} \leftarrow \emptyset$                                   nodes in $J$ being considered
2:  $\Sigma_{int} \leftarrow \emptyset$                                  events internal to $\hat{\mathbb{G}}$
3:  repeat
4:      $v \leftarrow PickNode(J, \hat{\mathbb{G}})$
5:      $\langle \hat{\mathbb{G}}, \Sigma_{int} \rangle \leftarrow AddToScope(v)$
6:      $Propagate(\hat{\mathbb{G}})$
7:      $\hat{\mathbb{G}}_{\Sigma_{int}} \leftarrow ProjectPaths(\hat{\mathbb{G}}, \Sigma_{int})$
8:      $Propagate(\hat{\mathbb{G}}_{\Sigma_{int}})$
9:      if $ExistsTwinPlantWithCritPath(\hat{\mathbb{G}}_{\Sigma_{int}})$ then
10:         return $GetCritPath(\hat{\mathbb{G}}_{\Sigma int})$
11:     end if
12: until $\hat{\mathbb{G}} = J$ or $IsDiagnosable(root)$
        or $\neg SufficientMemory(\hat{\mathbb{G}})$
13: if $SufficientMemory(\hat{\mathbb{G}})$ then
14:     return "F is diagnosable"
15: else
16:     $\omega$ ← set of components included in $\hat{\mathbb{G}}$
17:     if $ExistsTwinPlantWithCritPath(\hat{\mathbb{G}})$ then
18:         return "$\omega$ has a critical path"
19:     else
20:         return "$\omega$ has no critical path"
21:     end if
22: end if
```

returns a conservative approximation of the exact solution that can help to guide further analysis of a system.

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 in p, since other twin plants cannot affect the path. Our aim is it to find a critical path defined over as few events as possible to limit the scope of the jointree that must be processed and to lower computational effort.

We search for such a path by iteratively increasing the set of jointree nodes (twin plants) $\hat{\mathbb{G}}$ under consideration and limit our search to critical paths defined over *internal events* Σ_{int} in $\hat{\mathbb{G}}$ that do not appear in the remaining jointree. If such a path can be found, nondiagnosability has been shown and the search terminates.

Algorithm 1 outlines our search procedure. Assume that *root* denotes the root node chosen for propagation; function *PickNode* returns *root* upon initial invocation, and a node in J neighbouring $\hat{\mathbb{G}}$ on subsequent invocations. Node selection heuristics are discussed in the next section. Initially, the root node is the only source of diagnosability information. In each iteration a new node that has a neighbour in $\hat{\mathbb{G}}$ is selected (line 4), expanding the scope of our search by adding it to $\hat{\mathbb{G}}$ and updating event set Σ_{int}. Jointree propagation is then run twice:

1. On $\hat{\mathbb{G}}$ (line 6) to remove inconsistent paths. This can lead to the removal of nondiagnosable states which in turn may cause the root to become diagnosable ($IsDiagnosable(root)$ is *true*) and thus verify diagnosability;

2. On $\hat{\mathbb{G}}_{\Sigma_{int}}$ (line 8), which is obtained by removing from all twin plants in $\hat{\mathbb{G}}$ the transitions labelled by events not in Σ_{int} (line 7). This allows to detect if 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. In this case, Algorithm 1 stops and returns the critical path that implies nondiagnosability (line 10).

The algorithm continues until one of the following conditions is satisfied:

- The root node (and hence the entire system) has been shown diagnosable. Note that it is indeed sufficient to check only the root node, since if the root has no nondiagnosable states, none of the messages it propagates and hence no twin plant includes a nondiagnosable state.
- The entire jointree is considered, but none of the twin plants contains a critical path; hence the diagnosability of the system is verified (line 14). The case where *root* contains a critical path is covered by line 10, since $\hat{\mathbb{G}} = J$ implies that Σ_{int} contains all events.
- The available resources have been exhausted (lines 16–21). In this case the maximal subsystem ω for which the existence of critical paths has been decided (but not yet verified against the rest of the system) is returned.

  Any critical path in ω can be interpreted as hint indicating nondiagnosability (of at least the isolated subsystem considered so far). 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. Such an approximate solution is also useful in that it implies that on-line monitoring of this particular subsystem will not be sufficient to reliably detect faults.

Example 8 Applied to our running example, Algorithm 1 selects the jointree node containing G_S in the initial iteration. Since the events in any critical path in $\hat{G}_S$ are not internal to G_S, neither diagnosability nor nondiagnosability can be established, and the scope of the search must be expanded to include another subsystem. Assume G_L is selected and added to the scope, leading to $\Sigma_{int} = \{l : alert, r : alert, log, ready, write\}$. Again, (non)diagnosability cannot be decided since all critical paths in the $\hat{\mathbb{G}}$ contain either a *request* or a *reply* event shared with $\hat{G}_C$. After extending $\hat{\mathbb{G}}$ with the remaining subsystem G_C, Σ_{int} contains the events shared by G_S and G_C. Now, a critical path in $\hat{\mathbb{G}}_{\Sigma_{int}}$ exists and the algorithm terminates in line 10.

Had the size of the system exceeded the available resources after the second iteration, our algorithm would have returned that the subsystem $\{G_L, G_S\}$ is potentially nondiagnosable (line 18), approximating the exact result.

3.5 Node selection heuristics

The heuristics used to select a jointree node to explore next can have considerable impact on the number of nodes necessary to decide diagnosability. Instead of directly choosing a node, we select nodes based on the set of events that are introduced into Σ_{int} by a candidate node.

Let Σ_p denote the set of shared events appearing in a critical path p in a twin plant $\hat{G} \in \hat{\mathbb{G}}$. Our heuristic is to expand Σ_{int}

with a 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 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 select one that appears in the fewest nodes outside $\hat{\mathbb{G}}$. The idea here is to minimise the number of nodes that need to be included in $\hat{\mathbb{G}}$ for that event to be internal. After choosing an event, we iteratively add to $\hat{\mathbb{G}}$ the neighbouring nodes containing that event.

4 Related Work

The diagnosability problem of discrete-event systems was first addressed in [Sampath et al., 1995] by constructing a deterministic diagnoser for the global system model. The main drawback of this method is its space complexity that is exponential in the number of system states.

Jiang et al. (2001) and Yoo and Lafortune (2002) proposed different algorithms that are of polynomial complexity and introduce the twin plant method. The question of efficiency is also raised in [Cimatti, Pecheur, & Cavada, 2003] where the authors propose to use symbolic techniques to test a restrictive diagnosability property by taking advantage of efficient model-checking tools. However, diagnosability assessment remains exponential in the number of components, even when encoded by means of binary decision diagrams as in [Cimatti, Pecheur, & Cavada, 2003].

More recent work aims at establishing either diagnosability or nondiagnosability, but not both. The work by Rintanen and Grastien (2007) shows how to detect nondiagnosability by searching for critical paths using SAT. If the algorithm cannot find a critical path it does not imply that there is indeed none, and it remains unknown whether the system is diagnosable or not. Conversely, the decentralised approach of Schumann and Pencolé (2007) can only establish diagnosability.

The approach of Pencolé (2004) is the closest to ours. It is based on the assumption that the observable behaviour of every component is live, which is more restrictive than our assumption (and that of [Sampath et al., 1995]), namely that the observable behaviour 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 synchronising $\hat{G}_F$ with other local twin plants until diagnosability can be decided. In comparison to our approach this corresponds to the synchronisation of all twin plants $\hat{\mathbb{G}}$ considered at each iterative step of Algorithm 1. We do not require this synchronisation but achieve consistency by propagating messages with bounded event sets. If we adopted similar liveliness restriction, it would be sufficient to search the jointree root for critical paths and skip outward propagation.

5 Conclusion and Future Work

We have presented a new approach to attack the diagnosability problem that addresses the fundamental complexity bottleneck of the classical twin plant method. By limiting our iterative analysis to a subsystem at a time, both the construction of the global system model as well as the synchronisation of local twin plants for entire subsystems can be avoided. Instead, local twin plants are made consistent by passing messages in a jointree representing clusters of related system components represented as finite state machines. Even with this improved algorithm, computational resources may be insufficient to find an exact solution and our algorithm returns an approximate solution that may guide further analysis. As part of future work we plan to extend our approach such that possible causes of nondiagnosability can be isolated and to explore ways to restore diagnosability.

References

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

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

[Jiang et al., 2001] 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, 2004] Pencolé, Y. 2004. Diagnosability analysis of distributed discrete event systems. In *ECAI-04*, 43–47.

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

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

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

[Sampath et al., 1995] 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 and Pencolé, 2007] Schumann, A., and Pencolé, Y. 2007. Scalable diagnosability checking of event-driven systems. In *IJCAI-07*, 575–580.

[Schumann, 2007] Schumann, A. 2007. *Towards Efficiently Diagnosing Large Scale Discrete-Event Systems*. Ph.D. Dissertation, Computer Science Laboratory, The Australian National University.

[Schumann and Huang, 2008] Schumann, A., and Huang J. 2008. A scalable jointree algorithm for diagnosability. In *AAAI-08*.

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

[Yoo and Lafortune, 2002] 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.