Local Consistency and Junction Tree for Diagnosis of Discrete-Event Systems

Priscilla Kan John*
priscilla.kanjohn@anu.edu.au

Alban Grastien*
alban.grastien@nicta.com.au

Keywords: Diagnosis, Discrete-event systems, Junction Tree, Local consistency

Abstract

We extend the decentralised/distributed approach of diagnosis of discrete-event systems modeled using automata. The goal is to avoid computing a global diagnosis, which is expensive, and to perform local diagnoses instead. To still ensure global consistency, we transform the topology of the system into a junction tree where each vertex represents a subsystem. Local consistency between the diagnoses of these subsystems ensures global consistency due to the tree structure. This technique will work best for systems whose natural structure is close to a tree structure, as the generated automata will be of reasonable size.

1 Introduction

Nowadays, many technical systems are highly automated, if not completely controlled by computers. As such systems increase in complexity, their supervision becomes more and more challenging such that there is a strong need to automate the task. New methods are required to meet this objective. We are here concerned with the model-based diagnosis of systems modeled as discrete-event systems (DES, [Cassandras and Lafortune, 1999]).

It is well-known that the diagnosis of discrete-event systems [Lamperti and Zanella, 2003] can be seen as the computation of all the trajectories on the model consistent with the observations. This can be done by unfolding the model according to the observations. The main challenge is then to cope with the complexity of the task as the representation of these trajectories is usually exponential in the number of components in the system.

To deal with systems of increasing size, several approaches have been investigated. A first approach trades time for space: the model of the system is compiled into a structure called the Sampath diagnoser [Sampath *et al.*, 1995] to enable efficient on-line computation. However, this structure is double exponential in the number of components and cannot be built in most cases [Rintanen, 2007]. Use of symbolic tools has also been proposed, giving interesting results [Schumann *et al.*, 2004; 2007; Grastien *et al.*, 2007].

Another option is to consider local computations. Rather than computing the trajectories on the whole system, the trajectories are computed locally. The problem is then to make sure that the local sets of trajectories are consistent with each other. Unfortunately, local (pairwise) consistency does not ensure global consistency; worst, an algorithm that refines the local diagnoses pairwisely may not terminate. Methods were proposed to avoid global computation [Pencolé and Cordier, 2005; Cordier and Grastien, 2007; Su and Wonham, 2005; Fabre *et al.*, 2005], but these methods do not scale up nicely.

The complexity of numerous algorithms in different domains drops when applied to trees. This is especially relevant to the case of ensuring global consistency as local consistency ensures global consistency on a tree structure. A popular solution to convert a graph into a tree is to make it into a *junction tree* [Huang and Darwiche, 1996], where the vertices are gathered in clusters. We thus transform the topological graph of the system into a junction tree where each cluster corresponds to a subsystem. The diagnosis is performed locally on each cluster, and local consistency is applied until a fixpoint is reached.

The paper is divided as follows: we first present basic notations on languages and diagnosis. In Section 3, we discuss the issues of distributed diagnosis, and the central notion of *consistency*. Our approach based on junction trees and local consistency is presented Section 4.

2 Preliminaries

In this section, we present basic notations on language and how it applies to the diagnosis of discrete-event systems.

2.1 Language Formalism

Let Σ be any set. We denote $\Sigma^\star$ the set of all finite sequences on Σ; an element $\sigma = e_1.\cdots.e_n \in \Sigma^\star$ is called a *word* over Σ; the empty word is denoted ε. A *language* $\mathcal{L}$ over Σ is a subset of $\Sigma^\star$. The projection on Σ' of a word σ over $\Sigma \supseteq \Sigma'$ denoted $P_{\Sigma\to\Sigma'}(\sigma)$ keeps all the elements of σ in Σ'. Formally,

$$P_{\Sigma\to\Sigma'}(\sigma) = \begin{cases} \varepsilon & \text{if } \sigma = \varepsilon \\ P_{\Sigma\to\Sigma'}(\sigma') & \text{if } \sigma = e.\sigma' \text{ and } e \in \Sigma \setminus \Sigma' \\ e.P_{\Sigma\to\Sigma'}(\sigma') & \text{if } \sigma = e.\sigma' \text{ and } e \in \Sigma' \end{cases}$$

---

*Priscilla Kan John and Alban Grastien are with National Information and Communications Technology Australia (NICTA) and The Research School of Information Sciences and Engineering, Australian National University, Canberra.

The projection on Σ' of a language $\mathcal{L}$ over Σ is denoted $P_{\Sigma\to\Sigma'}(\mathcal{L})$ and defined by $\{P_{\Sigma\to\Sigma'}(\sigma) \mid \sigma \in \mathcal{L}\}$. The inverse operation $P^{-1}_{\Sigma\to\Sigma'}$ of the projection from Σ to Σ' generates all the finite words on Σ whose projection on Σ' is the parameter: $P^{-1}_{\Sigma\to\Sigma'}(\mathcal{L}) = \{\sigma \in \Sigma^\star \mid P_{\Sigma\to\Sigma'}(\sigma) \in \mathcal{L}\}$.

The synchronous product $\otimes$ between two languages $\mathcal{L}_1$ over Σ_1 and $\mathcal{L}_2$ over Σ_2 computes all the words over $\Sigma_1 \cup \Sigma_2$ whose projection on Σ_i is $\mathcal{L}_i$: $\mathcal{L}_1 \otimes \mathcal{L}_2 = \{\sigma \in (\Sigma_1 \cup \Sigma_2)^\star \mid \forall i \in \{1, 2\},\ P_{\Sigma_1\cup\Sigma_2\to\Sigma_i}(\sigma) \in \mathcal{L}_i\}$.

The local consistency operation of language $\mathcal{L}_1$ over Σ_1 on $\mathcal{L}_2$ over Σ_2 denoted $cons_{\Sigma_1,\Sigma_2}(\mathcal{L}_1, \mathcal{L}_2)$ returns the minimum sublanguage of $\mathcal{L}_2$ such that the synchronous product with $\mathcal{L}_1$ is not modified: $cons_{\Sigma_1,\Sigma_2}(\mathcal{L}_1, \mathcal{L}_2) = \{\sigma \in \mathcal{L}_2 \mid P_{\Sigma_2\to\Sigma_1\cap\Sigma_2}(\sigma) \in P_{\Sigma_1\to\Sigma_1\cap\Sigma_2}(\mathcal{L}_1)\}$ or equivalently $cons_{\Sigma_1,\Sigma_2}(\mathcal{L}_1, \mathcal{L}_2) = \mathcal{L}_2 \cap P^{-1}_{\Sigma_2\to\Sigma_1\cap\Sigma_2}(P_{\Sigma_1\to\Sigma_1\cap\Sigma_2}(\mathcal{L}_1))$.

2.2 Diagnosis of Discrete-Event Systems

We consider a system whose state can be described as the assignment of *state variables* over a discrete domain. We consider the evolution of the state variables to also be discrete. The set of all – including unexpected – possible behaviours of this system is a language denoted Mod over the set of events Σ that can possibly occur on the system. The set of events is partitioned into *observable* Σ_o and *unobservable* Σ_u events. The occurrence of an observable event generates an observation. While the system is running, it generates a flow of observations. The sequence of observable events that occur on the system is a word on Σ_o.However, it is not always possible to determine precisely the sequence of observable events from the sequence of observations received. We represent the sequence of observations by a language denoted Obs, where each word in Obs is a sequence of observable events consistent with the observations received.

The diagnosis of the system is the problem of determining what possibly happened on the system given the observations on its behaviour. This can be simply computed by

$$\Delta = \mathrm{Mod} \otimes \mathrm{Obs}. \qquad (1)$$

Languages can be represented by several tools. Regular languages are often represented by automata or Petri nets. The problem with these tools is that of *state explosion*. The size of these structures is exponential in the number of state variables, which makes them difficult to use in practice.

3 Consistency in a Distributed Model

Real-world systems are often distributed by nature, *i.e.* a set of interconnected components. The global behaviour of the system is complex, whereas each component has a simple behaviour. Recent approaches take advantage of this distributed nature to avoid computational blow up.

3.1 Distributed Modeling

Modern technical systems are usually formed by combining simple components with simple behaviours leading to a device that exhibits complex behaviours. Rather than modeling the whole system, it is often preferable to model each component separately for many good reasons: fewer chances to make mistakes or forget behaviours, reusability, compactness.

Since the system is a set of components, each component γ_i can be modeled separately: Mod_i defined on alphabet Σ_i. Some formalisms consider that components share variables. Here, without loss of generality, we consider that components share events such that an event shared by several components must occur on each component at the same time. Other events may occur in a completely concurrent manner.

The system $\Gamma = \{\gamma_1, \ldots, \gamma_n\}$ composed of components $\gamma_1, \ldots, \gamma_n$ is modeled as a set of languages $d\mathrm{Mod} = \{\mathrm{Mod}_1, \ldots, \mathrm{Mod}_n\}$ over the alphabets $\Sigma_1, \ldots, \Sigma_n$. The global model of the system is implicitly defined by $\mathrm{Mod} = \mathrm{Mod}_1 \otimes \cdots \otimes \mathrm{Mod}_n$ but never explicitly computed.

3.2 Distributed Diagnosis and Global Consistency

The alphabet Σ_i that represents the events of each component γ_i is partitioned into observable events Σ_{io} and unobservable events Σ_{iu}. Moreover, we consider that the global observations Obs on the system can be distributed into Obs_i defined on Σ_{io} such that $\mathrm{Obs} = \mathrm{Obs}_1 \otimes \cdots \otimes \mathrm{Obs}_n$.

A *distribution* $\mathcal{S} = \{S_1, \ldots, S_m\} \in 2^{2^\Gamma}$ is a set of subsets of Γ such that $\mathcal{S}$ covers Γ: $S_1 \cup \cdots \cup S_m = \Gamma$. A *distributed diagnosis* is a mapping that associates with each subset S_i a diagnosis $d\Delta(S_i)$ such that $d\Delta(S_1) \otimes \cdots \otimes d\Delta(S_m) = \Delta$. The literature usually considers that $\mathcal{S}$ is a partition of Γ [Pencolé and Cordier, 2005].

The local diagnoses can be simply computed by:

$$d\Delta(S_i) = \bigotimes_{\gamma_k \in S_i} (\mathrm{Mod}_k \otimes \mathrm{Obs}_k). \qquad (2)$$

This returns a distributed diagnosis that can be easily computed as long as any S_i contains a small number of elements. However, the local diagnoses can be inconsistent with each other. Basically, some words of $d\Delta(S_i)$ should be removed because they disappear when S_i is synchronised with other S_j elements. Thus, we are interested by the globally consistent distributed diagnosis:

A distributed diagnosis $d\Delta$ is *globally consistent* if $\forall i \in \{1, \ldots, m\}$, $d\Delta(S_i) = P_{\Sigma\to\Sigma_{S_i}}(\Delta)$ where $\Sigma_{S_i} = \bigcup_{\gamma_k \in S_i} \Sigma_k$.

The globally consistent distributed diagnosis is such that no word of any $d\Delta(S_i)$ can be removed. We want to compute this *refined* distributed diagnosis but the goal is to avoid the computation of Δ.

3.3 Local Consistency

The local consistency property requires that any pair of local diagnoses are consistent. Formally, a distributed diagnosis $d\Delta$ is locally consistent if $\forall \{S_1, S_2\} \subseteq \mathcal{S}$, $P_{\Sigma_{S_1}\to\Sigma_{S_1}\cap\Sigma_{S_2}}(d\Delta(S_1)) = P_{\Sigma_{S_2}\to\Sigma_{S_1}\cap\Sigma_{S_2}}(d\Delta(S_2))$.

It is possible to refine a distributed diagnosis using local consistency as presented in Algorithm 1. After the distribution is performed, and a local diagnosis is computed for each subsystem, the algorithm takes pairs of subsystems and performs a local consistency on these diagnoses. Basically, the idea is to remove the word of $d\Delta(S_1)$ that cannot be synchronised with any word of $d\Delta(S_2)$, and vice versa. The local consistencies can actually be performed in any order.

Algorithm 1 Distributed diagnosis algorithm based on local consistency

input $\Gamma, \{\mathrm{Mod}_1, \dots, \mathrm{Mod}_n\}, \{\mathrm{Obs}_1, \dots, \mathrm{Obs}_n\}$
$\mathcal{S} = \{S_1, \dots, S_m\} := \mathit{distribution}(\Gamma)$
for all $i \in \{1, \dots, m\}$ **do**
  $d\Delta(S_i) = \bigotimes_{\gamma_k \in S_i} (\mathrm{Mod}_k \otimes \mathrm{Obs}_k)$
repeat
  for all $\{S_1, S_2\} \subseteq \mathcal{S}$ **do**
    $d\Delta(S_2) := \mathit{cons}_{\Sigma_{S_1}, \Sigma_{S_2}}(d\Delta(S_1), d\Delta(S_2))$
    $d\Delta(S_1) := \mathit{cons}_{\Sigma_{S_2}, \Sigma_{S_1}}(d\Delta(S_2), d\Delta(S_1))$
until $d\Delta$ is stable

However, as shown in [Su and Wonham, 2005], local consistency does not ensure global consistency. Moreover, because the languages may be infinite, no fix-point is reached in the worst case; the algorithm does not terminate. As noticed by the authors in [Su and Wonham, 2005] , both problems disappear when the topology of the system forms a tree.

A *topology* of a distributed representation $\mathcal{S}$ of the system is a graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ where $\mathcal{V} = \mathcal{S}$ is the set of vertices and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a symmetric and anti-reflexive set of edges such that $\forall \{S, S'\} \subseteq \mathcal{V}, \forall e \in \Sigma_S \cap \Sigma_{S'}, \exists S_0, \dots, S_{k+1}$ such that:

- $S_0 = S$ and $S_{k+1} = S'$
- $\forall i \in \{1, \dots, k\},\ e \in \Sigma_{S_i}$, and
- $\forall i \in \{0, \dots, k\},\ \langle S_i, S_{i+1} \rangle \in \mathcal{E}$

Two subsystems that share an event are connected through an edge, or through a chain of edges where intermediate subsystems also share this event.

The graph $\mathcal{G}$ is a tree if for any pair S_i and S_j, there is exactly one path on the graph that contains no loop and leads from S_i to S_j. Provided that the distribution of the system can be represented by a tree, the algorithm presented above terminates and is sound.

Because of space requirement, we only give a simplified proof of this last result. Similar proofs can be found in [Su and Wonham, 2005] with slightly different definition of the topology. In particular in [Su and Wonham, 2005], an edge connects two vertices whenever these two vertices share an event, while here we only required them to be connected through a chain of vertices that share this event.

Consider that the distribution generates a tree $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$. Consider that the local diagnosis $d\Delta(S_i)$ is computed for each subsystem $S_i \in \mathcal{V}$ and that the local consistency procedure is applied until stability is reached.

Choose randomly some subsystem $S_i \in \mathcal{V}$. We want to determine whether $P_{\Sigma \to \Sigma_i}(\Delta) = d\Delta(S_i)$ which states that the diagnosis is globally consistent. To do so, we set S_i as the root of the tree $\mathcal{G}$. Let $X \subseteq \mathcal{V}$ be a subset of subsystems, we denote $\Sigma_X = \bigcup_{S \in X} \Sigma_S$ and $\mathcal{L}_X = \bigotimes_{S \in X} d\Delta(S)$. We build X incrementally from $X = \{S_i\}$ by adding $S_k \notin X$ such that $S_j \in X$ and $\langle S_k, S_j \rangle \in \mathcal{E}$; note that because of the definition of $\mathcal{G}$ and since $\mathcal{G}$ is a tree, $\Sigma_X \cap \Sigma_k = \Sigma_j \cap \Sigma_k$. We note $X' = X \cup \{S_k\}$. We prove by induction that for any $S_p \in X$, $P_{\Sigma_X \to \Sigma_p}(\mathcal{L}_X) = d\Delta(S_p)$.

- This is clearly the case for $X = \{S_i\}$.
- $P_{\Sigma_X \to \Sigma_X \cap \Sigma_k}(\mathcal{L}_X)$
  $= P_{\Sigma_X \to \Sigma_j \cap \Sigma_k}(\mathcal{L}_X)$ (because $\Sigma_X \cap \Sigma_k = \Sigma_j \cap \Sigma_k$)
  $= P_{\Sigma_j \to \Sigma_j \cap \Sigma_k}(P_{\Sigma_X \to \Sigma_j}(\mathcal{L}_X))$ (since $\Sigma_j \cap \Sigma_k \subseteq \Sigma_j \subseteq \Sigma_X$)
  $= P_{\Sigma_j \to \Sigma_j \cap \Sigma_k}(d\Delta(S_j))$ (by induction)
  $= P_{\Sigma_k \to \Sigma_j \cap \Sigma_k}(d\Delta(S_k))$ (by local consistency)
  $= P_{\Sigma_k \to \Sigma_X \cap \Sigma_k}(d\Delta(S_k))$ (because $\Sigma_X \cap \Sigma_k = \Sigma_j \cap \Sigma_k$)

  Thus, $\mathcal{L}_X$ and $d\Delta(S_k)$ are locally consistent. Thus, for any $S_p \in X$, $P_{\Sigma_{X'} \to \Sigma_p}(\mathcal{L}_X \otimes d\Delta(S_k)) = P_{\Sigma_X \to \Sigma_p}(\mathcal{L}_X) = d\Delta(S_p)$, and $P_{\Sigma_{X'} \to \Sigma_k}(\mathcal{L}_X \otimes d\Delta(S_k)) = d\Delta(S_k)$.

Thus, for $X = \mathcal{S}$, we have the following result: $\forall S_i \in \mathcal{S}$, $P_{\Sigma \to \Sigma_i}(\Delta) = d\Delta(S_i)$. The distributed diagnosis is then globally consistent. □

We propose to build such a distribution of the system, using the junction tree theory.

4 Diagnosis by Junction Tree

4.1 Junction Tree

The concept of the junction tree is borrowed from the field of probabilistic inference where its structure is useful for working in complex domains [Huang and Darwiche, 1996]. Note that *junction trees* are also referred to as *join trees* in the literature [Schumann and Huang, 2008].

Definition 1 (Junction Tree) *Let $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$ be a graph. A* junction tree *for $\mathcal{G}$ is a pair $(\mathcal{T}, \mathcal{C})$, where $\mathcal{T}$ is a tree and $\mathcal{C}$ is a function which maps each node i in tree $\mathcal{T}$ into a label $\mathcal{C}_i$ called a* cluster. *The junction tree must satisfy the following properties:*

1. $\mathcal{C}_i \subseteq \mathcal{V}$, i.e. *each cluster is a set of vertices from $\mathcal{G}$.*
2. *If two vertices are connected in $\mathcal{G}$, they will appear together in some cluster $\mathcal{C}_i$.*
3. *If a vertex appears in two clusters $\mathcal{C}_i$ and $\mathcal{C}_j$, it must also appear in every cluster $\mathcal{C}_h$ on the path connecting vertices i and j in the junction tree. This is known as the* running intersection property.

The separator *of edge i-j in a junction tree is defined as $\mathcal{C}_i \cap \mathcal{C}_j$. The* width *of a junction tree is the size of its largest cluster minus one.*

One of the steps in obtaining a junction tree from a graph is to triangulate the graph, *i.e.*, add extra links such that every cycle of length greater than three has a chord. There are different ways to triangulate a graph, yielding different sets of clusters. Moreover, each triangulated graph may have several different junction trees. It is therefore desirable to have optimal triangulations and optimal junction trees with respect to complexity. As discussed later, the complexity here depends on the size of the clusters: an optimal junction tree minimises the size of the largest cluster. However, the optimality problem for triangulation is NP-complete. Given a triangulated graph, we can obtain an optimal junction tree using an algorithm from [Jensen and Jensen, 1994] which is quadratic in the number of cliques.

The Figure 1 gives an example of three graphs and their junction trees. Note that the ith junction tree is also a junction tree for the jth graph if $i > j$ while it is not true if $i < j$. The best junction tree is the first, as its biggest cluster contains three elements against five for the last; furthermore, the first and second junction tree have the same largest cluster CDG, but the second largest cluster of the first junction tree is smaller than that of the second junction tree.

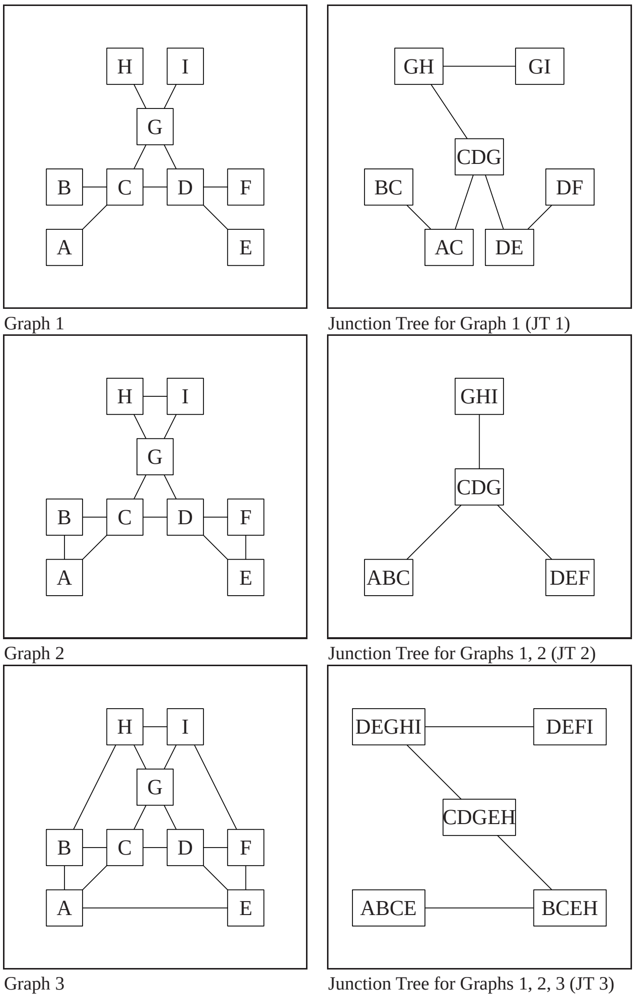

Figure 1: Three graphs and corresponding junction trees

The reasoning behind the use of junction trees in diagnosis is that it could help avoid the need to compute a global diagnosis. Using a junction tree representation of a system has two main advantages [Su and Wonham, 2005]:

1. A tree representation of a system implies that local consistency is equivalent to global consistency.
2. Non-termination issues with local consistency algorithms can be resolved.

4.2 Distribution Algorithm

The junction tree algorithm returns a topology as defined previously, provided it is followed by computation of the edges of the tree itself. Indeed, let $e \in \Sigma_{S_1} \cap \Sigma_{S_2}$ be an event that is shared by subsystems S_i and S_j. We prove that any vertex S in the path between S_i and S_j contains this event ($e \in \Sigma_S$). There are two (possibly identical) components γ_1 and γ_2 such that $\forall i \in \{1, 2\}$, $e \in \Sigma_i$ and $\gamma_i \in \mathcal{S}_i$. Since component γ_1 and γ_2 share an event, they are connected in the original topology and because of the second property of junction trees, there is a cluster S in the junction tree that contains both components ($\{\gamma_1, \gamma_2\} \subseteq S$). By the third property of the junction tree, all clusters between S_i and S contain component γ_i and thus event e. S can be between S_i and S_j or outside, but in both cases there is a path of clusters between S_1 and S_2 that share event e. Thus, the junction tree algorithm returns a tree-shaped distribution. □

Algorithm 2 Distribution using Junction Tree Algorithm

1: **input** $\Gamma, \{\text{Mod}_1, \ldots, \text{Mod}_n\}$
2: $\mathcal{V} := \Gamma$
3: $\mathcal{E} := \{\langle V_i, V_j\rangle \in \mathcal{V}^2 \mid i \neq j \;\&\; \Sigma_i \cap \Sigma_j \neq \emptyset\}$
4: $\mathcal{S} := \{\}$
5: **while** $\mathcal{V} \neq \emptyset$ **do**
6: pick a vertex $V \in \mathcal{V}$
7: $\mathcal{C} := \{V\} \cup \{V' \mid \langle V, V'\rangle \in \mathcal{E}\}$
8: $\mathcal{E} := \mathcal{E} \cup \{\langle V_1, V_2\rangle \mid V_1 \in \mathcal{C}, V_2 \in \mathcal{C}\}$
9: $\mathcal{V} := \mathcal{V} - \{V\}$
10: $\mathcal{E} := \mathcal{E} - \{\langle V_1, V_2\rangle \in \mathcal{E} \mid V_1 = V \vee V_2 = V\}$
11: **if** not ($\exists \mathcal{C}' \in \mathcal{S} \mid \mathcal{C} \subseteq \mathcal{C}'$) **then**
12: $\mathcal{S} := \mathcal{S} \cup \{\mathcal{C}\}$
13: **return** $\mathcal{S}$

We perform distribution by rearranging the topology of the system into a junction tree, as described in Algorithm 2. We first obtain a graph of the original system, $\mathcal{G} = \langle \mathcal{V}, \mathcal{E}\rangle$. Each component γ in the system is a vertex $\mathcal{V}$ on the graph. The edges, $\mathcal{E}$, on the graph represent connected components. We use the junction tree algorithm [Huang and Darwiche, 1996] to obtain the clusters that make up $\mathcal{S}$. We pick a vertex $V \in \mathcal{V}$. A cluster $\mathcal{C}$ is obtained by taking the set formed by V and its neighbours, *i.e.* the vertices on the graph that are connected to V by an edge. We add edges so that all the vertices that make up a cluster are connected. $\mathcal{C}$ is added to $\mathcal{S}$ if it is not a subset of an element of $\mathcal{S}$. We update the original graph by removing V and its associated edges from it. This procedure is repeated until no more vertices are left on the original graph. It is then trivial to calculate the separators that link the clusters into a junction tree.

As mentioned, building an optimal junction tree is NP-complete. However, we can use heuristics in the vertex selection phase of the algorithm (line 6) that would achieve polynomial-time while still producing a high quality tree [Huang and Darwiche, 1996]. One heuristic is to minimise the number of edges added to the graph [Kjrulff, 1990] (line 8 of the algorithm), which then achieves a low-polynomial complexity.

We mentioned in section 3.3 that local consistencies can be performed in any order. However, we can use a strategy, *global propagation* [Huang and Darwiche, 1996], that would only require two ordered series of local consistency computations on the junction tree to achieve global consistency. We consider a message pass from a cluster C_X to its neighbour C_Y to be an operation that makes the components of C_X locally consistent with those of C_Y. By performing these message passes in an ordered manner, we ensure that the consistency introduced by previous message passes is preserved.

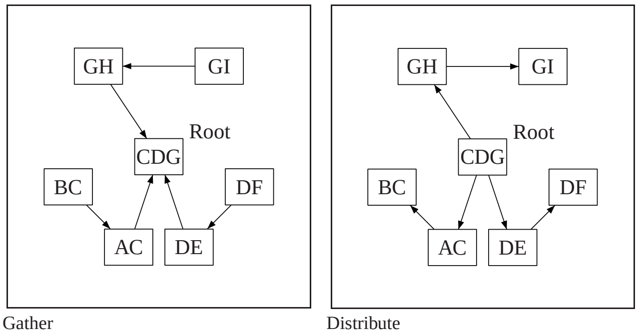

Figure 2: Global Propagation on Junction Tree

We arbitrarily pick a cluster $S_r \in \mathcal{S}$ to be the root of the junction tree. We start from each leaf node and perform local consistency with the neighbour until the root is reached (the *gather* phase). We then perform local consistency in the other direction, from the root back to the leaves (the *distribute* phase). All the clusters are now locally, and consequently globally, consistent with one another. This procedure is illustrated in Figure 2.

4.3 Discussion

Using a junction tree is very interesting as the resulting subsystems tends to be of small size. However, this does not necessarily imply that the local diagnoses will actually be small as we show in the next example.

Consider a tree with n nodes N_1 to N_n. Each node N_i is associated with events e_{i-1} and e_i. The topology of the tree is thus simply a line as node N_i shares event e_i with node N_{i+1}. The automaton of each node N_i is represented in Figure 3. Since the initial state is the same as the final state, the number of occurrences of event e_i is twice that of event e_{i-1} for any i. Consider the node N_1 runs $k \in \mathcal{N}$ loops. Then, event e_0 occurs k times, event e_1 occurs $2 \times k$ times, etc. Event e_i occurs $2^i \times k$ times. The globally consistent automaton representation the behaviour on node N_i must represent the fact that event e_{i-1} occurred $2^{i-1} \times k$ times and the event e_i occurred $2^i \times k$ times for any natural number k (and not for rational non natural numbers). This requires $2^{i-1} + 2^i$ states and transitions. In this example, the number of states after local consistency is exponential in the number of nodes. ■

The result basically comes from the fact that the events e_i and e_j in this example are not concurrent events but they occur in sequence. We expect that most systems actually exhibit concurrent behaviours. In this case, the size of the local diagnosis on a cluster is a direct function of the number of events attached with this cluster, and thus smaller cluster lead to a better efficiency.

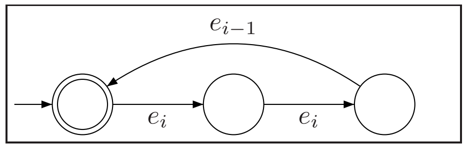

Figure 3: Automaton that models the language of node N_i

The natural topology of the system has an important impact on the quality of the produced junction tree, and hence the size of the subsystems. If we start off with a near tree-like structure, the resulting junction tree will produce smaller size clusters, and hence smaller automata to work with, reducing complexity. *E.g.* in Figure 1, graph 1 produces the best junction tree with smallest clusters (JT 1). With graph 3, because of the larger size clusters, the local diagnoses will actually be quite big (JT 3). The *tree-width* of a graph is the size of the largest cluster in its optimal junction tree minus one. Grids for instance have a tree-width linear in the size of the shortest side. Thus, we emphasize the importance of designing the system in a tree-like structure to make it easier to diagnose.

In [Su and Wonham, 2005], the authors proposed a similar algorithm for the distributed diagnosis of discrete-event systems. A local diagnosis is computed for each component. Then, a given diagnosis is incrementally synchronised with the other diagnoses, which ensures global consistency. After each synchronisation, the events that appear only in components that have already been synchronised can be safely abstracted: the current diagnosis is projected on the relevant events, which reduces the complexity.

This algorithm can be seen as a special case of our approach with three main differences. First, it implicitly builds a *junction line*, since the diagnoses are synchronised in sequence. This restriction potentially increases the width of the junction tree, with a negative impact on the global efficiency.

Second, this algorithm builds a junction tree/line on the graph of *events* rather than the graph of components. This can also be done in our approach. In this case, two events are connected in a graph of events iff they are shared by some component. Since all the events of a given component are interconnected, at least one cluster will contain all these events and will be initialised with the diagnosis of this component (potentially synchronised with the diagnosis of other components). Considering the graph of events leads to clusters with less, or in the worst case as many, events than in the approach presented in this paper, thus reducing complexity.

Finally, a dynamic strategy to choose the order of the synchronisation is proposed in [Su and Wonham, 2005]. In this paper, the junction tree is computed before the local diagnoses are generated. Future works include such a dynamic construction of the junction tree that takes the size of the local diagnoses into account.

5 Conclusion and Future Works

In this article, we identified the importance of a distribution of the system into (possibly overlapping) subsystems for the diagnosis of discrete-event systems. If the distribution generates a tree-shaped topology, an algorithm based on local consistency can ensure global consistency of the diagnosis. We used the graph theory of junction trees to obtain good distributions. The complexity of the diagnosis is then often bounded by the tree width of the system topology which places an upper bound on the number of automata to synchronise together, though counter-examples exist.

We think there is still room for improvement. First, we proposed a static construction of the junction tree based only on the topology of the system. We want to investigate a more flexible technique where the junction tree is built after diagnoses and simple pruning operations are performed locally on components. The idea is that some connections in the system topology can be removed when no communication happened through these connections, leading to a graph with a smaller tree width. Moreover, we could then assign weight on each vertex of the graph. These technique should then improve the efficiency of diagnosis. More generally, we want to investigate more dynamic computations of junction trees: experiments have shown that the connections can often be removed after the distributed diagnosis is computed during the local consistency algorithm. For this reason, we want to start the diagnosis algorithm while the junction tree is being computed so as to dynamically change the construction of the junction tree. This is not trivial as the construction of the junction tree must satisfy some properties.

Regarding system design, an interesting exploration would be to interact with the system designer to propose alternative topology structures in the system in order to ensure a reasonable tree width of the system.

Finally, we considered that the observations emitted by different components were completely independent. However, it is often the case that a (partial) order exists between the observations. *E.g.* the alarm emitted by component 1 was surely emitted before the alarm from component 2. This generates some kind of connection between the two components and potentially interconnect all the components. We want to investigate this issue and determine when these connections can be removed, possibly with an approach based on time slicing [Cordier and Grastien, 2007].

6 Acknowledgements

This research was supported by NICTA in the framework of the SuperCom project. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

[Cassandras and Lafortune, 1999] C. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[Cordier and Grastien, 2007] M.-O. Cordier and A. Grastien. Exploiting independence in a decentralised and incremental approach of diagnosis. In M. Veloso, editor, *Twentieth International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 292–297. AAAI press, 2007.

[Fabre *et al.*, 2005] E. Fabre, A. Benveniste, S. Haar, and Cl. Jard. Distributed monitoring of concurrent and asynchronous systems. *Journal of Discrete Event Systems*, pages 33–84, 2005. special issue.

[Grastien *et al.*, 2007] A. Grastien, Anbulagan, J. Rintanen, and E. Kelareva. Diagnosis of discrete-event systems using satisfiability algorithms. In R. Holte, editor, *Nineteenth National Conference on Artificial Intelligence (AAAI-07)*. AAAI Press, 2007.

[Huang and Darwiche, 1996] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[Jensen and Jensen, 1994] F.V. Jensen and F. Jensen. Optimal junction trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, Seattle, Washington*, 1994.

[Kjrulff, 1990] U. Kjrulff. Triangulation of graphs - algorithms giving small total state space, 1990.

[Lamperti and Zanella, 2003] G. Lamperti and M. Zanella. *Diagnosis of Active Systems*. Kluwer Academic Publishers, 2003.

[Pencolé and Cordier, 2005] Y. Pencolé and M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence (AIJ)*, 164:121–170, 2005.

[Rintanen, 2007] J. Rintanen. Diagnosers and diagnosability of succint transition systems. In M. Veloso, editor, *Proceedings of the 20th Joint Conference on Artificial Intelligence (AAAI-07)*. AAAI Press, 2007.

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

[Schumann and Huang, 2008] A. Schumann and J. Huang. A scalable jointree algorithm for diagnosability. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI-08)*, 2008.

[Schumann *et al.*, 2004] A. Schumann, Y. Pencolé, and S. Thiébaux. Symbolic models for diagnosing discrete-event systems. In *Sixteenth European Conference on Artificial Intelligence (ECAI'04)*, 2004.

[Schumann *et al.*, 2007] A. Schumann, Y. Pencolé, and S. Thiébaux. A spectrum of symbolic on-line diagnosis approaches. In R. Holte, editor, *Nineteenth National Conference on Artificial Intelligence (AAAI-07)*. AAAI Press, 2007.

[Su and Wonham, 2005] R. Su and W. M. Wonham. Global and local consistencies in distributed fault diagnosis for discrete-event systems. *Transactions on Automatic Control*, 50(12):1923–1935, 2005.