A Model of Component Consistency in Distributed Diagnosis

R. Su and **W.M. Wonham** [1]

Abstract. In distributed diagnosis, each local diagnoser first generates a local estimate then communicates with other local diagnosers to reach suitably de£ned consistency. In this paper we propose two types of consistency and provide efficient computational procedures to achieve them.

1 Introduction

So far there is a large body of research on fault diagnosis for discrete-event systems, e.g. finite-state automaton approaches [10] [12], and Petri net approaches [1]. These can be roughly classi£ed as follows: (1) centralized approaches like [10] [16]; (2) decentralized [4] [12]; and (3) distributed [11] [6]. It is well known that centralized approaches may lead to high space complexity in order to store the final diagnoser. Meanwhile, weak scalability and robustness are also features of centralized approaches. Decentralized approaches usually lead to relatively small local diagnosers and improved scalability and robustness. Nevertheless, designing a decentralized diagnoser still requires existence of a centralized target system model at the intermediate stage, which again forces a designer to face the space complexity issue. As for distributed approaches, some methods like [11] [9] [1] adopt the same design procedure as decentralized approaches except that, instead of requiring a global clock during communication as in a decentralized approach, those distributed approaches utilize concurrent communication governed by a set of local clocks. In general, they still have to face the high space complexity introduced by the intermediate centralized system model during the diagnoser design stage.

Recently years have seen "pure" distributed approaches in the sense that: (1) the system model is distributed; (2) diagnosis is locally generated; (3) constraints imposed by interaction among local components are satisfied by achieving a suitable consistency property. Approaches like [3] [6] [13] belong to this category. By introducing inter-component consistency, these approaches can avoid the potentially enormous intermediate system model. So space complexity in the design stage and during online diagnosis can be brought under control. To capture the interaction among local components, there are roughly two types of consistency in the literature: one is *global consistency*, which was more or less described in, e.g. [2] [7] [8] [6] [3]; the other one is *local consistency*, as explained in [13]. By using purely local computation and achieving appropriate inter-diagnoser consistency, we can greatly reduce space complexity of online diagnosis in many practical systems and substantially improve the scalability and robustness of diagnosis. This paper proposes a unified model to describe the different types of consistency and presents efficient computational algorithms to achieve them.

The paper is organized as follows. In Section 2 we present two types of consistency. In Section 3, we introduce the concept of supremal global support and propose an efficient computational procedure to achieve it. Then in Section 4, we introduce the concept of supremal local support and propose a computational procedure to achieve it. We introduce hierarchical computation in Section 5 and draw conclusions in Section 6.

2 Global and Local Consistencies

Let I be an index set. For each $i \in I$ let Σ_i be an event set. As usual [14], we use "||" to represent synchronous product. All natural projections in this paper obey the following rules of notation: given an index set J with a predefined collection of event sets $\{\Sigma_j | j \in J\}$, we define

$$\Sigma_J := \cup_{j \in J} \Sigma_j$$

Given two event sets Σ_i and Σ_j, we use $\mathrm{P}_{i,j}$ to mean the natural projection

$$\mathrm{P}_{i,j} : \Sigma_i^* \to (\Sigma_i \cap \Sigma_j)^*$$

and $\mathrm{P}_{J,i}$ to mean the natural projection

$$\mathrm{P}_{J,i} : \Sigma_J^* \to (\Sigma_J \cap \Sigma_i)^*$$

If we have another index set K with a predefined set $\{\Sigma_k | k \in K\}$, then we use $\mathrm{P}_{J,K}$ to mean the natural projection

$$\mathrm{P}_{J,K} : \Sigma_J^* \to (\Sigma_J \cap \Sigma_K)^*$$

Let $\mathcal{L} = \{\mathrm{L}_i \subseteq \Sigma_i^* | i \in I\}$ be a set of languages. Now we introduce two types of consistency.

Definition 2.1 $\{\mathrm{L}_i | i \in I\}$ is *globally consistent* if for each $i \in I$, $\mathrm{L}_i = \mathrm{P}_{I,i}(||_{j \in I} \mathrm{L}_j)$. □

Here $||_{j \in I} \mathrm{L}_j$ denotes synchronous product of all local components of I. It is well known that synchronous product is one way to combine local transitional behavior to form a global (or composite) transitional behavior. If we treat each local language L_i as a local estimate in diagnosis, then Def. 2.1 says that when a system reaches global consistency, knowing all other local estimates will not improve the current local estimate L_i.

[1] Dept. ECE, University of Toronto,10 King's College Road, Toronto, ON M5S 3G4, Canada email: surong,wonham@control.utoronto.ca, telephone:416-9784124, fax:416-9780804

Definition 2.2 $\{\mathrm{L}_i | i \in I\}$ is *locally consistent* if for each $i, j \in I$, $\mathrm{P}_{i,j}(\mathrm{L}_i) = \mathrm{P}_{j,i}(\mathrm{L}_j)$. □

Again, if we treat each local language L_i as a local estimate in diagnosis, then Def. 2.2 says that when a system reaches local consistency, knowing the local estimate of any local component that is a neighbor of component i will not improve the current local estimate L_i.

From above description we can see that the two types of consistency provide ways to determine whether or not all local components have reached an "agreement". On that basis we can decide when communication among local components should be terminated. The relation between global and local consistencies is described as follows.

Proposition 2.1 If $\mathcal{L}$ is globally consistent then it must be locally consistent. □

The reverse statement need not always hold. For example, let

$$\begin{array}{lll} \Sigma_1 := \{\alpha_1, \alpha_2, \beta_1, \beta_2\} & & \mathrm{L}_1 := \{\alpha_1\beta_1, \alpha_2\beta_2\} \\ \Sigma_2 := \{\beta_1, \beta_2, \gamma_1, \gamma_2\} & & \mathrm{L}_2 := \{\beta_1\gamma_2, \beta_2\gamma_1\} \\ \Sigma_3 := \{\alpha_1, \alpha_2, \gamma_1, \gamma_2\} & & \mathrm{L}_3 := \{\alpha_1\gamma_1, \alpha_2\gamma_2\} \end{array}$$

Then it is not difficult to verify that

$$\begin{array}{rcl} \mathrm{P}_{1,2}(\mathrm{L}_1) & = & \mathrm{P}_{2,1}(\mathrm{L}_2) = \{\beta_1, \beta_2\} \\ \mathrm{P}_{1,3}(\mathrm{L}_1) & = & \mathrm{P}_{3,1}(\mathrm{L}_3) = \{\alpha_1, \alpha_2\} \\ \mathrm{P}_{2,3}(\mathrm{L}_2) & = & \mathrm{P}_{3,2}(\mathrm{L}_3) = \{\gamma_1, \gamma_2\} \end{array}$$

So $\{\mathrm{L}_1, \mathrm{L}_2, \mathrm{L}_3\}$ is locally consistent. But it is not globally consistent because $\mathrm{L}_1 || \mathrm{L}_2 || \mathrm{L}_3 = \varnothing$.

It is interesting to know under what condition(s) global consistency is equivalent to local consistency. To this end, we need more technical preparation.

Given $\{\Sigma_i | i \in I\}$ we define a graph $\mathrm{Gr} =< \mathrm{Ver}, \mathrm{Edg} >$ where $\mathrm{Ver} = I$ and $\mathrm{Edg} \subseteq \mathrm{Ver} \times \mathrm{Ver}$ such that

$$(\forall i, j \in \mathrm{Ver})\, (i, j) \in \mathrm{Edg} \iff i \neq j \,\&\, \Sigma_i \cap \Sigma_j \neq \varnothing$$

Clearly, if $(i, j) \in \mathrm{Edg}$ then so is (j, i). Thus Gr is an undirected graph. Figure 1 depicts such a graph.

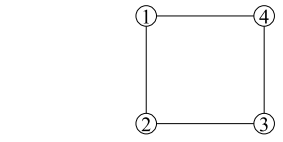

Figure 1. Example of Graph Gr

Proposition 2.2 Given $\mathcal{L} = \{\mathrm{L}_i \subseteq \Sigma_i^* | i \in I\}$, if the corresponding graph Gr is a tree as commonly defined in graph theory [5], then $\mathcal{L}$ is globally consistent if and only if it is locally consistent. □

Proposition 2.1 indicates that local consistency is a necessary condition for global consistency, and Proposition 2.2 indicates that this necessary condition becomes sufficient if the graph Gr is a tree.

3 Supremal Global Support

Given $\mathcal{L} = \{\mathrm{L}_i \subseteq \Sigma_i^* | i \in I\}$, a set $\{\mathrm{E}_i \subseteq \mathrm{L}_i | i \in I\}$ is a *global support* of $\mathcal{L}$ if it is globally consistent, namely

$$(\forall i \in I)\, \mathrm{E}_i = \mathrm{P}_{I,i}(||_{j \in I} \mathrm{E}_j)$$

Let $\Delta(\mathcal{L})$ be the set of all global supports of $\mathcal{L}$. Clearly $\Delta(\mathcal{L})$ is not empty because it contains $\{\mathrm{E}_i = \varnothing | i \in I\}$. We define a binary relation "$\leq$" among elements in $\prod_{i \in I} \mathrm{Pwr}(\Sigma_i^*)$, where $\mathrm{Pwr}(\Sigma_i^*)$ means the power set of Σ_i^*.

$$\{\mathrm{E}_i | i \in I\} \leq \{\tilde{\mathrm{E}}_i | i \in I\} \iff (\forall i \in I)\, \mathrm{E}_i \subseteq \tilde{\mathrm{E}}_i$$

$\mathcal{E} \in \Delta(\mathcal{L})$ is called the *supremal global support* of $\Delta(\mathcal{L})$, written $\mathrm{Sup}\Delta(\mathcal{L})$, if $(\forall \mathcal{E}' \in \Delta(\mathcal{L}))\, \mathcal{E}' \leq \mathcal{E}$.

Proposition 3.1 $\mathrm{Sup}\Delta(\mathcal{L}) = \{\mathrm{P}_{I,i}(||_{j \in I} \mathrm{L}_j) | i \in I\}$. □

It is interesting in practice, e.g. for conservative diagnosis, to know how to compute such a supremal global support. Suppose $I = \{1, 2, \cdots, n\}$ and we want to compute $\mathrm{P}_{I,n}(||_{j \in I} \mathrm{L}_j)$. We first adopt an ordering on nodes in I which makes n the last node in the ordering. For simplicity, suppose the ordering is $[1, 2, \cdots, n]$. The ordering is an important factor for the space complexity of the following computational procedure. Due to limited space, we leave the ordering issue to a full version of this paper, where a heuristic ordering algorithm is given. Now we present our computational procedure, where the notation used below follows the notational rules described in Section 2.1.

Computational Procedure for Global Consistency: (CPGC)

1. Initially set $\mathrm{T}_0 := \Sigma_1$ and $\mathrm{W}_0 := \Sigma_1^*$.
2. For $k = 1, 2, \cdots, n-1$,
   - Set $J_k := \{1, 2, \cdots, k\}$ and $\mathrm{T}_k := \Sigma_{J_k} \cap \Sigma_{I-J_k}$.
   - $\mathrm{W}_k := \mathrm{P}_{\mathrm{T}_{k-1} \cup \Sigma_k, \mathrm{T}_k}(\mathrm{W}_{k-1} || \mathrm{L}_k)$
3. $\mathrm{E}_n := \mathrm{W}_{n-1} || \mathrm{L}_n$ □

Proposition 3.2 Let E_n be as above. Then $\mathrm{E}_n = \mathrm{P}_{I,n}(||_{j \in I} \mathrm{L}_j)$. □

To demonstrate CPGC, consider the following example. The system consists of 18 local components, namely $I = \{1, 2, \cdots, 17, 18\}$, whose transition structures are depicted in the left side of Figure 2. The right picture shows the system's network structure. Suppose we

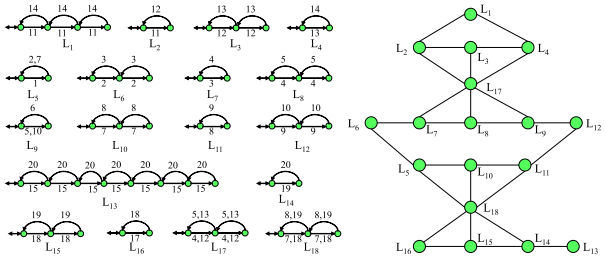

Figure 2. Plant $\mathcal{L}$ (Left) and Graph Gr (Right)

want to compute $\mathrm{E}_{18} = \mathrm{P}_{I,18}(||_{k \in I} \mathrm{L}_k))$. We select an arbitrary ordering, say $[1, 2, 3, \cdots, 17, 18]$. The computational result of CPGC is shown as follows.

Table 3.1

k	L_k	$W_{k-1}\|\|\mathrm{L}_k$	$W_k = \mathrm{P}_{\mathrm{T}_{k-1}\cup\Sigma_k,\mathrm{T}_k}(\mathrm{W}_{k-1}\|\|\mathrm{L}_k)$	Null Events
1	[4,6]	[4,6]	[4,6]	∅
2	[2,2]	[8,13]	[5,8]	11
3	[3,4]	[15,30]	[15,30]	∅
4	[2,2]	[6,9]	[3,4]	14
5	[2,3]	[6,17]	[6,17]	∅
6	[3,4]	[18,60]	[3,10]	1,2
7	[2,2]	[6,20]	[3,10]	3
8	[3,4]	[9,33]	[9,33]	∅
9	[2,3]	[18,78]	[9,42]	6
10	[3,4]	[27,135]	[27,135]	∅
11	[2,2]	[54,279]	[54,279]	∅
12	[3,4]	[162,756]	[27,108]	9,10
13	[8,14]	[216,1242]	[27,135]	15
14	[2,2]	[54,270]	[27,135]	20
15	[3,4]	[81,432]	[81,432]	∅
16	[2,2]	[162,891]	[81,432]	17
17	[3,8]	[54,252]	[9,24]	4,5,12,13
18	[3,8]	[6,12]	-	-

In Table 3.1, the column of "Null Events" represents the set of events that are projected out by $\mathrm{P}_{\mathrm{T}_{k-1}\cup\Sigma_k,\mathrm{T}_k}$ for each $k : 1 \leq k \leq 17$. The entry $[a_1, a_2]$ means that the resultant finite-state automaton has a_1 states and a_2 transitions. From the table we see that our objective E_{18} is realized by an automaton with 6 states and 12 transitions. All results are obtained by using the software CTCT [15]. Columns under the titles $W_{k-1}||\mathrm{L}_k$ and W_k describe the size of all intermediate computational results before we obtain E_{18}. The maximum size of intermediate results is 216 states with 1242 transitions. The brute force computation completely fails in this example because the size of the result $||_{k\in I}\mathrm{L}_k$ is too big to be handled by our software CTCT, which exhausts memory before it finishes the computation. By that time the computation has already reached 10^6 states and 8×10^6 transitions. If we compare $[216, 1242]$ with $[10^6, 8 \times 10^6]$, then clearly CPGC is much more efficient than the brute-force method. The correctness of E_{18} is guaranteed by Prop. 3.1. In order to obtain $\mathrm{Sup}\Delta(\mathcal{L})$, we need to repeat CPGC for each local component $i \in I$. At this point, we can see that CPGC may entail high time complexity if the system consists of many local components. We will see later that hierarchical computation may come to the rescue.

4 Supremal Local Support

4.1 Concept of Supremal Local Support

In the spirit of section 3, we say that a set $\{\mathrm{E}_i \subseteq \mathrm{L}_i | i \in I\}$ is a *local support* of $\mathcal{L}$ if it is locally consistent, namely

$$(\forall i, j \in I)\, \mathrm{P}_{i,j}(\mathrm{E}_i) = \mathrm{P}_{j,i}(\mathrm{E}_j)$$

Let $\Gamma(\mathcal{L})$ be the set of all local supports of $\mathcal{L}$. Clearly $\Gamma(\mathcal{L})$ is not empty because it contains the support $\{\mathrm{E}_i = \varnothing | i \in I\}$. Then we have the following result.

Proposition 4.1 $(\exists \mathcal{E} \in \Gamma(\mathcal{L}))(\forall \mathcal{E}' \in \Gamma(\mathcal{L}))\, \mathcal{E}' \leq \mathcal{E}$ □

We call $\mathcal{E}$ in Prop. 4.1 the *supremal local support* of $\Gamma(\mathcal{L})$, written $\mathrm{Sup}\Gamma(\mathcal{L})$. By Prop. 2.1 we can derive that $\mathrm{Sup}\Delta(\mathcal{L}) \leq \mathrm{Sup}\Gamma(\mathcal{L})$. In distributed diagnosis, $\mathrm{Sup}\Delta(\mathcal{L})$ leads to a tighter estimate than $\mathrm{Sup}\Gamma(\mathcal{L})$ does. But we have two reasons to be interested in $\mathrm{Sup}\Gamma(\mathcal{L})$. First, mathematically it imposes a much more difficult problem than $\mathrm{Sup}\Delta(\mathcal{L})$ does. Prop. 3.1 defines a concrete map from the initial set $\mathcal{L}$ to $\mathrm{Sup}\Delta(\mathcal{L})$. But we do not have a similar map between $\mathcal{L}$ and $\mathrm{Sup}\Gamma(\mathcal{L})$. Second, from a computational point of view, we will see that we can use a rather *scalable* computational procedure to compute $\mathrm{Sup}\Gamma(\mathcal{L})$, as long as the procedure finitely terminates. But to find a similar scalable procedure for $\mathrm{Sup}\Delta(\mathcal{L})$ is still an open question. In distributed diagnosis, by computing $\mathrm{Sup}\Gamma(\mathcal{L})$ we can find a tradeoff between quality of diagnosis and scalability and robustness of the diagnoser. Next, we introduce a computational procedure for the supremal local support.

4.2 Computing Supremal Local Support

In [13] the authors proposed a scalable computational algorithm that can be used here to achieve the supremal local support defined above. A similar algorithm appeared later in [3] in the form of a Petri net computation. In this paper we revise our algorithm by imposing a binary relation on the nodes of the target network.

Given $\mathcal{L}$, let $\mathrm{Gr} = (\mathrm{Ver}, \mathrm{Edg})$ be the associated graph. Suppose we pick one node of Ver as the *root* of Gr. Without loss of generality, suppose the root is node 1. From graph theory [5], a *simple path* between two different nodes $i, j \in \mathrm{Ver}$ is a sequence of edges

$$(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k) \in \mathrm{Edg}$$

such that

$$v_0 = i \,\&\, v_k = j \,\&\, (\forall l, r : 0 \leq l, r \leq k)\, l \neq r \Rightarrow v_l \neq v_r$$

The *length* of such a simple path is k. We define the distance between the root node 1 and a node $v \in \mathrm{Ver}$, written $d(1, v)$, as the *length* of the shortest *path* between v and node 1. We assume that **the graph Gr is connected**. Let $\phi : \mathrm{Ver} \to \{1, \cdots, |\mathrm{Ver}|\}$ be a one-to-one map such that

$$(\forall (i, j) \in \mathrm{Edg})\, d(1, i) < d(1, j) \Rightarrow \phi(i) < \phi(j)$$

In general there may exist more than one choice for ϕ. Fix one choice of ϕ and define the following binary relation among vertices:

$$(\forall i, j \in \mathrm{Ver})\, i \downarrow j \iff (i, j) \in \mathrm{Edg} \,\&\, \phi(i) < \phi(j)$$

We call i a *father node* of j and the binary relation $\downarrow$ the *father-son* relation. With the proposed father-son relation $\downarrow$, we can partition Ver of Gr as follows.

Proposition 4.2 Let $\mathrm{Gr} = (\mathrm{Ver}, \mathrm{Edg})$ be a graph as described above, where $\mathrm{Ver} = I$. Suppose a father-son relation $\downarrow$ among nodes in Ver is given. Then there exists a partition $\{V_0, V_1, \cdots, V_k\}$ on Ver such that for each set V_i $(0 \leq i \leq k)$ the following two conditions hold,

1. $(\forall j \in V_i)\, \{r \in I | r \downarrow j\} \subseteq \cup_{m=0}^{i-1} V_m$
2. $(\forall j \in V_i)\, \{r \in I | j \downarrow r\} \subseteq \cup_{m=i+1}^{k} V_m$

where $\cup_{m=0}^{-1} V_m := \varnothing$ and $\cup_{m=k+1}^{k} V_m := \varnothing$. □

With such a partition we present the following computational procedure to compute the supremal local support $\mathrm{Sup}\Gamma(\mathcal{L})$.

Computational Procedure for Local Consistency: (CPLC)

1. (Initialization) $(\forall i \in I)\, \mathrm{E}_i^0 := \mathrm{L}_i$
2. (Centripetal Propagation) At odd round $n \in 2\mathbb{N}+1$, starting from V_k and ending at V_0, for each node $i \in V_j$ $(k \geq j \geq 0)$ compute

$$\mathrm{E}_i^n := \begin{cases} \mathrm{E}_i^{n-1} & \text{if } (\nexists r \in I)\, i \downarrow r \\ \mathrm{E}_i^{n-1} \Big|\Big| (||_{r:i\downarrow r} \mathrm{P}_{r,i}(\mathrm{E}_r^n)) & \text{otherwise} \end{cases}$$

3. (Centrifugal Propagation) At even round $n \in 2\mathbb{N}^+$, starting from V_0 and ending at V_k, for each node $i \in V_j$ $(0 \leq j \leq k)$ compute

$$\mathrm{E}_i^n := \begin{cases} \mathrm{E}_i^{n-1} & \text{if } (\nexists r \in I)\, r \downarrow i \\ \mathrm{E}_i^{n-1} \Big|\Big| (||_{r:r\downarrow i} \mathrm{P}_{r,i}(\mathrm{E}_r^n)) & \text{otherwise} \end{cases}$$

4. **Termination Condition:** $(\exists n \in \mathbb{N}^+)(\forall i \in I)\, \mathrm{E}_i^n = \mathrm{E}_i^{n+1}$ □

Proposition 4.2 guarantees that in step 2 before we compute E_i^n all those E_r^n with $i \downarrow r$ have been computed already. Similarly in step 3 before we compute E_i^n all those E_r^n with $r \downarrow i$ have been computed. Therefore CPLC is well defined. The main feature of CPLC is its *scalability*, in the sense that when we add a new node to the graph or remove one from the graph, only a few neighboring nodes need to update their local communication protocol, which is usually done by simply changing their list of father nodes and son nodes. As a comparison, in CPGC when we add a new node or remove one, we need to redesign the ordering among nodes. So CPGC is not scalable.

Figure 3 depicts how the CPLC procedure works in a simple example. In this example, the father-son relation is defined as follows:

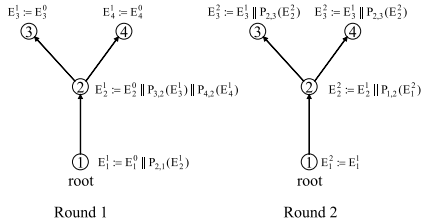

Figure 3. 2-Round CPLC

$$1 \downarrow 2 \text{ and } 2 \downarrow 3 \text{ and } 2 \downarrow 4$$

So the partition is $\{V_0 = \{1\}, V_1 = \{2\}, V_2 = \{3, 4\}\}$. We can see that during the computational process, in odd-number rounds computation starts from nodes in V_2, then the node in V_1 and finally the node in V_0. In even-number rounds computation goes in the opposite direction, namely starts from the node in V_0, then the node in V_1 and finally nodes in V_2. So the computational process is well organized.

Proposition 4.3 If CPLC terminates at $n \in \mathbb{N}^+$, then we have that $\{\mathrm{E}_i^n | i \in I\} = \mathrm{Sup}\Gamma(\mathcal{L})$. □

So far we have shown in Prop. 4.3 that as long as CPLC finitely terminates, the result is the supremal local support $\mathrm{Sup}\Gamma(\mathcal{L})$. Nevertheless, it turns out that CPLC may not always terminate. So next we briefly discuss termination issue.

4.3 Termination of CPLC

It has been pointed out in [3] that the original algorithm may not always terminate. This is also true of CPLC. It is natural to seek conditions for termination. We have the following results.

Proposition 4.4 If for each $i \in I$, L_i contains a finite number of strings, then CPLC terminates. □

Proposition 4.5 If Gr is a tree then the termination condition holds at the end of round 2. □

The advantage of imposing the father-son relation is demonstrated in Prop. 4.5, namely we know the maximum number of rounds that we have to wait before CPLC terminates in a tree. In another paper on CPLC, we show that the existence of a similar upper bound of rounds in a graph called *simple net* which contains disjoint loops. Such information is very helpful for online diagnosis because we can predict when to report diagnostic result at the latest. The original algorithm proposed in [13] did not have this feature.

So far we have introduced concepts of supremal global support and supremal local support, and provided concrete computational procedures. Nevertheless, we have seen that CPLC may not always terminate and CPGC can only compute the supremal global support in a sequential way, which may result in high time complexity. So next we introduce hierarchical computation, which can partially reduce high time complexity of CPGC.

5 Hierarchical Computation

5.1 Concept of Hierarchical Computation

First, we describe how to construct a two-level hierarchy. Given the index set I, let $\hat{I}$ be a second index set based on a partition $\{J_{\hat{k}} \neq \varnothing | \hat{k} \in \hat{I}\}$ of I. From now on we use $i, j, k, \cdots$ for elements of I, and $\hat{i}, \hat{j}, \hat{k}, \cdots$ for elements of $\hat{I}$. Similarly, we use $J, K, \cdots$ for subsets of I and $\hat{J}, \hat{K}, \cdots$ for subsets of $\hat{I}$. For each $\hat{k} \in \hat{I}$, recall that by the notational rules we have

$$\Sigma_{J_{\hat{k}}} := \cup_{i \in J_{\hat{k}}} \Sigma_i \text{ and } \Sigma_{I-J_{\hat{k}}} := \cup_{j \in I - J_{\hat{k}}} \Sigma_j$$

We define a new event set $\Sigma_{\hat{k}} \subseteq \Sigma_{J_{\hat{k}}}$ such that

$$\Sigma_{J_{\hat{k}}} \cap \Sigma_{I-J_{\hat{k}}} \subseteq \Sigma_{\hat{k}} \tag{1}$$

In other words, the new event set $\Sigma_{\hat{k}}$ should contain all events that are shared by $\{\mathrm{L}_i | i \in J_{\hat{k}}\}$ and $\{\mathrm{L}_j | j \in I - J_{\hat{k}}\}$. In fault diagnosis, besides those shared events, $\Sigma_{\hat{k}}$ usually contains only observable events which represent observation in the system.

As an illustration, let $I = \{1, 2, 3, 4\}$ and suppose $\Sigma_1 = \{\alpha, \beta\}$, $\Sigma_2 = \{\beta, \gamma\}$, $\Sigma_3 = \{\sigma, \gamma, \delta\}$ and $\Sigma_4 = \{\alpha, \delta\}$. Suppose we partition I into two subsets $J_a = \{1, 2\}$ and $J_b = \{3, 4\}$ with $\hat{I} = \{a, b\}$. Then

$$\Sigma_{J_a} = \{\alpha, \beta, \gamma\} \text{ and } \Sigma_{J_b} = \{\alpha, \gamma, \delta, \sigma\}$$

So $\Sigma_{J_a} \cap \Sigma_{I-J_a} = \Sigma_{J_a} \cap \Sigma_{J_b} = \{\alpha, \gamma\}$. To define a new set $\Sigma_a \subseteq \Sigma_{J_a}$ such that (1) holds, we can pick $\Sigma_a = \{\alpha, \gamma\}$ or $\Sigma_a = \{\alpha, \gamma, \beta\}$.

With the same notational rules for natural projections, for each $\hat{k} \in \hat{I}$ we define a new language on $\Sigma_{\hat{k}}^*$ as follows:

$$\mathrm{L}_{\hat{k}} := \mathrm{P}_{J_{\hat{k}}, \hat{k}}(||_{i \in J_{\hat{k}}} \mathrm{L}_i) \tag{2}$$

Thus we have a new set of languages $\hat{\mathcal{L}} = \{\mathrm{L}_{\hat{k}} | \hat{k} \in \hat{I}\}$ which serves as the abstraction of $\mathcal{L}$. Each element of $\hat{\mathcal{L}}$ is called an *abstract local module* of $\mathcal{L}$. The relation between $\mathcal{L}$ and $\hat{\mathcal{L}}$ can be described as follows:

$$\begin{aligned} \mathrm{P}_{I,\hat{I}}(||_{i \in I} \mathrm{L}_i) &= \mathrm{P}_{I,\hat{I}}(||_{\hat{k} \in \hat{I}}(||_{i \in J_{\hat{k}}} \mathrm{L}_i)) \\ &= ||_{\hat{k} \in \hat{I}} \mathrm{P}_{J_{\hat{k}}, \hat{k}}(||_{i \in J_{\hat{k}}} \mathrm{L}_i) \\ &= ||_{\hat{k} \in \hat{I}} \mathrm{L}_{\hat{k}} \end{aligned}$$

Thus we finally have

$$\mathrm{P}_{I,\hat{I}}(||_{i \in I} \mathrm{L}_i) = ||_{\hat{k} \in \hat{I}} \mathrm{L}_{\hat{k}} \tag{3}$$

We can define the supremal global support of $\Delta(\hat{\mathcal{L}})$ in the same way as for $\Delta(\mathcal{L})$. Then by Prop. 3.1 we have

$$\{\mathrm{E}_{\hat{i}} := \mathrm{P}_{\hat{I},\hat{i}}(||_{\hat{k}\in\hat{I}}\mathrm{L}_{\hat{k}})|\hat{i}\in\hat{I}\} = \mathrm{Sup}\Delta(\hat{\mathcal{L}})$$

From (3) we can derive the relation between $\mathrm{Sup}\Delta(\mathcal{L}) = \{\mathrm{E}_j|j \in I\}$ and $\mathrm{Sup}\Delta(\hat{\mathcal{L}}) = \{\mathrm{E}_{\hat{i}}|\hat{i}\in\hat{I}\}$ as follows:

$$(\forall\hat{i}\in\hat{I})\,\mathrm{E}_{\hat{i}} = \mathrm{P}_{I,\hat{i}}(||_{j\in I}\mathrm{E}_j) \tag{4}$$

Let

$$\mathbb{W} := \prod_{i\in I}\mathrm{Pwr}(\Sigma_i^*) \text{ and } \hat{\mathbb{W}} := \prod_{\hat{k}\in\hat{I}}\mathrm{Pwr}(\Sigma_{\hat{k}}^*)$$

Everything we have done so far in this section can be summarized in Figure 4, where the maps in the diagram are defined as follows:

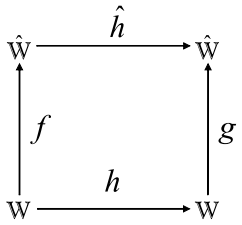

Figure 4. Two-Level Hierarchy

1. $f : \mathbb{W} \to \hat{\mathbb{W}}$,

$$f(\{\mathrm{L}_i|i\in I\}) := \{\mathrm{L}_{\hat{k}} = \mathrm{P}_{J_{\hat{k}},\hat{k}}(||_{j\in J_{\hat{k}}}\mathrm{L}_j)|\hat{k}\in\hat{I}\}$$

2. $g : \mathbb{W} \to \hat{\mathbb{W}}$,

$$g(\{\mathrm{E}_i|i\in I\}) := \{\mathrm{E}_{\hat{k}} = \mathrm{P}_{I,\hat{k}}(||_{j\in I}\mathrm{E}_j)|\hat{k}\in\hat{I}\}$$

3. $h : \mathbb{W} \to \mathbb{W} : \mathcal{L} \mapsto h(\mathcal{L}) := \mathrm{Sup}\Delta(\mathcal{L})$.
4. $\hat{h} : \hat{\mathbb{W}} \to \hat{\mathbb{W}} : \hat{\mathcal{L}} \mapsto \hat{h}(\hat{\mathcal{L}}) := \mathrm{Sup}\Delta(\hat{\mathcal{L}})$.

By expressions (2) and (4) we can see that the above diagram commutes, which tells us how to obtain the high-level structure from the one at low-level. Nevertheless, it is more interesting in practice to know how to obtain the supremal global support $\{\mathrm{E}_i|i\in I\}$ of $\Delta(\mathcal{L})$ at the low level if the supremal global support $\{\mathrm{E}_{\hat{k}}|\hat{k}\in\hat{I}\}$ of $\Delta(\hat{\mathcal{L}})$ at high level is given. To this end we have the following result.

Proposition 5.1 Let $\mathcal{L} = \{\mathrm{L}_i|i\in I\} \in \mathbb{W}$, $\hat{\mathcal{L}} = \{\mathrm{L}_{\hat{k}}|\hat{k}\in\hat{I}\} \in \hat{\mathbb{W}}$. Let f be defined as in Figure 4. Suppose $f(\mathcal{L}) = \hat{\mathcal{L}}$. If we have $\{\mathrm{E}_i|i\in I\} = \mathrm{Sup}\Delta(\mathcal{L})$ and $\{\mathrm{E}_{\hat{k}}|\hat{k}\in\hat{I}\} = \mathrm{Sup}\Delta(\hat{\mathcal{L}})$, then we get that $(\forall\hat{k}\in\hat{I})(\forall i\in J_{\hat{k}})\,\mathrm{E}_i = \mathrm{P}_{J_{\hat{k}},i}((||_{j\in J_{\hat{k}}}\mathrm{L}_j)||\mathrm{E}_{\hat{k}})$. □

By Prop. 3.1 we have

$$\mathrm{E}_i = \mathrm{P}_{I,i}(||_{j\in I}\mathrm{L}_i)$$

Since computation is over I, if the size of I is large, then the time complexity may be a problem. Nevertheless, Proposition 5.1 tells us that if we can achieve the supremal global support $\mathrm{E}_{\hat{k}}$ at the high level, where $\hat{k}\in\hat{I}$ and $i\in J_{\hat{k}}$, then we only need the module $\{\mathrm{L}_j|j\in J_{\hat{k}}\}$ plus $\{\mathrm{E}_{\hat{k}}\}$ to achieve E_i. Notice that $J_{\hat{k}}$ is usually much smaller than I in a large scale system, and furthermore, concurrent computation can be applied on different local modules which are disjoint from each other. Both factors contribute to low time complexity of hierarchical computation. This is the main advantage gained from hierarchical computation. Due to the nature of abstraction, achieving the supremal global support in the abstract model at the high level is relatively easy. Prop. 5.1 has crucial application in multi-resolution diagnosis. It is not difficult to see that the current theory can be extended to a hierarchy with more levels such that, between every two adjacent levels we have a commutative diagram as Figure 4 and Proposition 5.1 holds. To demonstrate the hierarchical computational procedure, consider the following example.

5.2 Example of Hierarchical Computation

Suppose we have four languages $\mathrm{L}_1, \cdots, \mathrm{L}_4$ which are depicted in Figure 5 with the corresponding graph Gr. The index set $I = \{1,2,3,4\}$. Suppose we partition I into two sets $J_a = \{1,2\}$ and

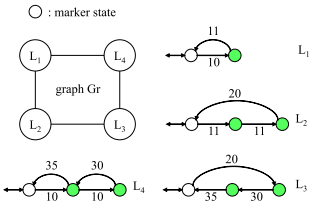

Figure 5. Plant Model and Network Graph

$J_b = \{3,4\}$, where $\hat{I} = \{a,b\}$. Suppose we assign new event sets at the high level based on expression (1) as follows:

$$\Sigma_a := \{10,20\} \text{ and } \Sigma_b := \{10,20,35\}$$

Then we can compute abstract local modules based on expression (2) as follows:

$$\mathrm{L}_a := \mathrm{P}_{J_a,a}(\mathrm{L}_1||\mathrm{L}_2) \text{ and } \mathrm{L}_b := \mathrm{P}_{J_b,b}(\mathrm{L}_3||\mathrm{L}_4)$$

where the results are shown in Figure 6. After that we can compute

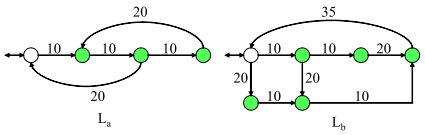

Figure 6. Construction of $\hat{\mathcal{L}} = \{\mathrm{L}_a, \mathrm{L}_b\}$

the supremal global support at the high level according to

$$\mathrm{E}_a := \mathrm{P}_{\hat{I},a}(\mathrm{L}_a||\mathrm{L}_b) \text{ and } \mathrm{E}_b := \mathrm{P}_{\hat{I},b}(\mathrm{L}_a||\mathrm{L}_b)$$

where the results are depicted in Figure 7. After we get E_a and E_b we can compute the supremal global support at the bottom level. Suppose we want to compute E_1 and E_3, which are shown as follows:

$$\mathrm{E}_1 = \mathrm{P}_{J_a,1}(\mathrm{L}_1||\mathrm{L}_2||\mathrm{E}_a) \text{ and } \mathrm{E}_3 = \mathrm{P}_{J_b,3}(\mathrm{L}_3||\mathrm{L}_4||\mathrm{E}_b)$$

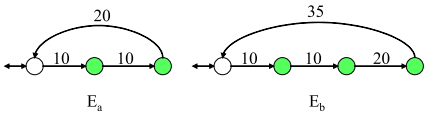

Figure 7. $\mathrm{Sup}\Delta(\hat{\mathcal{L}}) = \{\mathrm{E}_a, \mathrm{E}_b\}$

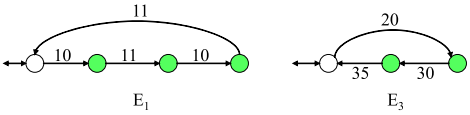

Figure 8. Results of E_1 and E_3

The results are depicted in Figure 8. Notice that E_1 and E_3 can be computed simultaneously because they belong to different modules. The results are verified to be isomorphic to the ones obtained by the brute-force method, namely

$$\mathrm{E}_1 := \mathrm{P}_{I,1}(\mathrm{L}_1||\cdots||\mathrm{L}_4) \text{ and } \mathrm{E}_3 := \mathrm{P}_{I,3}(\mathrm{L}_1||\cdots||\mathrm{L}_4)$$

The computation demonstrated above is realized by the software CTCT [15].

6 Conclusions

In this paper we introduced two main concepts: the supremal global support and the supremal local support. In distributed diagnosis each type of supremal support can be used to model consistency among local diagnoses. Since global consistency is stronger than local consistency, applying the supremal global support in diagnosis usually results in a tighter estimate about fault status of a target system than applying the supremal local support does. Nevertheless, we can see that the computational procedure CPLC for achieving the supremal local support is much more scalable than the computational procedure CPGC for achieving the supremal global support, as long as CPLC terminates. At this point, the supremal local support, which only considers neighborhood information of each local component, offers a tradeoff between diagnosis quality, in terms of the number of fault candidates contained in each local diagnosis, and scalability and robustness of the distributed diagnoser. To the best of our knowledge, CPLC first appeared in [13] then in [3] a similar algorithm was given in the form of a Petri net computation. In this paper we improved it further by imposing the father-son relation $\downarrow$ among nodes. With this improved CPLC, we can tell in the worst case, how many rounds a user has to wait before termination, if indeed it terminates. Nevertheless, we admit that with a general network graph, the decidability of the termination of CPLC is still unsolved. As for CPGC, from the example we can see that it is superior to the brute-force method as far as space complexity is concerned. Nevertheless, since it is a purely sequential computational procedure, if a system consists of many local components, CPGC may take a long time to compute the supremal global support. To reduce time complexity of CPGC, we introduced hierarchical computation, which achieves the supremal global support at the high level first, then descends to the low level to achieve the supremal global support at the low level. CPGC can be used at each single level during hierarchical computation. By Prop. 5.1 we can see that when computation comes to the low level, concurrent computation can be done in each disjoint local abstract module. This is another application of divide-and-conquer. In online diagnosis, this may substantially reduce time complexity. Nevertheless, we admit that we have to pay the price of using extra memory to store the high-level abstract model. A rigorous space complexity analysis remains to be carried out. But a brief informal analysis will be given in the full version paper. Our experience shows that, if we consider both static memory to store the hierarchical distributed model and dynamic memory to store the intermediate computational results, the space complexity of the hierarchical method is no worse, and in many cases much better, than the non-hierarchical method.

REFERENCES

[1] A. Aghasaryaiu, E. Fabre, A. Benveniste, R. Boubour, and C. Jard, 'A petri net approach to fault detection and diagnosis in distributed systems', in *Proc. 1997 IEEE Conference on Decision and Control (CDC'97)*, pp. 720–725, December 1997.

[2] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella, 'Diagnosis of large active systems', *Arti£cial Intelligence*, **110**(1), 135–183, May 1999.

[3] A. Benveniste, S. Haar, E. Fabre, and C. Jard, 'Distributed monitoring of concurrent and asynchronous systems', in *Proc. ATPN-Workshop on Discrete Event Systems Control*, pp. 97–142, Eindhoven, The Netherlands, June 2003.

[4] R. Debouk, S. Lafortune, and D. Teneketzis, 'Coordinated decentralized protocols for failure diagnosis of discrete event systems', *Discrete Event Dynamic Systems: Theory and Applications*, **10**(1/2), 33–86, January 2000.

[5] Reinhard Diestel, *Graph Theory*, Springer-Verlag, New York, 2nd edn., 2000.

[6] E. Fabre, A. Benveniste, and C. Jard, 'Distributed diagnosis for large discrete event dynamic systems', in *Proc. 15th IFAC World Congress*, Barcelona, Spain, July 2002.

[7] Gianfranco Lamperti, Marina Zanella, and Paolo Pogliano, 'Diagnosis of active systems by automata-based reasoning techniques', *Arti£cial Intelligence*, **12**(3), 217–237, May 2000.

[8] Yannick Pencol, Marie-Odile Cordier, and Laurence Roz, 'Incremental decentralized diagnosis approach for the supervision of a telecommunication network', in *Proc. 2002 IEEE Conference on Decision and Control (CDC'02)*, pp. 435–440, Las Vegas, USA, December 2002.

[9] Gregory Provan, 'A model-based diagnosis framework for distributed systems', in *Proc. of International Workshop on Principles of Diagnosis (DX'02)*, pp. 16–25, Austria, May 2002.

[10] M. Sampath, S. Lafortune, and D. Teneketzis, 'Active diagnosis of discrete-event systems', *IEEE Transactions on Automatic Control*, **40**, 908–929, July 1998.

[11] R. Sengupta, 'Diagnosis and communication in distributed systems', in *Proc. International Workshop on Discrete Event Systems (WODES'98)*, pp. 144–151, IEE, London, August 1998.

[12] R. Su and W.M. Wonham, 'Decentralized fault diagnosis for discrete-event systems', in *Proc. 2000 CISS*, pp. TP1:1–6, Princeton, New Jersey, March 2000.

[13] R. Su, W.M. Wonham, J. Kurien, and X. Koutsoukos, 'Distributed diagnosis for qualitative systems', in *Proc. 6th International Workshop on Discrete Event Systems (WODES'02)*, pp. 169–174, Zaragoza, Spain, October 2002.

[14] W. M. Wonham, *Notes on Control of Discrete-Event Systems: ECE 1636F/1637S 2003-2004*, Systems Control Group, Dept. ECE, University of Toronto. URL: www.control.utoronto.ca/DES.html, 2003.

[15] W.M. Wonham, *Software Package for Supervisory Control*, Systems Control Group, Dept. of ECE, University of Toronto. URL: www.control.utoronto.ca/DES.html, 2003.

[16] S. Hashtrudi Zad, R.H. Kwong, and W.M. Wonham, 'Fault diagnosis in discrete-event systems', in *Proc. 1998 IEEE Conference on Decision and Control (CDC'98)*, pp. 3769–3774, Tampa, Florida, USA, December 1998.