DESIGN OF A DISTRIBUTED DIAGNOSIS SYSTEM

Eva Hudlická, Victor Lesser, Jasmina Pavlin,
and Anil Rewari

COINS Technical Report 86-63
December 1986

ABSTRACT

We describe an architecture of a Distributed Diagnosis Module (DDM) designed to work in conjunction with a distributed problem-solving system to diagnose inappropriate system behavior. The problem-solving system being diagnosed is the Distributed Vehicle Monitoring Testbed (DVMT), an interpretation system based on the Hearsay-II architecture. The work described here is the continuation of our work on a centralized Diagnosis Module (DM), which has been implemented and is used to identify faults in the DVMT, such as sensor failures or inappropriate control parameter settings. In the centralized configuration, we assume that one diagnostic node is responsible for diagnosing the symptoms at all problem-solving nodes. We briefly describe the architecture of the centralized DM and illustrate its diagnosis via causal pathway construction. We then focus on the issues in distributing the centralized DM and on the changes necessary to make the DM function in a distributed environment; primarily more flexible local control. We discuss cases where the diagnosis results in two or more fault candidates and evidential reasoning, through interaction with other diagnosis modules, is necessary to disambiguate these. The evidence necessary for this comes from redundancy in the DVMT system. We propose an architecture where each problem-solving node has its own Diagnosis Module and these modules then interact in constructing the causal pathways necessary to explain the inappropriate DVMT behavior. We conclude with an example of a distributed diagnostic session which illustrates how the individual Diagnosis Modules interact to cooperatively solve the diagnosis problem.

This research was sponsored, in part, by GTE under Contract GTE# 851024A, by the National Science Foundation under Grants NSF# DCR-8500332 and NSF# DCR-8318776 and by the Defense Advanced Research Projects Agency (DOD) monitored by the Office of Naval Research under Contract N00014-79-C-0439.

Contents

List of Figures

1. Introduction

An important issue in the design of the next generation of large, complex, and autonomous distributed systems is how to make these systems robust in the face of dynamically changing task and hardware characteristics. Our approach to achieving this robustness is not to go to a replicated component approach, but rather to have the system be able to reorganize itself (organization self-design) based on the changed task characteristics and the available hardware resources. This capability for organization self-design requires the system to be able to detect symptoms which indicate that current assumptions about task and hardware characteristics are invalid and then to diagnose these symptoms to ascertain their causes. (See Figure 1).

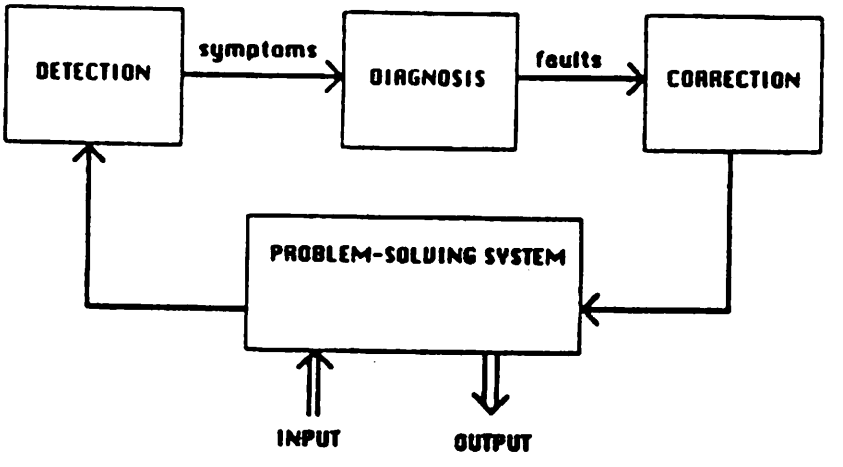


Figure 1: Architecture of a Fully Fault Tolerant Problem–Solving System
A fully fault tolerant problem–solving system would contain a detection, a diagnosis, and a correction component. The detection component would monitor the problem–solving system behavior and would detect any deviations from expected behavior. The diagnosis component would receive reports of such misbehaviors and would identify the faults which caused them. The correction component would correct the faults in the problem–solving system.

In our previous work on the Distributed Vehicle Monitoring Testbed (DVMT) we have developed such a diagnosis component that attempts to locate problem–solving control failures and certain hardware failures. The aim of the Diagnosis Module (DM) is to explain some observed inappropriate behavior of the DVMT in terms of "primitive faults". A primitive fault is either a hardware failure, such as a sensor or a communication channel

failure, or, more often, an inappropriate setting of one of the numerous control parameters (control parameters relate not to task characteristics but organizational design) in the DVMT. The diagnosis is based on a causal model of the DVMT system's possible behaviors. The DM diagnoses the DVMT problems by constructing a causal pathway that links the observed symptom with one or more primitive faults. The causal pathway is constructed by determining how the system reached each state in the chain of events that lead up to the symptom. The reconstruction of the DVMT behavior is done by examining the DVMT data structures where the intermediate results are stored.

Since the initial emphasis was on first constructing the causal diagnostic model, the system complexity was reduced by building it as a **centralized diagnosis** model. The DM resides in a separate node (see Figure 2) and has access to all the DVMT nodes' data structures. Thus a single diagnostic node is responsible for diagnosing all DVMT nodes. Such centralized organization is simple but requires much communication since all necessary data must be transmitted back and forth between the individual DVMT nodes and the diagnostic node. A centralized organization is also subject to a catastrophic failure; when the single diagnostic node fails the system is left with no diagnostic capabilities.

For these reasons we would like to explore an alternative system organization for the diagnosis process, one which would be integrated naturally with the underlying distributed problem-solving system. In a **distributed diagnosis** system each DVMT problem-solving node (PSN) would have its own Diagnosis Module which would be responsible for diagnosing only the behavior of that PSN. Any time the local diagnosis would require information about another node's behavior, for example in determining why a node did not receive some expected message from another node, the Diagnosis Module would have to request this information from the DM at the other PSN. Such requests would be done in the form of sending a symptom to the other node and waiting for a reply that would consist of the result of diagnosing that symptom. The system organization is shown in Figure 3.

Distributing the Centralized DM In this paper we focus on the changes that must be made to the centralized DM to enable it to function in a distributed setting. These changes fall into three major categories:

1. *Communication policies among the individual Diagnosis Modules.* Such policies define the organizational structure of the Distributed Diagnosis System.

2. *Flexible Local Control.* In order for the individual Diagnosis Modules to attend to diagnosis requests from other nodes, as well as to assimilate information arriving from these nodes, the local diagnosis control must be made more flexible. The DM must be able to suspend its current processing and resume it at a later time. It must

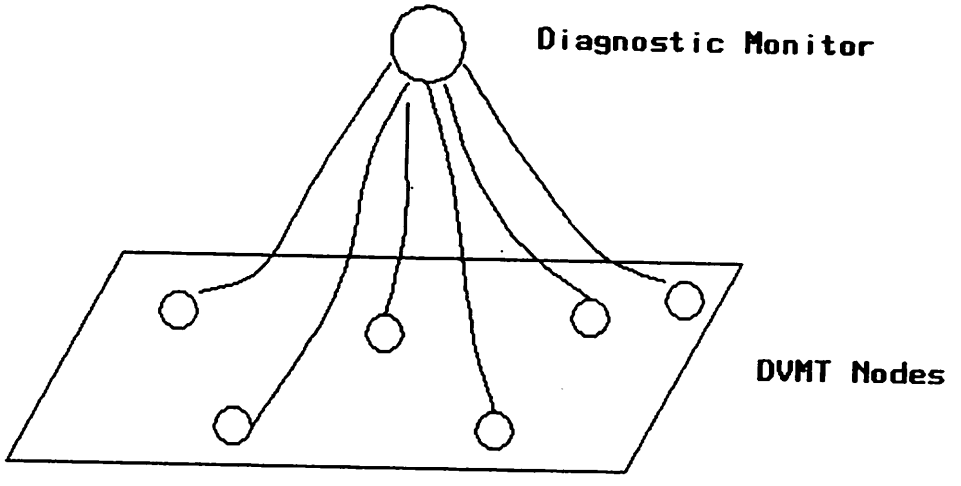


Figure 2: A Centralized Diagnosis Module
In a centralized approach to diagnosis, the Diagnosis Module occupies a single node. This node is separate from the nodes of the underlying distributed problem-solving system. The DM has access to all problem-solving nodes' data structures and it is from these data structures that the past problem-solving behavior is reconstructed. In a centralized organization much communication is required to transmit the necessary data among the individual DVMT nodes and the single DM. This type of organization is also subject to catastrophic failures; when the single diagnostic node fails, the system looses its entire diagnostic capabilities.

also be able to rate the different possible causal pathways and at any time pursue the most promising one.

3. *Resolving ambiguous situations* based on combining evidence from multiple nodes.

Since it is impossible to predict fixed control and communication strategies among the distributed diagnostic nodes, the Distributed Diagnosis System must be highly parameterized to allow for empirical adjustment of both local control and the internode communication policies.

Before we begin, let us make clear the assumptions we make about the DVMT and the Diagnosis Modules: A diagnostic node is assumed to work correctly, the model of the DVMT system is assumed to be correct (if possibly incomplete), and the channels transmitting the diagnostic messages are assumed to be error-free. As a result, we need not worry about diagnosing the diagnostic system. However, in the future, we could apply the same diagnostic techniques to the Diagnosis Module as we are applying to the DVMT

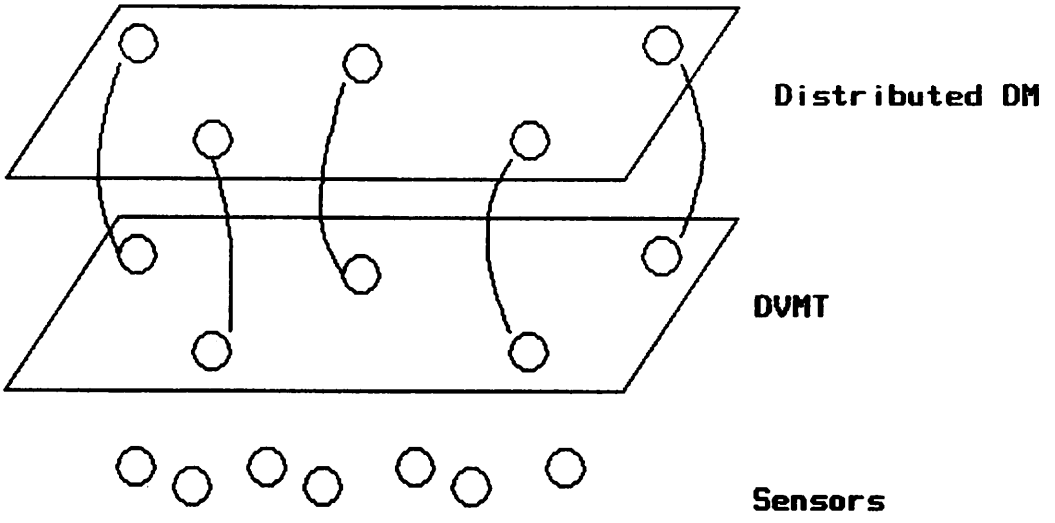


Figure 3: Distributed Diagnostic Architecture
The system organization here assumes a Diagnosis Module in each of the DVMT problem-solving system nodes. Each Diagnosis Module has access only to its own problem-solving node's data structures. When information is needed from other problem-solving nodes, the Diagnosis Module must request this information via messages to appropriate diagnosis modules.

system. This would then lead to a layered architecture. At some point, however, we would still have to assume that the highest level diagnostic system is correct.

Reader's Guide. *Section 2* describes the distributed problem-solving system whose behavior we want to diagnose, the Distributed Vehicle Monitoring Testbed (DVMT). Section 2 also describes the existing centralized Diagnosis Module. *Section 3* discusses the changes necessary to the centralized DM to make it capable of functioning in a distributed environment. *Section 4* discusses issues related to making the local control in each node more flexible. *Section 5* discusses our approach for handling situations where reasoning with uncertainty is required because no single fault hypothesis can be derived by the causal-model based diagnosis. *Section 6* concludes with a summary of this paper. **Appendix A** gives an example of local processing in the distributed diagnosis system. **Appendix B** outlines an example of how an ambiguous situation is resolved using the type of reasoning describe in section 5. **Appendix C** outlines the simulation of a distributed run of a 4 node diagnosis scenario.

2. Background Information: The DVMT and Diagnosis Systems

2.1 The DVMT Problem-Solving System

The problem-solving system we model and diagnose is the Distributed Vehicle Monitoring Testbed (DVMT). The DVMT is a distributed problem-solving system where a number of processors cooperate to interpret acoustic signals. The goal of the system is to construct a high-level map of vehicle movement in the sensed environment. The data is sensed at discrete time locations at the signal level. The final answer is a pattern track describing the path of a group of one or more vehicles, moving as a unit in some fixed pattern formation. In order to derive the final pattern track from the individual signal locations the data undergoes two types of transformation. The individual locations must be aggregated to form longer tracks, and both the tracks and the locations must be driven up several levels of abstraction, from the signal level, through the group and vehicle levels, up to the pattern level.

Each processor in the DVMT system is based on an extended Hearsay-II architecture where data-directed and goal-directed control are integrated [2]. The problem-solving cycle at each processor consists of creating a hypothesis that represents the position of a vehicle (See Figure 4). Hypotheses generate goals that represent predictions about how the existing hypotheses can be extended by incorporating more of the sensed data. A hypothesis together with a goal triggers the scheduling of a knowledge source (knowledge source instantiation) whose execution will satisfy the goal by producing a more encompassing hypothesis (one which includes more information about the vehicle motion). This process begins with the input data and repeats until a complete map of the environment is generated or until there are no more knowledge source instantiations to invoke.

2.2 The Centralized Diagnosis Module

The system behavior in the DVMT is modeled as a series of events. Each event results in the creation of an object (e.g., hypothesis, goal, or knowledge source instantiation) or the modification of the attributes of some existing object. *The states in the model represent the results of such events in the DVMT system.* Depending on what we want to model, a state may represent simply whether some event has occurred or it may represent some finer aspect of the event's outcome.[1]

Based on this model of system behavior, we have developed the **System Behavior Model (SBM)**, which consists of three major components:

[1] There are two types of states in the model. **Predicate states**, which represent whether an event has occurred or not, and **relationship states**, which represent the relationship among two objects in the DVMT.

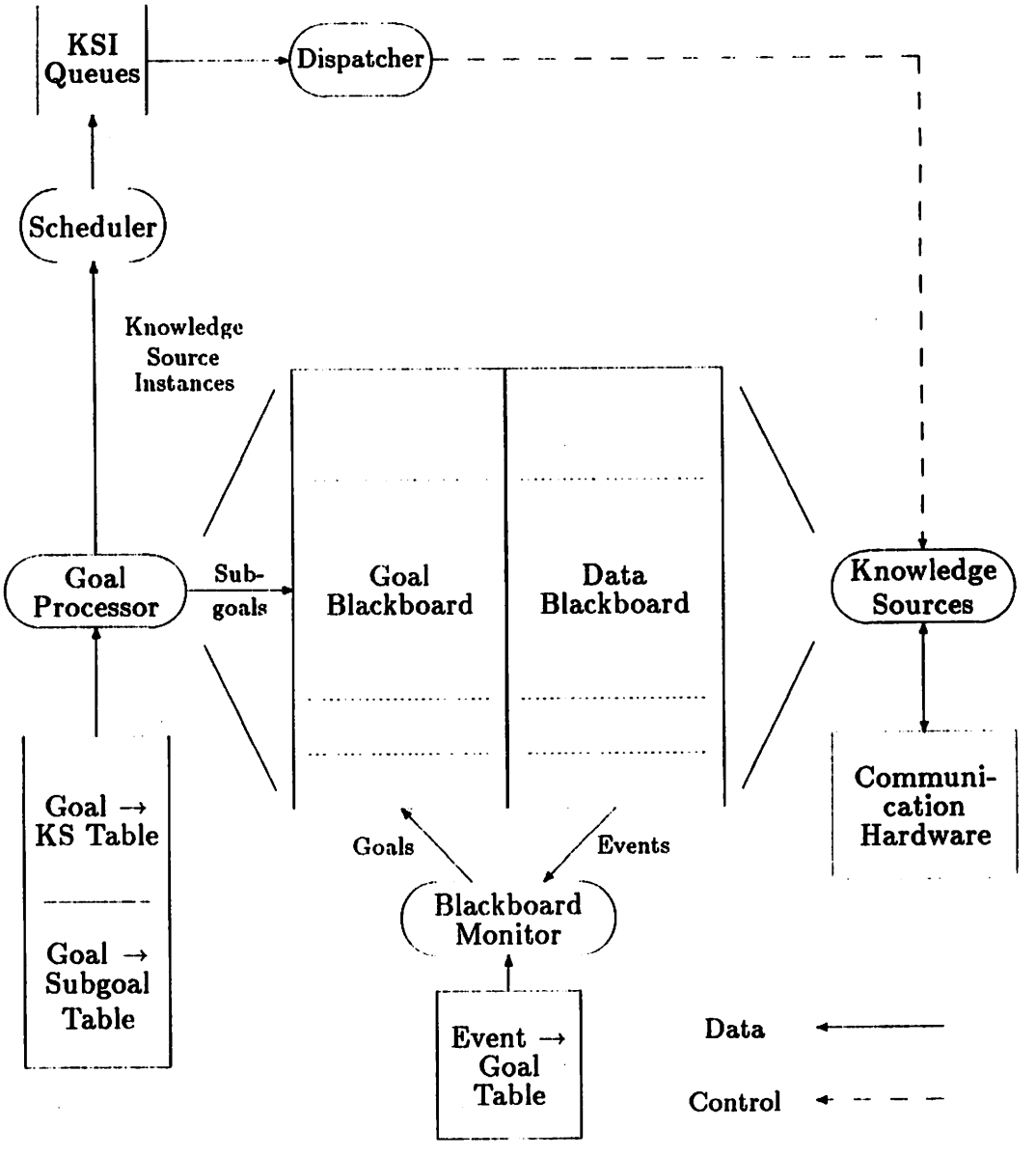


Figure 4: DVMT Node Architecture

HIERARCHICAL STATE TRANSITION DIAGRAMS which represent the possible system behaviors at various levels of detail. The causal relationships among events in the system are represented as sequences of states in the model.

ABSTRACTED OBJECTS which represent individual objects (i.e., data structures) or classes of objects in the DVMT system.

CONSTRAINT EXPRESSIONS among the different attributes of the abstracted objects which represent the relationships among the objects.

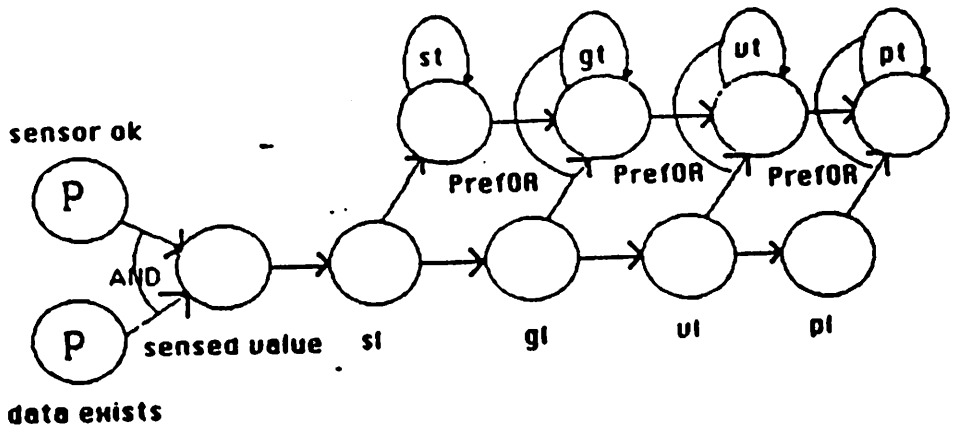
States are linked to other states through an AND/OR graph. If an event is influenced independently by a number of preceding events, then the states representing these events will be ORed. If the outcome of an event is influenced by a number of preceding events acting together, then the states representing these events will be ANDed.

The states are linked to the abstracted objects which describe characteristics of objects in the problem-solving system record; if objects exist that match these characteristics then the desired event that is specified by the state has occurred. The objects are represented as separate entities for efficiency reasons, to avoid the duplicate representation of similar sets of object attributes since several states may need to refer to the same object. An abstracted object may represent an individual object or it may represent a whole class of objects. An abstracted object representing a class of objects is called an **underconstrained object.**

The state transition diagram representing the system behavior is organized into small clusters for manageability (See Figure 5). These clusters are then organized into a hierarchy corresponding to increasingly detailed views of the system. Thus a high-level cluster represents selected events as contiguous states while a more detailed cluster represents other events which occur in between these states. Such a hierarchical representation allows reasoning at different levels of abstraction. This is useful during diagnosis because it allows the system to focus quickly on the problem by postponing a more detailed analysis until it is necessary. A subset of the states, designated as primitive, represents reportable faults during diagnosis.

Reasoning about DVMT behavior consists of instantiating the part of the SBM that represents the system behavior relevant to the situation being analyzed. The aim of all the different reasoning strategies is to explore the causes, or the effects, of an initial situation provided as input to the DM.[2] This situation is represented by an instantiated state and its abstracted object; that is, a state and object whose attributes have been evaluated. The DM propagates these known values through the SBM, using the constraint expressions

[2]Currently this is done by hand. In a fully fault tolerant system the input would come from a detection component.



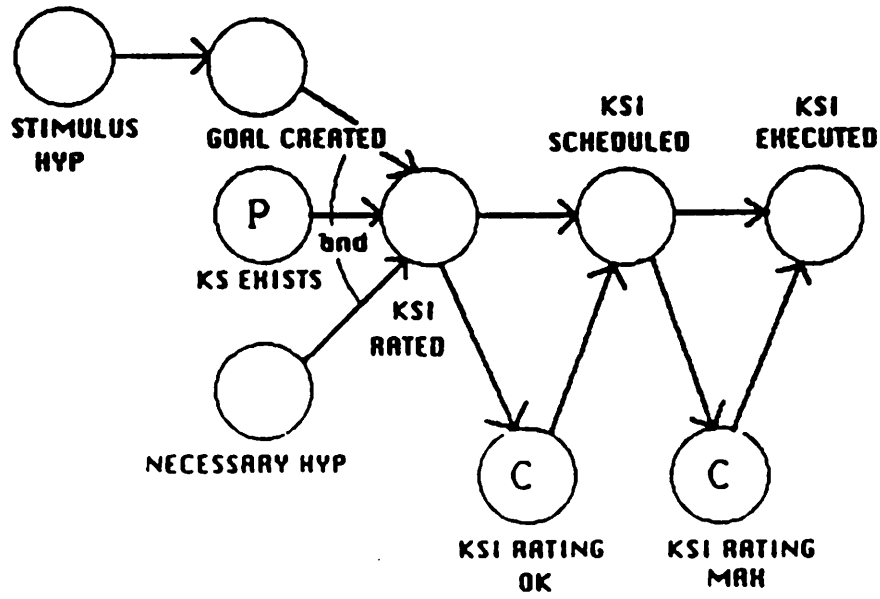


Figure 5: The Model Clusters

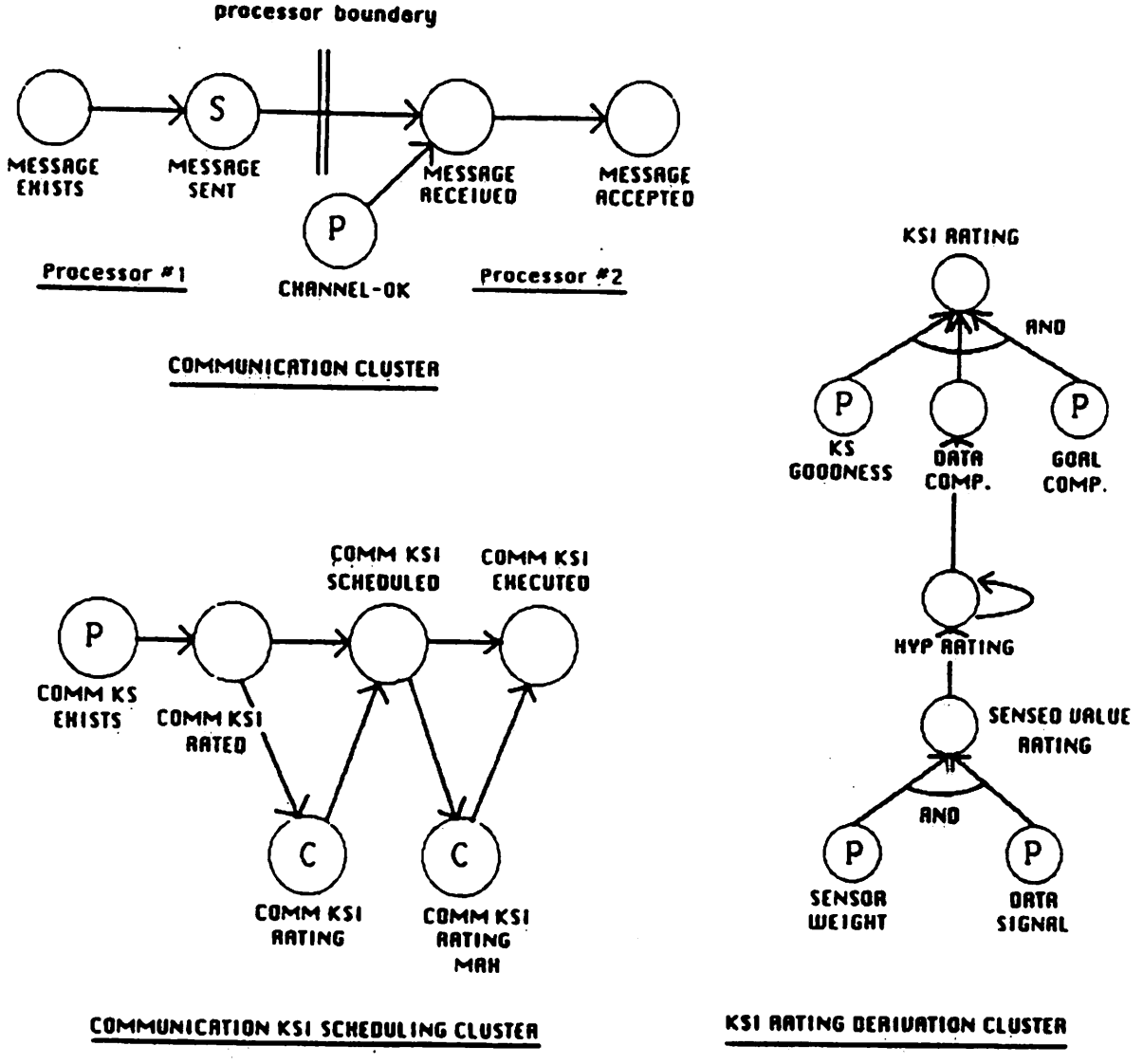


Figure 5: The Model Clusters Representing the DVMT Behavior.
This figure shows the diagrams of the five model clusters used in the diagnosis model. At the highest level are the Answer Derivation Cluster and the Communication Cluster. The Ksi Scheduling and Communication Ksi Scheduling Clusters represent the events that take place in between the states at the higher level models. The Ksi Rating Derivation Cluster represents the additional knowledge about value relationships among the rating components of the various objects. The circles represent the states, with the name of the state above or below them. The states with a **P** *indicate a primitive state. The state with* **S** *in the communication cluster indicates a state where the intermediate symptom is a primitive state so far as that node is concerned, and it needs to communicate that intermediate symptom to the connecting node. The states with* **C** *indicate states where Comparative Reasoning is used.*

among the abstracted objects, and thereby instantiates a sequence of states causally related to the initial situation. We call such a sequence a **causal pathway.**

When the initial situation represents some desirable event in the DVMT that never occurred, we call it a **symptom.** A symptom represents an object that was never created by the DVMT, usually a hypothesis.[3] Upon receiving a symptom, the DM traces back through the SBM in order to find out at which point the DVMT stopped working "correctly". This is done by comparing the behavior necessary for the desired situation to occur, as represented by the instantiated model, with what actually did occur in the problem-solving system, as determined from the DVMT data structures. The aim is to construct a path from the symptom state to some false primitive states which caused it and explain, in terms of these primitive causes, why the DVMT system did not behave as expected. This type of diagnosis consists of backward chaining through the SBM. Since it constructs a causal pathway linking the initial symptom to the faults that caused it, we call this type of reasoning **Backward Causal Tracing (BCT).**

The BCT search stops when all pathways relevant to the situation being analyzed have been explored. For example, in order to determine why some hypothesis was not constructed, the DM must examine all possible ways in which it could have been generated: via several pathways within a node, from locally available data, or from data received from other nodes. The analysis is done exhaustively by a depth–first search.

In addition to symptoms, initial situations may represent arbitrary events in the DVMT whose effect on the DVMT behavior needs to be simulated. This is the case, for example, when the DM needs to see what effects some identified fault, such as a faulty parameter setting or a failed hardware component, has on the DVMT system. This type of simulation thus consists of forward chaining through the SBM. Here the DM constructs a causal pathway which links the initial situation to all situations caused by it. We call this type of reasoning **Forward Causal Tracing (FCT).** FCT uses underconstrained objects to reason about the class of problems caused by the fault, rather than just the individual cases.

Both BCT and FCT are more complex than simple backward and forward chaining because the model is hierarchical and the interpreter must decide when to change the level of resolution, for instance, when to reason at different levels of abstraction. It must also deal with reasoning about classes of situations. We have also found the need for a new type of diagnostic reasoning that we call **Comparative Reasoning.** It deals with cases where no absolute standard for correct behavior is available. In such cases the diagnosis system selects its own "correct behavior criteria" from objects within the problem-solving system which did achieve some desired situation.

---

[3] A symptom could also represent a class of objects, such as all hypotheses at some blackboard level. This would be done using underconstrained objects.

3. Distributing a Centralized Diagnosis Module

3.1 Issues in Distributed Problem–Solving

When dealing with distributed problem-solving we must address the following points:

1. How to divide the problem into tasks which can then be solved by the individual agents.
2. How to allocate these tasks among the individual agents.
3. How to control the interaction among these agents.
4. And, finally, how to integrate the results of the agents into an overall solution to the original problem.

Dividing and Allocating Diagnostic Tasks. The study of a complex system often entails dividing it into subsystems that can be guided separately without constant attention to their interactions. In the case of diagnosing a distributed problem solving system like the DVMT this division naturally falls on the individual problem-solving nodes' boundaries. Each problem solving node (PSN), thus, has an associated diagnosis module that diagnoses only its behavior and that can directly access only its data structures. Whenever information is necessary from another DVMT node, it must be obtained by an explicit request to the Diagnosis Module (DM) of that node. For example, if DM in PSN 1 is diagnosing a missing PT (pattern-track) hypothesis symptom and reaches a MESSAGE-RECEIVED state which is false, it must send this intermediate symptom to the PSN that should have sent the desired message, say PSN 2. The DM at PSN 2 will then perform the necessary diagnosis and send the result back to PSN 1. PSN 1 can then make the final conclusions about the symptom, that is, whether faults were identified that account for it or whether it is unexplained.

Another form of task division occurs as a result of the causal model, where, due to the tree like structure of the model, there may be a number of alternative states, or a conjunction of states, that account for a given symptom. Thus, the initial problem is broken up into a number of subproblems. Frequently, it turns out that some of these subproblems are related, and that some of them have already been worked upon in a different context. It is important to recognize such situations for several reasons. First of all, it may save a lot of effort, since the diagnostic module can simply merge the current state with a previously investigated state if both represent the same situation. In such cases, the merged state inherits the causal pathway and pathvalues already traced out for the previously investigated state. Secondly, related subproblems might indicate interactions amongst several

causal pathways. When interactions can be anticipated, they can guide the prioritizing of the various subproblems.

Another issue in the division and allocation of tasks is that of redundancy. It is important to recognize both when to avoid it, and when to exploit it. One situation where it might be useful is when the same conclusion is arrived at from two different causal pathways, but both paths have low confidences. In such a situation this conclusion will get its confidence level boosted and the diagnosis will probably finish sooner.

On the other hand, redundancy would have a negative impact if pathways are needlessly being traced. For instance, when a particular state in a particular pathway has already been investigated in another context and does not need reaffirmation, it would be useful simply to merge it with that former one and let it inherit the previously calculated pathvalues, rather than tracing out the primitive fault along that pathway once again.

Coordination Among the Diagnosis Modules. The internode communication protocols control who talks with whom, how often, what type of messages they transmit, and how strongly each is affected by the other's messages. What this means in the distributed diagnosis context is which nodes send symptoms for diagnosis to which other nodes and how local processing is impacted by incoming symptoms or diagnostic results. These issues are similar to communication of partial results during the problem-solving process among the DVMT nodes.

Integrating the Results from Individual Nodes into an Overall Diagnosis. This is the problem of coherence in a distributed system. Given individual pieces of diagnosis from the distributed Diagnosis Modules, how do we combine them into an overall picture of the state of the DVMT's problem-solving? For instance, given an ambiguous situation, evidence or hypotheses from various nodes can collectively be weighed and used to make an overall diagnosis. Section 5 explains how such ambiguous situations are dealt with.

3.2 Architecture of the Distributed DM

The architecture of a distributed DM is shown in Figure 6. Each Distributed Diagnostic Module consists of the following components:

COMMUNICATION MODULES. These are the reception and transmission modules that are responsible for receiving and sending messages from/to other nodes in the form of symptoms to be diagnosed or diagnostic results to be integrated into the locally instantiated model. A local scheduling mechanism in each of these processors rates the pending messages and decides which to execute next.

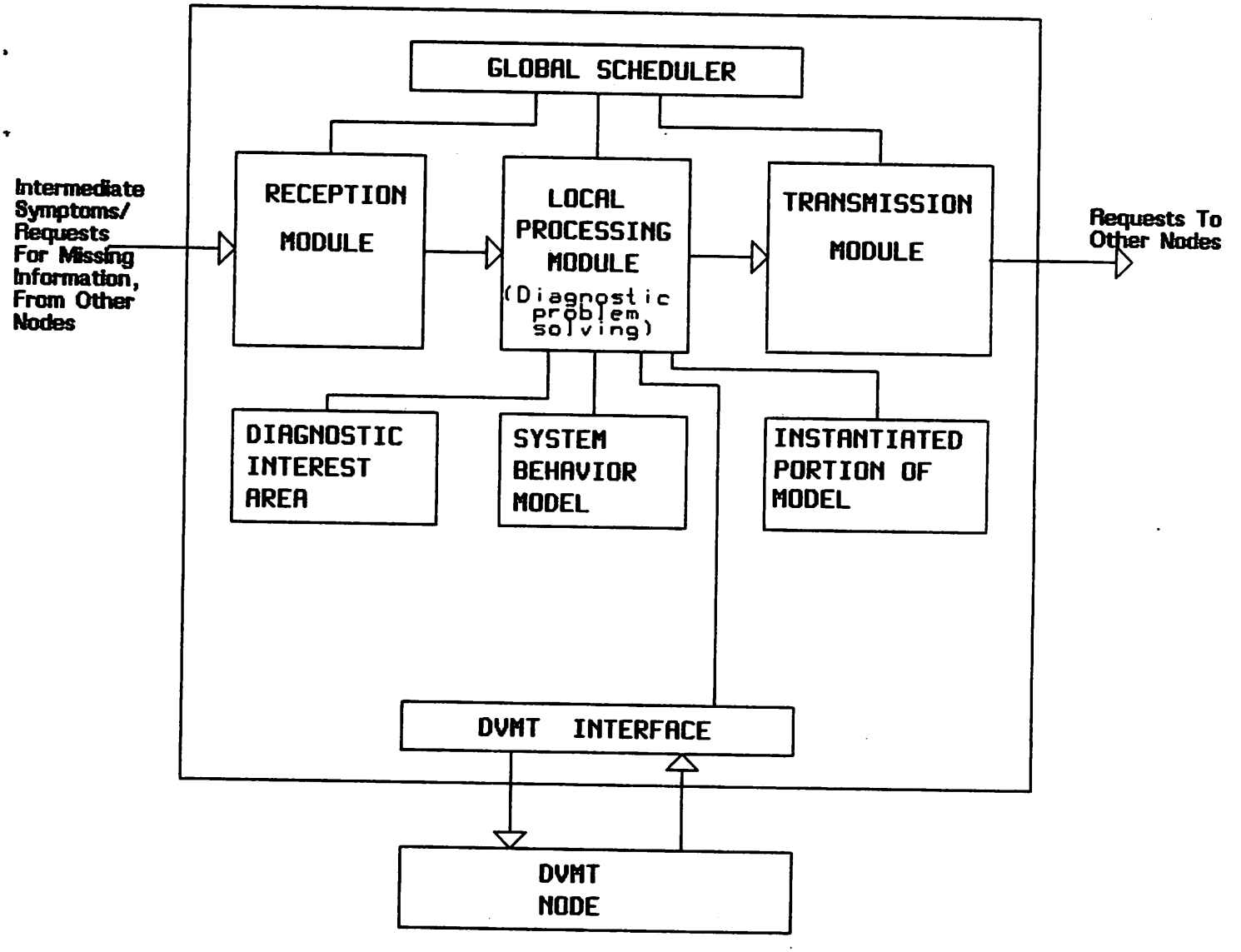


Figure 6: Architecture of A Distributed Diagnosis Module

LOCAL DIAGNOSTIC MODULE. This is the diagnostic inference mechanism that instantiates the causal model and constructs the causal pathways for the local symptom diagnosis. A local scheduling mechanism assigns a numeric rating to each partially expanded pathway.

GLOBAL SCHEDULER. The global scheduler's function is to decide which of the three possible primitive tasks (sending, receiving, or local diagnosis) will occur next. Depending on the number of independent processors one, two , or three tasks may be executed.

DVMT INTERFACE. This component controls all access to the DVMT data structures: the blackboards and the scheduling queues. In some cases direct probes into another node's data structures may be allowed (in addition to or instead of requests for diagnosis). In such cases these probes will go to the DVMT interface.

DATA STRUCTURES:

- *DIAGNOSTIC INTEREST AREA.* This data structure contains the information that enables the module to recognize those areas where it has priority in diagnosis. These areas are specified in terms of data type, location, and time.
- *SYSTEM BEHAVIOR MODEL.* This is the data structure that contains the uninstantiated causal model.
- *INSTANTIATED PORTION OF MODEL.* This data structure contains the instantiated version of the causal model that is valid for the current situation.

4. Making Local Control of Diagnosis More Flexible

Why Make Flexible Local Control? The major impact of distribution on local diagnosis is the need to make the diagnosis more efficient and its control more flexible. This flexibility is necessary so that the node can improve integrating its own local diagnosis with information coming from other diagnostic nodes. This information is either in the form of requests for diagnosis or in the form of identified faults. Currently, the centralized DM does an exhaustive depth-first search of the instantiated model. In a distributed setting this will no longer be appropriate since symptoms and identified faults from other nodes will have to be attended to and integrated into local processing.

There are various reasons that call for cooperation among nodes. First of all, the model itself might extend across node boundaries, for instance, in the case of the communication cluster which models the communication of messages among two DVMT nodes. In that case, once a node is tracing the cause of a symptom using Backward Causal Trace it will reach a point where the model continues across the node boundaries to the connecting node with whom the communication is supposed to have taken place. Thus the node will find it convenient to transfer the information over to that other node, let the other node continue with the BCT, and wait for the reply.

Secondly, a node might need to acquire some missing information from another node before it can continue with its diagnosis. For instance, consider again the case where the model continues across node boundaries. If the diagnostic node #1 had only this symptom to process, whereas diagnostic node #2 was already busy, it would be better (for load balancing purposes) for node #1 simply to request the raw data regarding the performance of the DVMT node #2 from diagnostic node #2, and then to continue diagnosing on its own.

Finally, a node will need to communicate with other nodes to resolve ambiguous situations. It will need to gather evidence from other nodes so as to arrive at a plausible explanation despite the ambiguity. (As outlined in Section 5). The Diagnosis Module will therefore have to be able to suspend a local task, to integrate new information into its current processing, and to make decisions as to which task to process next.

By executing the most highly rated task on a queue, the diagnostic reasoning will change from a depth-first search to a best-first search. In order to implement this flexible scheduling we must develop criteria for rating the primitive tasks. This amounts to rating each partially expanded causal pathway. Hence, it is important to determine the factors that influence the ratings of these tasks.

Factors Affecting the Rating of Causal Pathways. There are a number of factors affecting the promise of a causal pathway. Below is a list of some of these factors.

1. **A fixed rating associated with a specific state.** This would allow to specify preferences for examining certain states before others. For example, given the three back neighbors of the PT state: shorter PT segments, VT, and MESSAGE-ACCEPTED states, we could prefer to examine them in different orders. This would allow us, for example, to perform the diagnosis at one level of the model hierarchy before going on to a lower level.

2. **Diagnostic interest areas** control specific areas where diagnosis should have priority. These areas are specified in terms of data type, location, and time.

3. In addition to this generalized interest area, a more specific factor can influence the rating of some causal pathway: **the initial symptom rating.** All states that descend from this initial symptom state are influenced by its rating since it indicates the importance of investigating this pathway relative to others. If, for instance, it is found that this initial symptom is explained while investigating some other symptom, then obviously it is no longer necessary to continue tracing out its back neighbors. This is made possible by reducing the initial symptom rating so that states that lie along this causal pathway have their ratings decreased correspondingly. This is analogous to the subgoaling activity in the DVMT [2].

4. In some cases where states are related by an AND, **false states should be pursued with a higher priority**, since the cause is most likely along the false state path.

5. **Causal pathway reinforcement resulting from merged states.** If two or more causal pathways unite at some state that state should get a higher priority because that means that two symptoms require diagnosis along the same path.

6. The rating of symptoms transmitted among diagnostic nodes is influenced by the following factors.

   (a) **the reception rating** which is a function of
      i. rating at sending node
      ii. sending node authority factor
      iii. degree of similarity to local processing

   (b) **the transmission rating** which is a function of
      i. local rating
      ii. the number of other symptoms being sent out (the idea here is to give more importance to a given symptom if there are only few being sent – since this would indicate that there is less "confusion" about the possible diagnosis).

iii. receiving node factor

7. A symptom may represent either a specific situation which is being diagnosed or a number of situations. In the latter case we would represent this set of situations by a underconstrained object. Such symptoms, called **abstracted symptoms**, should have a rating that reflects the number and importance of each of their constituent individual symptoms.

8. **Length of the partially constructed causal pathway.** Here we see the horizon effect, also exemplified by the familiar graduate school dilemma: "If I've spent this much time on this, shouldn't I spend a little more and get the degree?" In this case the dilemma is whether the system should continue along this path and find the fault or should it abandon it.

Appendix C covers an example which uses the best-first strategy and some of the factors listed above that are valid for the situation. The example simulates the distributed diagnosis of a four node scenario that was actually run and solved by the centralized diagnostic monitor [10]. The example shows several ways in which distributed problem solving and flexible local control help in exploiting the parallelism in hardware, and yet at the same time, save in the amount of communication among nodes.

First of all, it helps to prioritize the causal pathways to be expanded next. In this example, the main factor is the concept of spatial "neighborness". The greater the proximity or overlap of a node to another, the more of a spatial neighbor it is to that node. Since pattern tracks normally extend over the node boundaries, it is more likely that a node receives a message from a closer node (one that is nearer or overlaps more) about a partial *pt* or *vt* hypothesis than from other nodes further away. Hence, it is better to start out examining those nodes that are nearer first, since it is most likely that it is from those nodes that messages might indeed have been received.

Secondly, on receiving an intermediate symptom to investigate, the node may sense that it would be advantageous to suspend its own local diagnosis and look into the other node's intermediate symptom first in the hope of finding out a fault that explains its other pending intermediate states or symptoms in its local queue. This happens in the example when a node realizes that the intermediate symptom that it receives from another node may be a related subproblem and, further, that it can be diagnosed with little effort. (In the example this occurs, for instance, in cycle #11 in the local diagnosis by node #1.)

Thirdly, the diagnosis by a node frequently explains not only the current symptom being investigated, but also other pending intermediate symptoms that other nodes are asking it to investigate as well. It thus expedites the overall diagnosis greatly in such situations. In the example this occurs frequently, when there are numerous MESSAGE-SENT states

concerning hypotheses of various types of pattern and vehicle tracks waiting to be traced out. On expanding any one if it is found out that the node lacks communication knowledge sources then on simulating the effects of such a fault (using FCT), it is found out that any MESSAGE-SENT state (represented by an underconstrained state in the FCT) is false as well. Thus the rest of the MESSAGE-SENT states are explained as false. It is not necessary to individually investigate all of those states anymore, and the replies to the respective nodes can be sent at this stage.

5. Resolving Ambiguous Situations

Ambiguous situations occur when an individual Diagnosis Module cannot decide on a fault because the values of some instantiated states cannot be determined simply by examining the DVMT data structures. Such situations are typical in the case of a suspected fault of a piece of hardware (e.g., sensor, communication channel) which cannot be probed directly to determine whether it is functioning.

For example, if there are no input hypotheses in a DVMT node, or if all hypotheses have very low beliefs, then a sensor fault is suspected. There are three possible combinations of events that can cause this situation:

- Sensor failed and there are vehicles in the environment.
- Sensor failed and there are no vehicles.
- Sensor did not fail and there are no vehicles.

Sensor failure can be further classified due to its type:

Full vs. partial sensor failure. In case of partial failure, the sensor's functionality can be impaired in a portion of its geographical range, or in a portion of its signal range (where it fails to detect or distorts only some types of signals). A partial failure can also be *transient* (a failure occurring only at a certain time).

Environmental irregularities. This occurs when a subset of the sensors is affected by features of the environment, for example a mountain that echoes some signals. This is a very difficult case to detect since the affected sensors do function correctly and yet disagree with their neighbors: the neighbors do not lie in the same position relative to the irregularity to be affected.

There are two ways to approach the ambiguous fault problem:

- by comparing the symptoms against some *external criteria.* The simplest external criterion is the correct result of the problem solving process. In general, however, the correct result is not available to the system, and we have to resort to less exact methods. One such method is a model of an ideal sensor with which a suspected sensor is compared to decide whether it is faulty or not.
- by relying on *internal consistency* and exploiting redundancy within the DVMT system. Thus, if the DM cannot determine based on local information whether there are any vehicles in the environment, it can examine the hypotheses at all nodes which had access to data. If none of them have any hypotheses, then it can assume that there are no vehicles.

We will concentrate on the means of exploiting redundancy in the system to disambiguate the fault. Redundancy can be both *local* (if more than one sensor receiving same or similar data report to the same node) and global (if nodes work on similar tasks). We call nodes working on similar problem solving tasks **behavioral neighbors**. These are the nodes which work with overlapping and/or contiguous data. Thus, some portions of their domains of interest (DOIs) either overlap or are adjacent.

Redundant information can be used to distinguish between the case where there are vehicles and the case where there are no vehicles in the observed part of the environment. If there are no vehicles, then there is still a question whether the sensor is faulty or not. However, the combination *no vehicles and a faulty sensor* is in general much less likely then the case of *no vehicles and a good sensor*; also, these two situations cannot be disambiguated by exploiting redundancy in the system, which is our primary interest. Thus, we will consider here only the mechanisms for disambiguating vehicles/no vehicles situation.

A node which detects the potential fault (we call it the **host**) starts off the disambiguation process by defining the two diagnostic hypotheses (one corresponding to no vehicles and the other corresponding to existing vehicles), and proceeds by assigning some measure of belief to each of them. The formalism we use for reasoning with uncertainty is Dempster–Shafer Theory. The theory is introduced in the next subsection, while Subsection 5.2 presents the formulation of the ambiguous diagnostic problem. In some cases, there will be enough information to disambiguate the situation locally. For example, if the host has no input hypotheses, and it has multiple sensors covering its DOI, then the host can be quite confident that there are no vehicles. More often than not, however, local information would not be decisive enough to make the "case closed", and communication will be necessary. Then, the host must make some communication decisions. Communication is examined in detail in Subsection 5.4. When the information from other nodes is received, the host must combine it with local information in a new attempt to disambiguate the situation. The mechanism for combining the information from different sources is described in Subsection 5.3.

5.1 Introduction to Dempster–Shafer Theory

Dempster–Shafer (DS) Theory provides a uniform framework for assigning and combining beliefs in a system which must deal with uncertain information. DS Theory generalizes Bayesian Probability Theory. Both theories model the world as a set of mutually exclusive and exhaustive propositions. However, in Bayesian Theory there is no distinction between conflicting evidence and the lack of evidence (in both cases, the probability of a proposition would be the same as the probability of its negation). DS Theory solves this

problem by introducing the concept of *ignorance* (lack of evidence). Its basic terminology and mechanisms are introduced below.

In order to emphasize the mutual exclusiveness and exhaustiveness of propositions of interest, it is convenient to think of these propositions as a finite set of answers to some question, where only one answer is the correct one. This set is denoted by Θ and called a *frame of discernment.* Each subset $P \subset \Theta$ can be interpreted as a proposition which states "*The correct answer is in P*".

A source of evidence distributes beliefs among the subsets of Θ by defining a **mass function.** It is a function

$$m : 2^{\Theta} \rightarrow [0, 1]$$

so that

$$m(\emptyset) = 0$$

and

$$\sum_{A \subseteq \Theta} m(A) = 1.$$

If complete information is available, then masses are assigned to singletons in the frame, just like probabilities in Bayesian theory. However, an evidential source can also represent a lack of evidence by attributing masses to multi-element subsets in the frame of discernment. The price paid for this generality is a greater computational complexity.

If multiple sources of evidence supply evidence about the propositions in the same frame of discernment, then the combined mass function can be obtained using **Dempster's rule.** For two sources of evidence with mass functions m_1 and m_2 respectively, Dempster's rule defines the combined mass function to be

$$m(C) = m_1 \otimes m_2 = \frac{\sum_{A \cap B = C} m_1(A) \cdot m_2(B)}{1 - K} \tag{1}$$

where

$$K = \sum_{A \cap B = \emptyset} m_1(A) \cdot m_2(B).$$

K is a measure of conflict between the two sources. If $K \neq 0$ then either an evidence source is in error, or the representation is incorrect (e.g., maybe a proposition is missing from Θ).

Belief in a proposition $A \subseteq \Theta$ is the extent to which the evidence tends to support the proposition:

$$bel(A) = \sum_{X \subseteq A} m(X). \tag{2}$$

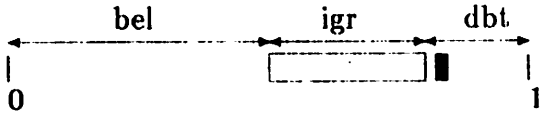


Figure 7: The Belief Intervals.

Dubiety of a proposition $A \subseteq \Theta$ is the extent to which the evidence tends to refute the proposition:

$$dbt(A) = \sum_{X \subset \overline{A}} m(X). \tag{3}$$

Plausibility of a proposition $A \subseteq \Theta$ is the extent to which the evidence fails to refute the proposition:

$$pls(A) = 1 - dbt(A). \tag{4}$$

Ignorance of a proposition $A \subseteq \Theta$ is the extent to which the evidence about the proposition is lacking:

$$igr(A) = pls(A) - bel(A). \tag{5}$$

The relation between belief, plausibility, dubiety and ignorance is depicted in Figure 7. Note that $0 \leq bel(A) \leq pls(A) \leq 1$.

Once the masses have been assigned to the propositions in the frame of discernment, it must be decided which proposition is most likely true. We introduced a decision measure, credibility, which is a function of belief, plausibility and ignorance. **Credibility** of a proposition $A \subseteq \Theta$ is defined as:

$$crd(A) = \frac{bel(A) + pls(A)}{2} \cdot (1 - igr(A)). \tag{6}$$

Credibility of a proposition is high when the belief interval is narrow (low ignorance) and its middle is close to 1. Thus, if enough information has been gathered about a proposition, and the information tends to support it, then this proposition is credible.

The following subsection describes the actions that a DM must take when it detects an ambiguous diagnostic problem.

5.2 Formulating the Ambiguous Diagnostic Problem

As noted earlier, a DM detects an ambiguous fault situation when it finds either that there are no input hypotheses in some part of its DOI or that all input hypotheses have very low belief. We will call this area the **diagnostic area** (DA).

After detecting the problem, the host forms a diagnostic frame of discernment, Θ :

$$\Theta = \{h, \bar{h}\}$$

where

$$h = \textit{There are vehicles in DA}$$

and

$$\bar{h} = \textit{There are no vehicles in DA.}$$

If the host has no data in the DA then true h indicates a sensor fault (missing data) while true $\bar{h}$ indicates no fault. The fault is total if the DA covers the whole sensor range, and partial if the DA covers a portion of the sensor range. On the other hand, if the host suspects a fault because all local data have low belief, then true h indicates a sensor fault (ghost data).

After defining Θ, the host assigns masses to its propositions based on local information. This distribution is based on a priori likelihoods of data being present, on data beliefs if there are data, and on the reliability of the sensors covering the DA. This information can favor one of the propositions, for example in the case of a reliable sensor or multiple sensors providing the same data. In the case of a single sensor, however, the information would not be decisive enough, so the host will need to find other nodes which have evidence to support or refute these propositions. When the host decides which nodes to communicate with (how to make this decision is the subject of the Subsection 5.4) it asks each node to supply the best partial solution correlating with data in the DA. Three classes of situations can arise:

Nodes send **no data** and the host has no data in the DA. The resulting mass distribution would make the host almost certain that h is the right answer, and the diagnostic process can stop.

One or more nodes send a very **credible partial solution.** This would make the host almost certain that h is the right answer, and the diagnostic process can stop also.

Only a **low belief partial solution** is received, or no data are received but the host has the low belief data. In this case, more information is needed before a decision about the fault can be made, and more communication with neighbors is necessary.

The next subsection describes an extension of DS Theory and its use in combining received information.

5.3 Combining Fault Information

Once the information from neighbors is received, it should be combined with local information. The weight given to evidence from a node depends on node's reliability and the relation between its DOI and the host's DA. To account for this weight, we consider the evidence from another node to be about propositions in the *evidence space*, E, while the host's fault frame, Θ is the *hypothesis space* $|2^{\Theta}|$. Thus, the mass attributed to a subset $Q \subseteq \Theta$ as a result of evidence e_i from node i has two components: conditional probability of the hypotheses in Q given the evidence e_i, $p(Q|e_i)$, and the probability of the evidence, $p(e_i)$:

$$m_i(Q) = \sum_{e_i} p(Q|e_i) \cdot p(e_i). \tag{7}$$

The conditional probability defines the strength with which the evidence implies hypotheses in Q. In the fault domain, this probability is determined by two factors: the relation between node's DOI and the DA and reliability of the node. For a perfectly reliable node i whose DOI completely overlaps the DA, $p(Q|e_i) = 1$.

Masses $m_i(Q)$ for different nodes i cannot be combined directly using Dempster's rule, since this would result in overweighing the prior probabilities $p(Q)$, as shown in [YEN86]. Instead, a mass $m_i(Q)$ must be converted into a *certainty value*, $c_i(Q)$, which discounts the priors:

$$c_i(Q) = \frac{\frac{m_i(Q)}{p(Q)}}{\sum_{A \subseteq \Theta} \frac{m(A)}{p(A)}} \tag{8}$$

Certainty values can be combined using Dempster's rule (equation 1 in Section 5.1) which can be converted back to a mass function using the equation:

$$m(Q) = \frac{c(Q) \cdot p(Q)}{\sum_{A \subseteq \Theta} c(A) \cdot p(A)} \tag{9}$$

If the best hypothesis in Θ is considered to be the one with the greatest credibility value, then the next step in uncertainty calculation would be to calculate credibility from belief, plausibility and ignorance for each singleton subset Q, using the equations 2, 4, 5 and 6 defined in Section 5.1. Appendix B describes an example of this calculation.

In this subsection we have described the means to incorporate received information into the local diagnosis, but we have not discussed the methods for obtaining the diagnostic information from other nodes. The next section discusses the issues involved in making communication decisions, and our approach to communication of diagnostic information.

5.4 Communication of Diagnostic Information

In the distributed diagnosis context, several communication decisions have to be made. The issues are similar to communication of partial results during the problem solving process among the DVMT nodes:

Policy: Which nodes communicate? We choose *selective* rather than *broadcast* communication, since only a small subset of nodes can help with the diagnosis of the host's symptom, namely, some or all of its behavioral neighbors.

Style: Should communication be *incremental* or *synchronous*? In incremental communication, the host asks the best single node for relevant information, and after the information is received decides whether to make another round of communication or not. In synchronous communication the host sends the request for information to every node in the list of relevant nodes. Incremental communication has a lower processing cost if there is only one round of communication, since there is less information to process. However, if additional communication is required, the amount of information can become as high as for the synchronous case, with an additional loss of timeliness caused by the delay between communication rounds. Thus, the incremental style would be appropriate if there is one node with a distinctly high likelihood for providing the necessary information, making it likely that none or little additional communication would be required. If this is not the case, then synchronous communication is preferred.

Content: What types of messages are transmitted? There are three categories of information, classified according to the function they perform:

1. requests for diagnosis
2. reply to a previous diagnostic request
3. reception of an unsolicited diagnostic information, i.e., an identified fault at another node.

Amount: Does each node communicate all of its locally generated symptoms/faults or only selected categories? For example, some parts of node's DOI may be more important than others (e.g., if they contain critical installations). The factors affecting the importance of a symptom/fault will be combined into a symptom/fault rating and only those above some threshold will be transmitted.

Reliability: Some nodes may be more reliable than others. A DM prefers to diagnose a reliable symptom or to integrate newly arrived diagnosis from a reliable node. Also,

a message coming from a reliable node is given more weight than one coming from a less reliable one, as described in the previous subsection.

Communication policy and style are decided based on the relevance of behavioral neighbors to the potential fault and on the cost of communication. The relevance of a neighboring node is based on the similarity between its problem solving task and the problem solving task of the host. A neighbor would be most relevant if it had a DOI which is exactly the same as the DA, since in that case the node would be most likely to provide data that could determine whether the fault has occurred. A node is also potentially relevant to the fault problem if its DOI does not overlap but borders the DA. Thus, the following are the main factors that determine relevance:

Overlap If a node has an area overlapping the DA, its relevance is proportional to the size of the overlapping area.

Adjacency If a node's DOI has a common boundary with the DA, its relevance is proportional to the length of the common boundary.

The relevance of a set of nodes is obtained by combining the overlap and adjacency contributions of each node. Multiple nodes covering the same part of the DA contribute more than a single node covering that part of the DA. If multiple nodes cover different parts of the DA, they contribute more then if they cover the same part of the DA.

Unfortunately, the task of finding the best nodes to communicate with cannot be effectively represented in DS Theory since the propositions of interest are not mutually exclusive. Namely, if all actions in some set of actions A are equally good, then all propositions of the form

$$p = \textit{The best actions are in } B$$

are true, where $B \subseteq A$.

Since the Utility Theory [1] is concerned with the problem of deciding what is the best action to take, we represent the problem of finding the best nodes to communicate with within a utility model.

Our utility model must be able to express various tradeoffs involved in communication. The utility of communication is a trade-off between timeliness of information and the cost to process it; this trade-off is different for incremental and synchronous communication. In Utility Theory, it is most common to represent various actions and corresponding outcomes by a decision tree. The *expected utility* of an action a, $Eu(a)$, is the sum of the products of probability and utility for each of its outcomes, o_i:

$$Eu(a) = \sum p(o_i) \cdot u(o_i).$$

The best course of action is the one which yields the greatest expected utility.

Our goal is to find the best sequence of actions, where each action is communication with some subset of relevant nodes. Possible outcomes fall in one of the two groups: either no more communication is required after an action (the host has acquired the necessary information about the fault), or some additional actions should be taken. The probabilities of various outcomes can be estimated from the relevance of other nodes to the fault problem.

The following example describes a decision tree and corresponding decisions for three relevant nodes, A, B and C.

5.4.1 An Example of a Communication Decision Tree

A portion of the decision tree for this problem is shown in 8.

There are two types of nodes in the tree: *choice nodes* (denoted by diamonds) and *chance nodes* (denoted by circles). A choice node has as many branches as there are possible actions at that stage. Each branch of a choice node is labeled by a corresponding action. For example, following the communication with node A, two actions are possible: sending to node B alone (the branch labeled by B) and sending to nodes B and C together (the branch labeled by $B\&C$).

Each chance node has exactly two branches, one labeled M (more communication is necessary) and the other labeled $\overline{M}$ (no more communication is necessary). The parenthesized expression at a leaf of the tree represents the utility of the corresponding outcome, while the expression below an outcome represents the probability of the outcome. A combined probability, such as $p(A\&B)$ is the probability of the event of getting the desired information from communicating with both nodes (A and B in this case), given the probability of getting the information from each node separately, $p(A)$ and $p(B)$. This probability depends on the relation between the DOIs of the two nodes. If the two DOIs do not overlap, for example, $p(A\&B) = p(A) + p(B)$. If DOIs overlap, we can treat these two events as independent, in which case $p(A\&B) = p(A) + p(B) - p(A) \cdot p(B)$.

The utility is defined to be the reciprocal of the total cost. The cost of communication has three components: the cost to send the message, the cost to process received information and the cost of delay while waiting for the reply. In the figure, c_i denotes the cost of communication with i nodes at once, and subsumes the cost to send the message and the cost to process received information. Cost d represents the cost of one communication cycle delay.

The action that has the best utility is obtained by calculating the utility of each node (starting from the leaves) and backing up these values through the tree. Calculation of utility for each type of node is described below.

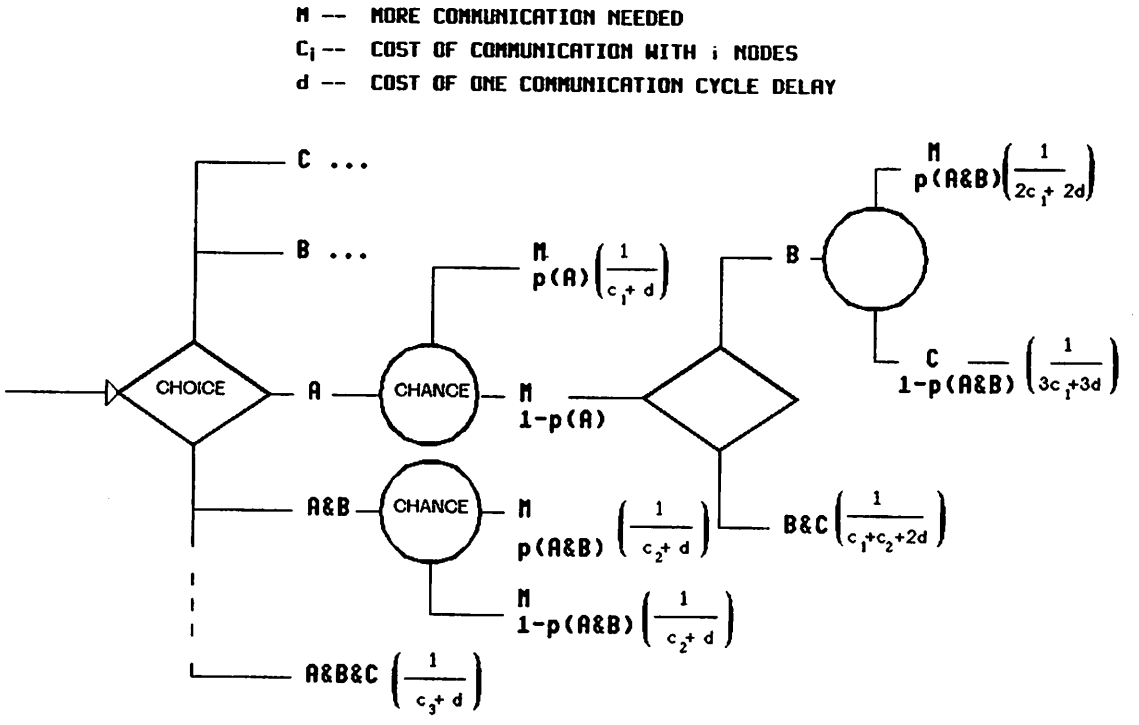


Figure 8: A Portion Of The Decision Tree For Three Relevant Nodes, A, B and C.

Each branch of a choice *node is labeled by a corresponding action. Label M on a* chance *node indicates that more communication is necessary and label $\bar{M}$ indicates that no more communication is necessary. The parenthesized expression at a leaf of the tree represents the utility of the corresponding outcome, while the expression below an outcome represents the probability of the outcome.*

The utility of a chance node is the product of probabilities and utilities of the two outcomes (M and $\overline{M}$).

The utility of a choice node is the maximum of the utilities of its branches. This value is backed up as the utility of the corresponding outcome of the higher level chance node. Thus, for example, if action $B\&C$ had higher utility then B, then the utility of the M outcome of the action A would be $\frac{1}{c_1+c_2+2\cdot d}$.

Even for a relatively small number of behavioral neighbors, the size of the decision tree is considerable and complete evaluation of all nodes is a costly procedure. However, most of the paths are known not to involve the best action, and they can be cut off without

changing the result of the evaluation. Probabilities of nodes can always be ordered, e.g., $p(A) \geq p(B) \geq p(C)$. In this case, sending to A first is guaranteed to be better then sending to any other single node first, and the two branches corresponding to sending to B and C first can be eliminated. Similarly, sending to $A\&B$ first is guaranteed to be better then sending to any other pair of nodes. In general, instead of all subsets of relevant nodes, only one subset of each size needs to be considered. Thus, the tree in Figure 8 has all the nodes that are needed to determine the best action.

When $p(A)$, $p(B)$ and $p(C)$ are all relatively low, then it is best to communicate with all three nodes at once. As $p(A)$ becomes higher with other two probabilities unchanged, sending to node A followed by sending to B and C together becomes the best action sequence. On the other hand, if both $p(A)$ and $p(B)$ are increased relative to $p(C)$, then the best action sequence is sending to A, followed by sending to B, followed by sending to C.

In this section, we have dealt with a situation where a DM does not have enough information to decide whether a fault has occurred, and needs to seek help from other nodes. We have formulated this ambiguous fault problem within the extended Dempster–Shafer framework, and shown how a DM can integrate the information about the fault received from other nodes to achieve more credibility in a proposed explanation of the diagnostic situation. We have also developed a utility model for deciding which nodes to communicate with in this situation. The utility of a communication action is a function of the expected value of received information (relevance of other nodes to the diagnostic problem) and the cost of communication. The cost of communication is different for synchronous and incremental communication.

6. Conclusion

We have presented a design for a Distributed Diagnosis System and have explored some of the implications of distributing a diagnosis system. Some of the issues we had to deal with in the design result from distribution, and are not relevant to the centralized system. For instance, communication policies among nodes falls into this category. However, an interesting aspect of this exploration is that many of the changes that we have made to the centralized diagnosis module in order to distribute it are, in fact, modifications that are also essential for the centralized system. For instance, the need for flexible control (that is, an agenda based view of diagnosis) introduced in the distributed system in order to effectively deal with non--local requests for diagnosis can be used in a centralized system to do diagnosis more efficiently. Likewise, the extension to the diagnosis system for handling locally ambiguous situations through evidential reasoning which became apparent when we thought of a distributed version is also appropriate for a centralized version. The study so far involves the system design and its hand simulation. The results from hand simulation are encouraging. They show several ways in which distributed problem solving and flexible local control help in exploiting the parallelism in hardware, and yet, at the same time, save in the amount of communication among nodes. However, it is very difficult to do hand simulation of any problem of real interest. We are currently implementing the distributed diagnosis system so that we can perform empirical evaluation of our ideas. In these empirical experiments, we expect to look at issues of global coherence, that is, do the diagnosis modules really work together or is there a need for high level (meta–level) control in order to coordinate them. We see this as one of the key issues that we need to further explore.

We are also excited about the evidential reasoning model which we used for integrating diagnosis evidence from multiple sources, and we intend to investigate the use of Theory of Evidence in integrating problem solving results from multiple sources as well.

We have found that the Utility Theory provides an appropriate framework for modeling tradeoffs involved in making decisions about communicating diagnostic information. Since communication is but one aspect of the general control problem, the next step in our investigation is to examine the applicability of the Utility Theory to the problem of making optimal control decisions.

REFERENCES

[1] H. Chernoff, Lincoln E. Moses. *Elementary Decision Theory.* **John Wiley and Sons, New York, 1959.**

[2] Daniel D. Corkill, Victor R. Lesser, and Eva Hudlicka. *Unifying data-directed and goal-directed control: An example and experiments.* **Proceedings of the Second National Conference on Artificial Intelligence,** August 1982, pp. 143–147

[3] Daniel D. Corkill. *A framework for organizational self-design in distributed problem solving networks.* **Ph.D. Dissertation,** Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, February 1983.

[4] Stephen E. Cross. *An Approach to Plan Justification Using Sensitivity Analysis.* **Sigart,** No. 93, July 1985, pp. 48–55,

[5] R. Davis, H. Shrobe, W. Hanscher, K. Wieckert, M. Shirley, and S. Polit. *Diagnosis based on descriptions of structure and function.* **Proceedings of the Second National Conference on Artificial Intelligence,** August 1982, pp. 137–142.

[6] Randall Davis. *Diagnostic Reasoning Based on Structure and Behavior.* **Artificial Intelligence,** Vol. 24, 1985, pp. 347 410.

[7] M. Genesereth. *Diagnosis using hierarchical design models.* **Proceedings of the National Conference on Artificial Intelligence,** August 1982, pp. 278–283.

[8] J. Gordon, and E. H. Shortliffe. *A method for managing evidential reasoning in a hierarchical hypothesis space.* **Artificial Intelligence,** 26(1985), pp. 323-357.

[9] Eva Hudlická and Victor Lesser. *Meta-level control through fault detection and diagnosis.* **Proceedings of the National Conference on Artificial Intelligence,** August 1984, pp. 153 161.

[10] Eva Hudlická. *Diagnosing Problem-Solving System Behavior.* **Ph.D. Dissertation,** Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, February 1986.

[11] Van E. Kelly and Louis I. Steinberg. *The CRITTER system: Analyzing Digital Circuits by Propagating Behaviors and Specifications.* **Proceedings of the National Conference on Artificial Intelligence**, 1982, pp. 284–289.

[12] Victor R. Lesser and Lee D. Erman. *Distributed Interpretation: A Model and an Experiment.* **IEEE Transactions on Computers**, Special Issue on Distributed Processing Systems, Vol.C-29,No. 12, December 1980, pp. 1144–1162.

[13] Victor Lesser and Daniel D. Corkill. *The Distributed Vehicle Monitoring Testbed: A tool for investigating distributed problem solving networks.* **AI Magazine** 4(3):15–33, Fall 1983.

[14] Ronald Loui, J. Feldman, H.G.E Kybung. *Interval-based Decisions for Reasoning Systems.*

[15] Drew McDermott and Ruven Brooks. *ARBY: Diagnosis with Shallow Causal Models.* **Proceedings of the National Conference on Artificial Intelligence**, 1982, pp. 370–372.

[16] Ramesh S. Patil, Peter Szolovits, and William B Schwartz. *Causal Understanding of Patient Illness in Medical Diagnosis.* **Proceedings of the Seventh International Joint Conference on Artificial Intelligence**, Vol. 2, 1981, pp. 893–899.

[17] Chuck Reiger and Milt Grinberg. *The Declarative Representation and Procedural Simulation of Causality in Physical Mechanisms.* **Proceedings of the Fifth Joint Conference on Artificial Intelligence**, Vol. 1, August 1977.

[18] G. Schaffer. *A Mathematical Theory of Evidence.* **Princeton University Press**, Princeton, NJ, 1976.

[19] Sholom M. Weiss, Casimir A. Kulikowski, Saul Amarel, and Aran Safir. *A Model-Based Method for Computer-Aided Medical Decision Making.* **Artificial Intelligence**, 11:145–172, 1978.

[20] Leonard P. Wesley *Evidential Knowledge-based Computer Vision* **Technical Note No. 374 SRI International**, January 1986.

[21] John Yen. *A Reasoning Model Based on an Extended Dempster-Shafer Theory.* **Proceedings of AAAI-86**, pp. 125-131.

A. Example of Local Processing at a Diagnostic Node

This section discusses in detail how the local processing is controlled at a Diagnostic Node. In order to make local control more flexible, we changed the original exhaustive depth-first approach of the model instantiation to a best-first. Each instantiated state has a rating that represents that state's promise in leading to a successful diagnosis. This rating is determined as a part of the state instantiation process. Section 3 discusses the rating determination in detail. The states awaiting expansion (i.e., the leaf states of the instantiated model that represent partially constructed causal pathways) are in a priority queue (agenda), ordered by their rating. Figure 9 illustrates this.

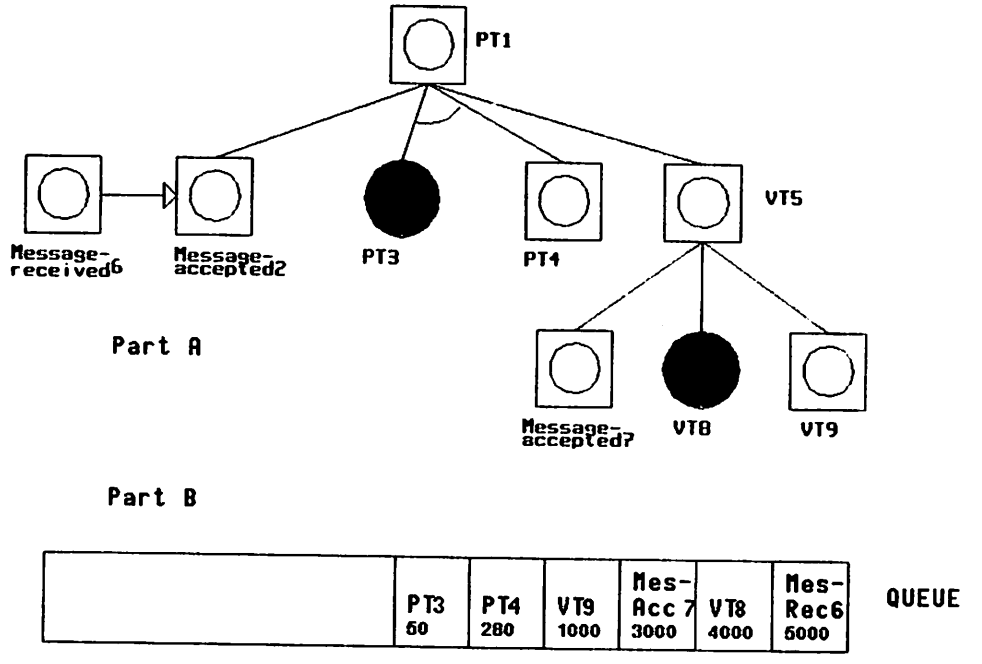


Figure 9: Agenda-Based Best First Search

Part A of the figure shows a partially instantiated model. Each leaf state represents an end of a partial causal pathway. Each instantiated state has a rating representing its diagnostic promise. Part B shows the priority queue consisting of the leaf states, ordered by their rating.

Each state includes contextual information that is necessary to resume the diagnosis by continuing the construction of the causal pathway. This contextual information resides in the RESTART-INFO attribute of each instantiated state and contains the type of search

being conducted, the state's previous state, and, if necessary, the state's neighbors at a higher level of the hierarchy.

A local processing system cycle consists of processing a single state. This state is taken from the priority queue (the highest rated state), and one of several actions is performed:

1. The construction of the causal pathway continues as before and the appropriate neighbors of the state are instantiated, along with their objects. (*Function instantiate-neighbor-list*).

2. The construction of a causal pathway cannot continue or a transition in the diagnosis occurs and some special handling must be done. Situations where this occurs are:

   - A primitive node in the model has been reached or the model cannot be expanded any further. In these cases a failure may be reported and the path value is propagated back through the model.
   - A node transition state has been reached and must be sent to the appropriate node for diagnosis. In that case the path value cannot be assigned and the local processing must wait until the symptom is diagnosed at the other node.
   - The reasoning must be suspended in order to perform another type of reasoning. This occurs in the case of Forward Causal Tracing where for each instantiated state its path value must be determined by Backward Causal Tracing before the construction of the forward pathway can continue.

In order to make clear the exact nature of local processing, we will "walk through" a simple example. Consider the model in Figure 10.

In this example we will begin with the arrival of the symptom A1, and its associated object ob-AB1. The state A1 is false and diagnosis will therefore begin by expanding the back neighbors of this state, until eventually the primitive states are reached. At that point, Forward Causal Tracing (FCT) will be invoked and the effect of the identified faults will be propagated forward in order to account for any pending symptoms. We will show in detail what occurs at each system cycle, how the agenda changes, how FCT is initiated, and how the path values are propagated through the constructed causal pathways.

System Cycle #1. At the beginning of cycle 1 the agenda is empty and the symptom arrives, resulting in the instantiation of state A into A1, and object ob-AB into ob-AB1 (see Figure 11, Part A). State A1 has a rating of 5000, and is inserted into the agenda when it is instantiated. A1 is false and the contents of its *restart-info* attribute are:

B1 nil nil nil

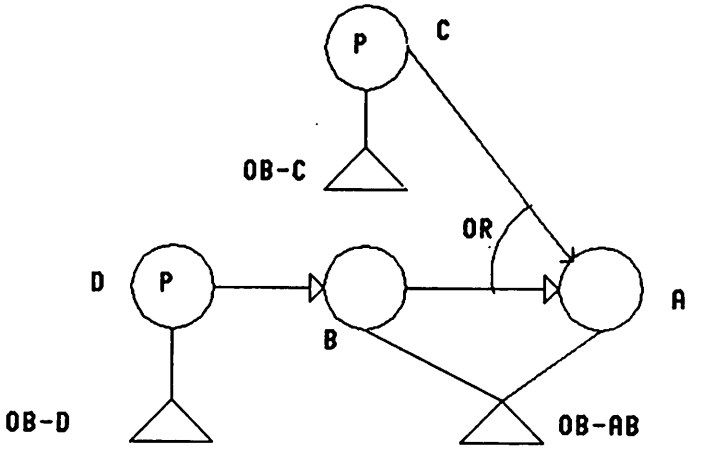


Figure 10: A Detailed Example of Local Processing
The model represents a series of situations, beginning with the primitive state D and its associated object ob-D. This state should be followed by state B, which should be followed by state A. Both state A and B refer to the same object, ob-AB. State A can also be achieved through another pathway, that beginning with the primitive state C, and its associated object ob-C.

indicating that the search currently active is B1 (a Backward Causal Tracing type search), and the PREVIOUS-STATE, UPPER-FRONT-NEIGHBOR, and UPPER-BACK-NEIGHBOR are all nil, since this is the initial symptom state. This information will allow the diagnosis to continue when the state is selected for expansion at some later point.

System Cycle #2. A new cycle begins with the selection of the highest rated state on the agenda for expansion. A1 is the only state so it is selected. The Diagnosis Module (DM) first checks whether model expansion can continue, by checking to see whether any terminating conditions, such as primitive state, non-expandable state etc., have been reached. They have not, so the expansion continues. Based on the search type currently active (B1) and the state value (false), one of the state's neighbors is selected for further expansion. In this case it is the back neighbor (b-n). The b-n neighbor list for state A is (OR (OR B) (OR C)). The Diagnosis Module must therefore instantiate both states B and C. This results in the model and agenda shown in Part B of Figure 11. Notice that since both A and B refer to the same object, ob-AB, both states A1 and B2 point to object ob-AB1. State C3 has its own object, ob-C2. The state A1, having been expanded, is removed from the agenda, and the new states are added, ordered by their rating. The b-n of state A1 is set to its instantiated version, which is (OR (OR B B2) (OR C C3)).

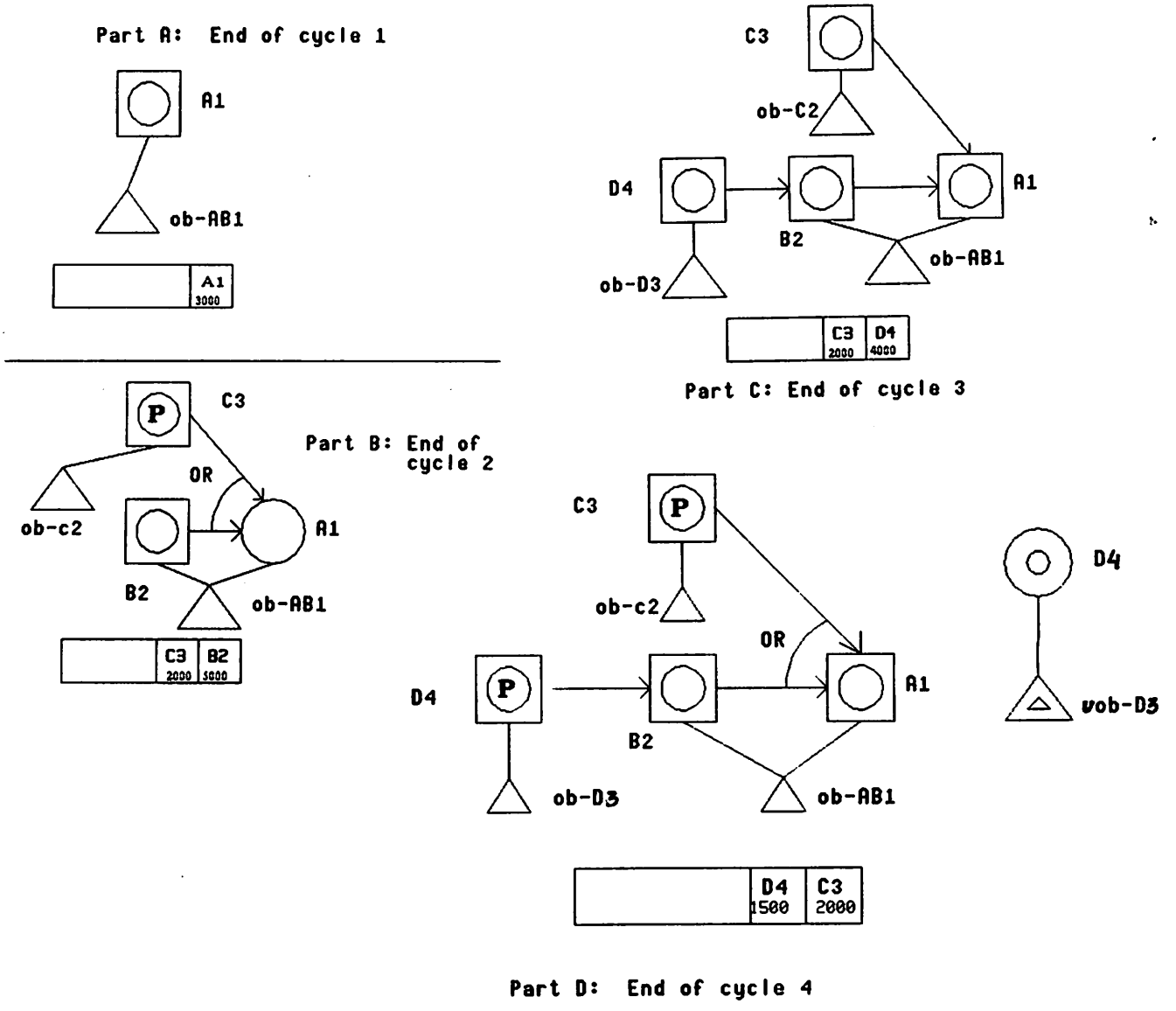


Figure 11: Progressive Construction of the Causal Pathways
The instantiated state A and its object ob-AB in Part A represent the initial symptom. The other parts show the subsequent expansion of the causal pathways, cycle by cycle.

System Cycle #3. Since B2 is higher rated, it is selected for expansion first. Since it is false and B1 type search is active, the DM again selects the back neighbors for instantiation. The b-n of state B is (OR (OR D)). State D is instantiated resulting in the model and agenda shown in Part C of Figure 11. The instantiated b-n of state B2 is now (OR (OR D D4)).

System Cycle #4. The highest rated state in the agenda is the primitive false state D4. This state cannot be expanded any further, because it is a primitive state. The DM therefore begins the special handling procedures for primitive states. First, the path value for this state is assigned. For primitive states the path value is the same as the state value, in this case False. This is inserted in the *path-value* attribute of the state D4. Since a false primitive state represents a fault, a fault is reported (a message is printed). If the state had been true, no fault would have been identified, and a message indicating this would have been printed. The next step is to perform the operations indicated in the *upon-completion* attribute of the state D4. A state is considered "completed" when its path value has been assigned. This indicates that it has been "explained" in terms of some set of primitive states and that a diagnostic result can be reported. In this case, D4 has an empty *upon-completion* attribute.

Forward Simulation of the Identified Fault. However, whenever a false primitive state is encountered, the identified fault represented by that state may be propagated forward, subject to that type of reasoning being allowed by the current DM system parameters. (In this case this is the *p:reasoning-types parameter.) The DM therefore checks this parameter to see whether FCT can be initiated. If it can, then an underconstrained object corresponding to the ob-D3 is created along with an instantiation of the D state. This results in the instantiated model and agenda shown in Part D of Figure 11. The difference between the underconstrained object uob-D3 and the fully constrained (sometimes also called regular) object ob-D3 is the lack of values for some attributes in the underconstrained object. The LTM (Long Term Memory) definition of each abstracted object contains an attribute called *fl-attributes*. This attribute lists the names of attributes which should have values in the underconstrained version of the object. This allows ignoring certain details about an object and thus represent an entire class of objects by just one instantiation of an underconstrained object. For example, an underconstrained object corresponding to a certain type of knowledge source instantiation represents the class of all knowledge source instantiations with KSs of the same type, regardless of the specific data the knowledge source is to work with. This underconstrained object does not include any of the KSI attributes dealing with specific data. In this case, FCT can indeed be initiated and the corresponding states and objects are instantiated. The *restart-info* attribute of the state D4 now looks like this: F1 nil nil nil; indicating that the type of search being done is F1 (FCT type search), and that the previous, front-upper-level, and back-upper-level states are all nil. FCT will be resumed whenever the state D4 is expanded. One last thing needs to be done to process a primitive state: the state's path value must be propagated backwards as far as possible in the causal pathway. This is done as follows.

Backward-Propagation of a Path Value. First, the previous state of the fault state must be found. This is obtained from the *restart-info* attribute of the fault state D4. The contents of this attribute are: B1 B2 nil nil, corresponding to the current search type, previous state, front-upper-level, and back-upper-level states respectively. Next, the neighbor relation between the states D4 and B2 is determined to be b-n; i.e., D4 is a back neighbor of B2 and we therefore need to calculate the b-n path value of state B2, using the instantiated b-n neighbor list of B2.

Calculating Path Value. A path value for a given neighbor type is calculated by evaluating the instantiated neighbor expression for that neighbor, substituting the overall path values for each instantiated state in that neighbor expression. In this case, the instantiated b-n of state B2 is: (OR (OR D D4)). If we substitute the states' path values for each instantiated state in this expression and if we remove the marker state D, we obtain the following expression: (OR (OR F)), which of course evaluates to False. The b-n path value for state B2 is thus F, and this information is inserted into the path value attribute of B2, resulting in the following: nil F nil nil nil nil.

The propagation of the path value continues. Again, the previous state of B2 is determined from its *restart-info* and this is state A1. The relation between states B2 and A1 is again b-n, so we repeat the process. This time we need to evaluate the instantiated b-n of state A1, which is (OR (OR B B2) (OR C C3)) . Substituting the path values for all instantiated states and removing the marker states we obtain: (OR (OR F) (OR ?)). The reason for the ? is that the path value of state C3 has not yet been determined, since that state is still awaiting expansion on the agenda. Although some expressions could be evaluated without the value for each term (take (AND F ?), for example), our expression cannot be evaluated. The calculation of the path value must therefore stop at this point. It will resume when the path value of state C3 is calculated. This completes cycle number 4.

System Cycle #5. The next state to expand is C3. This again, is a primitive state and similar type of processing results as in cycle # 4. When the path value of C3 is propagated back, the instantiated b-n of A1 can be evaluated, resulting in the path value F for state A1. This means that one or more faults were identified that caused the symptom represented by A1.

B. Example Of Determining Credibilities of Competing Hypotheses

Let us consider $\Theta = \{h, \bar{h}\}$ where h is the hypothesis

$$h = \textit{There are vehicles in host's DAI}$$

and $\bar{h}$ is the hypothesis

$$\bar{h} = \textit{There are no vehicles in host's DAI}$$

with prior probabilities $p(h) = 0.7$ and $p(\bar{h}) = 0.3$. One type of evidence, e_1, is a track on the host's blackboard. This evidence, when present, strongly supports h and disproves $\bar{h}$:

$$p(h|e_1) = 0.8\ p(\Theta|e_1) + 0.2\ p(\bar{h}|e_1) = 0.$$

When the evidence is not present the probabilities are divided between h and Θ:

$$p(h|\bar{e}_1) = 0\ p(h|\bar{e}_1) + 0.5\ p(\Theta|\bar{e}_1) = 0.5.$$

The host has a very weak evidence of a track: $p(e_1) = 0.2$ *and* $p(\bar{e}_1) = 0.8$. The host's mass distribution is obtained using equation 7:

$$m_1(h) = p(h|e_1)p(e_1) + p(h|\bar{e}_1)p(\bar{e}_1) = 0.8 \times 0.2 + 0 \times 0.8 = 0.16.$$

$$m_1(\bar{h}) = p(\bar{h}|e_1)p(e_1) + p(\bar{h}|\bar{e}_1)p(\bar{e}_1) = 0 \times 0.8 + 0.5 \times 0.8 = 0.40.$$

$$m_1(\Theta) = p(\Theta|e_1)p(e_1) + p(\Theta|\bar{e}_1)p(\bar{e}_1) = 0.2 \times 0.2 + 0.5 \times 0.8 = 0.44.$$

Credibilities of h and $\bar{h}$ are calculated using the equations 2, 4, 5 and 6 from Section 5.1 [6] $crd_1(h) = 0.2$ and $crd_1(\bar{h}) = 0.24$.

Since both hypotheses have low credibility, the host seeks evidence, e_2, from a fully reliable node whose DOI overlaps the DA. When this evidence is present with certainty, h is true. Thus, $p(h|e_2) = 1$ and $p(\bar{h}|e_2) = 0$. The absence of e_2 in this node strongly supports h and leaves Θ in part unknown:

$$p(h|e_2) = 0.8\ p(h|\bar{e}_2) + 0\ p(\Theta|\bar{e}_2) = 0.2.$$

There is no track on this node's blackboard, so $p(e_2) = 0$ and $p(\bar{e}_2) = 1$. Using equation 7 we get the following mass distribution for this node:

$$m_2(h) = p(h|e_2)p(e_2) + p(h|\bar{e}_2)p(\bar{e}_2) = 1 \times 0 + 0 \times 1 = 0,$$

[6] In the interest of simplicity, $\{h\}$ and $\{\bar{h}\}$ have been replaced by h and $\bar{h}$ respectively.

B. Example: Determining Credibilities of Hypotheses

$$m_2(\bar{h}) = p(\bar{h}|e_2)p(e_2) + p(\bar{h}|\bar{e}_2)p(\bar{e}_2) = 0 \times 0 + 0.8 \times 1 = 0.8,$$
$$m_2(\Theta) = p(\Theta|e_2)p(e_2) + p(\Theta|\bar{e}_2)p(\bar{e}_2) = 0 \times 0 + 0.2 \times 1 = 0.2.$$

Using the equation 8 the certainty assignments for the two nodes become:

$$c_1(h) = \frac{\frac{m_1(h)}{p(h)}}{\frac{m_1(h)}{p(h)} + \frac{m_1(\bar{h})}{p(\bar{h})} + \frac{m_1(\Theta)}{p(\Theta)}} = \frac{\frac{0.16}{0.7}}{\frac{0.16}{0.7} + \frac{0.40}{0.3} + \frac{0.44}{0.7+0.3}}$$

$$= \frac{0.23}{0.23 + 1.33 + 0.44} = 0.23/2.00 = 0.11,$$

$$c_1(\bar{h}) = \frac{\frac{m_1(\bar{h})}{p(\bar{h})}}{\frac{m_1(h)}{p(h)} + \frac{m_1(\bar{h})}{p(\bar{h})} + \frac{m_1(\Theta)}{p(\Theta)}} = \frac{0.40/0.3}{2.00} = \frac{1.33}{2.00} = 0.67,$$

$$c_1(\Theta) = \frac{m_1(\Theta)/p(\Theta)}{2.00} = \frac{0.44/(0.7 + 0.3)}{2.00} = 0.44/2.00 = 0.22.$$

and

$$c_2(h) = \frac{\frac{m_2(h)}{p(h)}}{\frac{m_2(h)}{p(h)} + \frac{m_2(\bar{h})}{p(\bar{h})} + \frac{m_2(\Theta)}{p(\Theta)}} = \frac{\frac{0}{0.7}}{\frac{0}{0.7} + \frac{0.8}{0.3} + \frac{0.2}{0.7+0.3}}$$

$$= \frac{0}{0 + 0.8/0.3 + 0.2} = \frac{0}{2.86} = 0.$$

$$c_2(\bar{h}) = \frac{\frac{m_2(\bar{h})}{p(\bar{h})}}{\frac{m_2(h)}{p(h)} + \frac{m_2(\bar{h})}{p(\bar{h})} + \frac{m_2(\Theta)}{p(\Theta)}} = \frac{0.8/0.3}{2.86} = \frac{2.66}{2.86} = 0.93.$$

$$c_2(\Theta) = \frac{m_2(\Theta)/p(\Theta)}{2.86} = \frac{0.2/(0.7 + 0.3)}{2.86} = \frac{0.2}{2.86} = 0.07.$$

respectively.

The combined certainty values are obtained using Dempster's rule (equation 1). For illustrative purposes, the *intersection table* [8] is a helpful device. Each entry contains two items: a subset and a parenthesized value. The first row and the first column contain the subsets and the values assigned by c_1 and c_2 respectively. The subset in entry i, j in the table is the intersection of subsets in row i and column j. The parenthesized value next to the subset is the product of c_1 and c_2. From the intersection table for our example

$c_1 \backslash c_2$	$h, 0$	$\bar{h}, 0.93$	$\Theta, 0.07$
$h, 0.11$	$h, 0$	$\emptyset, 0.1023$	$h, 0.0077$
$\bar{h}, 0.67$	$\emptyset, 0$	$\bar{h}, 0.6231$	$\bar{h}, 0.0469$
$\Theta, 0.22$	$h, 0$	$\bar{h}, 0.2046$	$\Theta, 0.0154$

we get the normalization constant

$$1 - 0.1023 = 0.8977$$

and

$$c(h) = 0.0077/0.8977 = 0.01,$$

$$c(\bar{h}) = (0.6231 + 0.2046 + 0.0469)/0.8977 = 0.97,$$

$$c(\Theta) = 0.0154/0.8977 = 0.02.$$

The combined mass function becomes (using equation 9):

$$m(h) = \frac{c(h)p(h)}{c(h)p(h) + c(\bar{h})p(\bar{h}) + c(\Theta)p(\Theta)} = \frac{0.01 \times 0.7}{0.01 \times 0.7 + 0.97 \times 0.3 + 0.02 \times 1}$$

$$= \frac{0.007}{0.007 + 0.291 + 0.02} = \frac{0.007}{0.318} = 0.02,$$

$$m(\bar{h}) = \frac{c(\bar{h})p(\bar{h})}{c(h)p(h) + c(\bar{h})p(\bar{h}) + c(\Theta)p(\Theta)} = \frac{0.291}{0.318} = 0.92,$$

$$m(\Theta) = \frac{c(\Theta)p(\Theta)}{0.318} = \frac{0.02}{0.318} = 0.06.$$

Credibilities of hypotheses are now $crd(h) = 0.04$ and $crd(\bar{h}) = 0.89$. The credibility of h has been reduced and the credibility of $\bar{h}$ has been greatly improved by incorporating the received information. The host can now conclude that, in fact, there are no vehicles in its DOI.

C. Example of Diagnosis: Missing Communication Knowledge Sources

This section discusses an experiment illustrating how the Distributed Diagnosis Module (DDM) identifies the faults in the DVMT system responsible for the system's inability to derive the correct result. The scenario for this example is a four node system (Figure 12).

Four sensors are distributed over the environment such that each node receives data from the sensor in its quadrant. The data are the signals generated by a moving vehicle. There are 8 time frames containing the signals. The signals will therefore be referred to as signals 1 through 8. The goal of the system is to integrate these signals into a pattern track describing the motion of the vehicle. Since no node receives all the data necessary to construct the final answer hypothesis, communication among nodes is necessary. Node #1 receives no signals locally. Node #2 receives signals 5 through 8. Node #3 receives signals 1 through 4. Node #4 receives signals 3 and 6. The local domain of interest (DOI) and the knowledge sources allow the nodes to process its local data up to the vehicle track level, by following the $sl \rightarrow gl \rightarrow vl \rightarrow vt$ derivation path. The locations are integrated into tracks at the *vt* level. Since Node #1 has no local data, its local interest area parameters have been set so that it does not work at levels *sl*, *gl*, or *vl*. (This was done only to simplify the example presentation.) The communication interest area parameters allow the nodes to communicate at the *vt* level. Partial results are thus transmitted in the form of vehicle track hypotheses. Each node works only in its own area up to the *vt* level but can work in the entire system area at the *vt* and *pt* levels, provided it has received the necessary data from the other nodes. Any node could therefore in principle derive the final answer. The domains of interest are shown in Figure 13.

The symptom that initiates the diagnosis is the lack of a spanning *pt* hypothesis at any of the nodes, i.e., a pattern track hypothesis integrating all eight sensed locations. In order to derive it, the nodes must communicate. However, due to faulty parameter setting, the Communication Knowledge Sources (CKSs) responsible for sending the hypotheses among the nodes at the vehicle track level, are missing. There are two CKSs which can transmit vehicle track hypotheses amongst nodes, *hyp-send:vt* and *hyp-reply:vt*, and thus there are two faults responsible for the lack of communication among the nodes.

As described earlier, each cycle for each node involves activities by 3 processors running concurrently: the Reception Processor that receives messages from other nodes, and integrates these messages into the current nodes diagnosis processing; the Local Diagnosis Processor that does the diagnosis based on a modified DM; and the Transmission Processor that transmits request/reply messages to other nodes.

The Local Diagnosis processor functions as follows:

C. Example: Model Use in Diagnosis

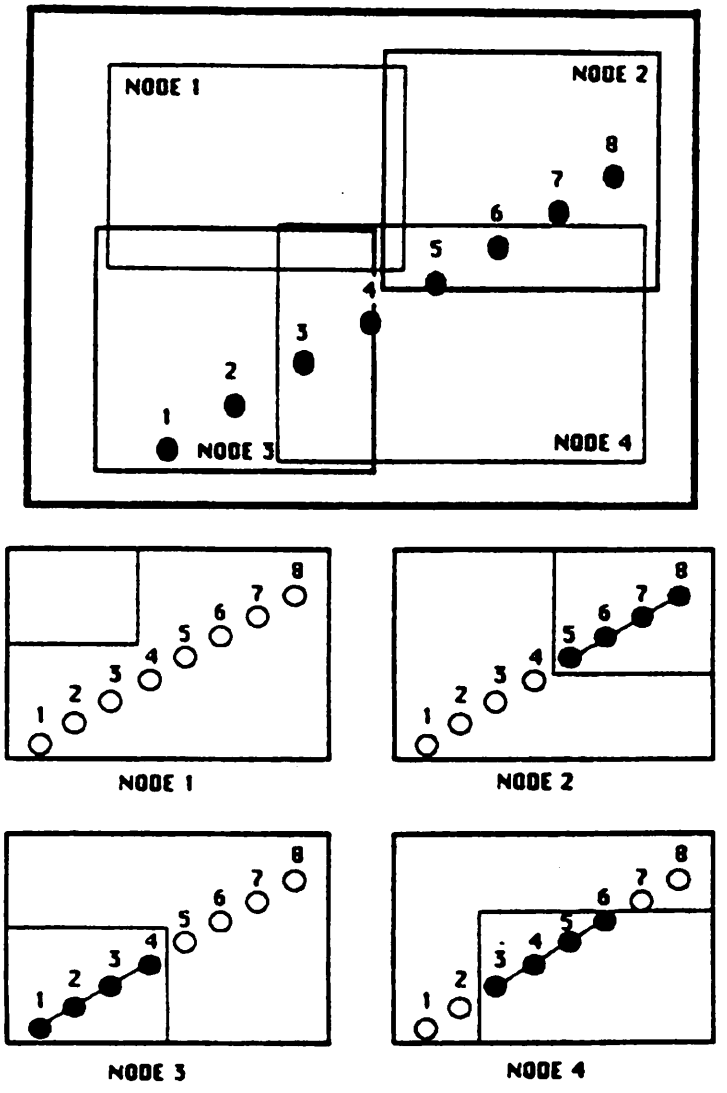


Figure 12: Scenario for Example

The upper part of the figure shows the overall view of the four nodes and the data. The lower part shows the pattern track segments derived at each of the individual nodes.

Step 1: The initial symptom is mapped on to a state and associated object in the causal model. This state, its associated object, and appropriate links are instantiated. Since the initial symptom usually represents some incorrect behavior of the monitored system, this initial symptom state is evaluated as false, and is placed at the head of the queue.

Step 2: The highest rated state (representing the most promising path to be explored

C. Example: Model Use in Diagnosis

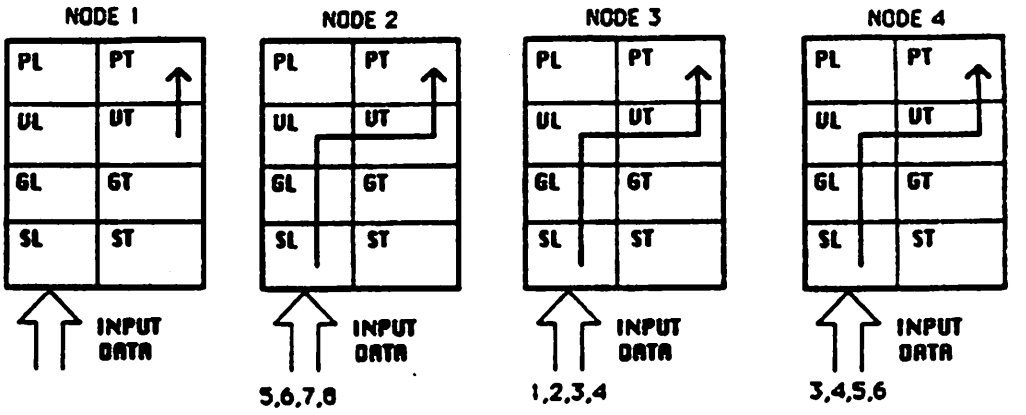


Figure 13: The Domains of Interest for Example I

This figure illustrates the answer derivation pathways allowed by the DOI in the four nodes. In addition to the pathways shown, there are communication links among the four nodes at the vehicle track level.

is considered next. (It is the state at the head of the queue). Expand the relevant set of its neighboring states. Initially, since most symptom states are false predicate states, diagnosis begins by expanding the back neighbors of the symptom state, in order to determine what predecessor states could have led up to it (Backward Causal Tracing). Subsequently, exactly which set of state-object pairs is instantiated next depends upon both the types of reasoning currently active and what the state value is; that is, whether the situation represented by the state occurred in the DVMT system. Only the parts of the model relevant to the situation being analyzed are instantiated. The unique state names, objects, and associated links are created.

Step 3: Evaluate the states in this set of neighboring states. In other words, determine whether these states have been reached by the DVMT system. This is done by either looking at the DVMT system data structures, or by looking at the surrounding states. If a state that is represented as a primitive state is evaluated as false, a fault has been found, and the effect of the fault is found by tagging Forward Causal Tracing to it as the reasoning type to be invoked when subsequently expanding that state. If, on the other hand, the state is found to have already been evaluated along another causal pathway, this state is merged with the corresponding state in the other causal pathway, and inherits its path value. It is not put on the queue, since its value is known.

Step 4: Rate these newly evaluated states. The ratings are derived using the applicable factors listed in *section 3*. These states are then placed appropriately on the queue.

Step 5: Go to step 2 for the next cycle, as long as there are states still in the queue.

This example is meant to illustrate 1) the communication of messages and their effects in the distributed diagnosis, and 2) how the determination of ratings and the best first strategy is used in expanding states.

The convention for naming the states is as follows:

State(Node#.Cycle#)/special information if necessary

Here, *State* refers to the state type, such as PT for pattern-track or MS for Message-sent; the *Node#* refers to the node involved in instantiating this state; the *Cycle#* refers to the particular cycle in which this state was instantiated; the *special information* is needed for certain states and not for others. In most cases the special information simply refers to the *pt* or *vt* tracks being considered. In the case of MS and MA states the *Node-from#* is needed as well to indicate the node from which the message is communicated as well. Thus, VT(1,9)/*vt 1-4* indicates the VT state instantiated in cycle #9 by node #1, and pertaining to a *vt 1-4* hypothesis. Similarly, MS(1,8)/3*vt 1-4* refers to the MS state instantiated during cycle 8 by node #1, and referring to a communication from node #3 of a *vt 1-4* hypothesis.

NODE #1

Node #1, in the upper left corner of the environment, receives no input data. Since it does not have any locally sensed data, this node relies on communication at the *vt* level to receive all its data, in the form of partial results generated by the other nodes in the system. The interest areas are set such that this node can only begin working on data at the *vt* level. Due to the missing CKSs however, this node never receives any hypotheses and therefore does not produce any results. Figure 14 shows a trace of the simulation.

Cycle 1: The initial symptom is, thus, the lack of a spanning pattern track (PT) hypothesis. This is mapped onto the *pt* state in the Answer Derivation Cluster. Therefore, diagnosis begins with the false state PT(1,1)/ *pt 1 8*, representing the desired answer pattern track integrating locations 1 through 8 (*pt* 1-8). The DM initiates diagnosis with BCT by expanding the back neighbors of this state.

Cycle 2: The back neighbors of PT are PT (since all track states are reflexive), PL, and VT states in the Answer Derivation Cluster, and MESSAGE-ACCEPTED state in the Communication cluster. In this case, no shorter *pt* segments are found, so no PT states are instantiated. Normally the PL states would be tried next, but since *pl* is not within the DOI, they will not be considered. This is also the case with the MESSAGE-ACCEPTED states, since the communication DOI allow communication only at the *vt* level. The remaining possibility is the state VT. The object which would normally lead to the derivation of the *pt 1 8* hypothesis is a *vt 1 8* hypothesis. The DM therefore creates its corresponding state, VT(1,2)/ *vt 1 8*. Since a vehicle track 1-8 does not exist in the

DVMT system, this state is false. This is the only back neighbor of the initial symptom state.

Cycle 3: The back neighbors of the false VT state are expanded next. The back neighbors of a VT state are VT, VL, and GT in the Answer Derivation Cluster, and MESSAGE-ACCEPTED in the Communication Cluster. Hence, the choices are shorter *vt* segments, vehicle locations, and, since *vt* is a communication level, messages received and accepted from the other three nodes in the system. Since no shorter *vt* segments exist, no VT back neighbors are instantiated. The VL state is found to be non-expandable because the corresponding *vl* objects lie outside the node's DOI. This non-expandable VL state thus provides a false path value that is propagated back up through the model. When the path values of the other back neighbors of the VT state become known, they will be combined using the OR logical operator which links the back neighbors of the VT state, to determine the overall VT path value.

The MESSAGE-ACCEPTED back neighbor of the VT state is expanded into 3 separate states, each representing messages from one of the other three nodes. These are the states MA(1.3)/2 *vt 1–8*, MA(1.3)/3 *vt 1–8*, and MA(1.3)/4 *vt 1–8*, representing messages accepted from node #2, node #3, and node #4 respectively. These states are determined to be false since no *vt 1–8* hypothesis exists in node #1 that was received by from either of the three other nodes. The three MA states are rated. At this stage the factor that influences the ratings the most is the extent to which the other nodes are spatial neighbors to node #1. Since node #3 overlaps the most with node #1, state MA(1.3)/3 *vt 1–8* is rated highest, followed by MA(1.3)/2 *vt 1–8*, followed by MA(1.3)/4 *vt 1–8*.

Cycle 4: Diagnosis continues with the MA(1.3)/3 *vt 1–8* state which represents the *vt 1–8* hypothesis expected from node #3. Since it is false, its back neighbor, the state MESSAGE-RECEIVED, is expanded, resulting in the creation of the state MR(1.4)/3 *vt 1–8*. This state is determined to be false.

Cycle 5: The back neighbors of the state MR(1.4)/3 *vt 1–8* are expanded, resulting in the instantiation of the states MS(1.5)/3 *vt 1 8* and CHANNEL-OK(1.5)/3. These states represent the sending of the message from another node and the state of the communication channel linking the two nodes respectively. CHANNEL-OK(1.5)/3 is found to be functioning and thustrue.

The diagnosis for this path ends here because the state MS(1.5)/3 *vt 1–8* is to be transmitted as an intermediate symptom to node #3 for further diagnosis, since its back neighbor lies across the nodes boundaries.

Cycle 6–9: The local diagnosis proceeds as above in *cycles 4–5* except this time first the back neighbors of the state MA(1.3)/2 *vt 1–8* are expanded, representing communication between node #1 and #2; and then the back neighbors of state MA(1.3)/4 *vt 1–8*, representing the communication between nodes #1 and #4.

During cycle 6 the transmission processor transmits the intermediate symptom MS(1.5)/3 *vt 1–8* to node #3. Similarly, during cycle 8 the intermediate symptom MS(1.7)/2 *vt 1–8* is transmitted to node #2.

Cycles 10 - 11: The local diagnosis processor has no other states in its queue, and waits for the results of the intermediate symptoms that it has sent.

The transmission processor transmits the intermediate symptom MS(1.9)/4 *vt 1–8* to node #4.

Cycle 12: The reception processor for node #1 receives an intermediate symptom, MS(2.10)/1 *vt 1–4* to be diagnosed from node #2. This message represents the communication of a *vt 1–4* hypothesis from node #1 to node #2.

Cycle 13: The local diagnosis processor evaluates MS(2.10)/1 *vt 1–4* to false by examining the executed knowledge sources at node #1 and seeing that no CKS's executed.

Cycle 14: Diagnosis continues by expanding the back neighbor of MS(2.10)/1 *vt 1–4*. This is the state MESSAGE-EXISTS(1.14)/ *vt 1–4*. This state is found to be false since no message is found to exist corresponding to the *vt 1–4* hypothesis.

Cycle 15: Diagnosis continues by trying to find out whether a *vt 1–4* hypothesis that should have been sent exists or not. Thus, the back neighbor of the MESSAGE-EXISTS state is VT(1.15)/ *vt 1–4*. This state is found to be false by looking at the DVMT data structures.

Cycles 15–21: The processing during these cycles is similar to that during cycles 2–10, except that here it is the *vt 1–4* hypothesis that we are concerned with, as opposed to the *vt 1–8* hypothesis.

During cycle 20, the reception processor receives the result of an intermediate symptom, MS(1.5)/3 *vt 1–8*, that it had transmitted to node #3 to diagnose in cycle 6. It concerns the communication by node #3 of the *vt 1–8* hypothesis. Node #3 diagnosed and identified the fault causing this false MS state – it was missing the communication knowledge sources. Hence, because an underconstrained MS state was sent back as confirmed false, all pending and future MS states concerning communication from node #3 to node #1 are accounted for, and a false path value can be propagated back from all such states.

During *cycle 21* the reception processor similarly receives a reply from node #3 regarding the diagnosis of the intermediate symptom MS(1.18)/3 *vt 1–4* representing the communication of a *vt 1–4* hypothesis from node #3 to node #1. For the same reason as above, node #3 returns a false path value associated with this state. Node #1 propagates this value backwards.

During this cycle, the reception processor also receives a request from node #4 to diagnose the intermediate symptom MS(4.19)/1 *vt 7–8* regarding the communication of a *vt 7–8* hypothesis from node #1 to node #4.

Cycle 22: The local diagnosis processor now starts diagnosing the only state in the

queue, MS(4.19)/1 *vt 7-8*. This is found to be false because no executed communication knowledge source is found.

During this cycle the reception processor receives a request from node #3 to diagnose the intermediate symptom MS(3.20)/1 *vt 5–8*, representing the communication of a *vt 5–8* hypothesis by node #1 to node #3. It places this symptom in its queue, but with a lesser rating than the current state being diagnosed.

It also receives a result from node #2 regarding the intermediate symptom MS(1.7)/2 *vt 1–8*. As in the case of node #3, node #2 finds that it is missing the communication knowledge sources, and sends back an underconstrained MS state with a false path value. Thus, all MS states representing communication from node #2 to node #1 are accounted for, and false pathvalues are propagated backwards.

Cycles 23-29: similar to cycles 13 to 21 except that here it is the *vt 7–8* hypothesis that is being investigated.

Any MS states corresponding to communication from nodes #2 and #3 are accounted for as false because of the false value for the underconstrained MS sent to node #1 from these two nodes. The only pending states whose truth values are yet to be assigned are the MS requests that node #1 has made to node #4. A few cycles later, when node #4 replies with a value of false for an underconstrained MS object, all these pending values will be determined, and an overall false pathvalue will be propagated upwards towards PT(1.1)/ *pt 1–8*, accounting for the detection of no pattern tracks at this node. **Diagnosis for**

Node #2

Node #2 is in the upper right corner of the environment and receives signals in locations 5 through 8 as its input data. It generates a pattern track integrating these signals but, due to the lack of communication, does not receive the *vt* hypotheses that would allow it to extend its 5–8 track to the full spanning *pt 1 8* hypothesis and thus derive the final pattern track answer.

Cycle 1: As in Node #1, diagnosis begins with the lack of the spanning *pt 1–8* hypothesis. As before, since PT(2.1)/ *pt 1-8* is false, its back neighbors are expanded.

Cycle 2: This is analogous to the second step of diagnosis in Node #1, but this time there are shorter *pt* tracks, so some PT states are created. Specifically, one true and one false state, with the attached objects representing the existing pt segment 5–8 (state PT(2.2)/ *pt 5–8*) and the missing *pt 1–4* segment (state PT(2.2)/ *pt 1–4*). As before, a VT state is instantiated with its associated abstracted object representing the desired *vt 1–8* hypothesis. Diagnosis continues in a best-first manner, with the expansion of the back neighbors of the highest rated state. In the centralized DM it has been empirically determined that for this cluster, for three nodes linked by an AND connection, it is better to proceed along the false-false path rather than the false-true path. This is one of the

factors determining the ratings of states to expand at this point. Thus, the PT(2.2)/ *pt 1–4* state has the highest rating, followed by the PT(2.2)/ *pt 5–8* state, followed by the VT(2.2)/ *vt 1–8* state.

Cycle 9–10: The back neighbor of the PT(2.2)/ *pt 1–4* state is expanded. This is the state VT(2.3)/ *vt 1–4*. Diagnosis proceeds as in cycles 2–9 in node #1, except that here it is a *vt 1–4* hypothesis that is being considered.

During cycle 9 the reception processor receives a request from node #1 to diagnose the causal pathway arising from the intermediate symptom MS(1.7)/2 *vt 1–8*, representing the communication of a *vt 1–8* hypothesis by node #2 to node #1. Similarly, in cycle 10 the reception processor receives requests from nodes #3 and #4, to diagnose the intermediate symptoms MS(3.8)/2(vt 5–8) and MS(4.8)/2 *vt 1-2* respectively. These intermediate symptom states are rated and placed in the queue.

Cycle 11: At this stage node #2 finds it advantageous to suspend its current exploration, and proceed with diagnosing the intermediate symptom MS(3.8)/2 *vt 5–8* that it received from node #3. The factor that mainly contributes to this decision is that on receiving this request this processor had made a preliminary evaluation of the relevance of this request and found that it pertained to a *vt 5–8* hypothesis, which it knew to be a related problem to its own investigation since it had already confirmed this as existing earlier. It knows that a fairly low level of effort might locate the fault, since it lies in some lower cluster between the true and the false states.

Cycle 12: During BCT it is found that the state MESSAGE-EXISTS(2.12)/ *vt 5–8*, corresponding to the object *vt 5-8*, is in fact true. This leads to tracing the states in between these two false and true states, and the lower Communication Ksi Scheduling cluster is therefore instantiated next.

Cycle 13: The front lower neighbor of the true state, MESSAGE-EXISTS(2.12)/ *vt 5–8*, is COMM-KSI-RATED(2.13). This is expanded next. It is found to be false.

Cycle 14: The back neighbor of the state COMM-KSI-RATED(2.13), the state COMM-KS-EXISTS(2.14) is expanded. This is a *primitive* state and represents the identified failure: the missing communication knowledge sources *hyp-send:vt* and *hyp-reply:vt*.

At this point FCT is invoked to simulate the effects of the identified failures on system behavior and thereby account for both pending and future symptoms caused by this failure. Such future symptoms are any states that are in some way effected by the false COMM-KS-EXISTS(2.14) state. In terms of the system behavior, these are any events in some way dependent on the existence of these two CKS's. In this case, these would be any events regarding the scheduling or execution of these CKS's and any messages relying on these CKS's for transmission. The affected states are any states forward of the COMM-KS-EXISTS(2.14) state, where forward means both at the same level in the model hierarchy and at any higher levels.

C. Example: Model Use in Diagnosis

FCT works by propagating forward an underconstrained abstracted object which represents the whole class of objects affected by the fault. At each step in the diagnostic cycle any abstracted objects associated with pending symptoms are checked to see whether they overlap with the newly created underconstrained object. If so, then their pending symptom has been successfully explained by the identified fault and need no longer be diagnosed. Now let's get back to the example.

Cycles 15-20: The FCT first constructs underconstrained objects representing the missing communication knowledge sources. The corresponding state is then created, COMM-KS-EXISTS(2,15). (In the diagrams, the states attached to underconstrained objects are represented by two concentric circles, instead of the single circle). FCT continues simulating the fault's effect on the DVMT by expanding the front neighbors of this state. This goes on until in cycle 19, the underconstrained MS(2,19)/ *vt* state is found to be false. The resulting underconstrained object represents the class of objects affected by the missing CKS's, namely all message hypotheses at the *vt* level. This time the underconstrained MESSAGE-OBJECT does overlap with the MESSAGE-OBJECTS associated with the various pending MESSAGE-SENT symptoms. As a result, all these pending MS symptoms are accounted for by the identified faults, the missing CKS's, and the results can now be transmitted back to the other nodes who had requested node #2 to diagnose those intermediate MS symptoms.

During cycle #20 the reception processor receives the result of an intermediate symptom, MS(2,6)/3 *vt 1-4*. It concerns the communication by node #3 of the *vt 1-8* hypothesis. Node #3 diagnosed and identified the fault causing this false MS state – it was missing the CKSs. Hence, because an underconstrained MS object was sent back as false, all pending and future MS states concerning communication from node #3 to node #2 are accounted for, and a false pathvalue can be propagated back from all such states.

Cycles 21-25: Node #2 sends out all the pending MS symptoms back to the nodes that had sent them to be diagnosed by node #2. All of them have confirmed false path values.

The local diagnosis processor continues processing the highest rated state in the queue, PT(2,2)/ *pt 5-8*, which represents the pt 5-8 object that does exist. When a true state is encountered during BCT, the DM must look for the problem in between the true and the false states, at a lower cluster of the model, which represents the states occurring in between the true and the false state. In this case the DM must find out why the the *pt 5-8* hypothesis was not extended to the *pt 1-8* hypothesis. The DM expands the lower cluster, the Ksi Scheduling Cluster, and finds that the reason the *pt 5-8* hyp was not extended is due to the missing *pt 1-4* hyp. The diagnostic pathways merge at this point at the state PT(2,2)/ *pt 1-4*, which represents the missing *pt 1-4* segment. This state was already diagnosed earlier.

Cycle 26-27: The highest rated state on the queue is VT(2.2)/ *vt 1-8*, which is expanded next. The back neighbors are VT(2.27)/ *vt 1-4* representing the *vt 1-4* object, which is merged with the existing state VT(2.3)/ *vt 1-4* already expanded in cycle 3; the state VT(2.27)/ *vt 5-8* representing the *vt 5-8* hyp, which does exist at this node; and the three MA states representing possible communication received from the other three nodes. Since VT(2.27)/ *vt 1-4* has merged with VT(2.3)/ *vt 1-4*, the highest rated state is VT(2.27)/ *vt 5-8*.

Cycles 28 onwards: the processing continues similarly for a few more cycles.

After some more cycles, node #2 receives replies from nodes #1 and #4 regarding the MS intermediate symptoms that it sent to them to be diagnosed. It then has the values of all the requested symptoms, and it propagates all the false values upwards. Thus, the initial symptom PT(2.1)/ *pt 1-8* does receive a false pathvalue propagated upwards to it, and the missing spanning PT hyp is accounted for.

Diagnosis for Node #3

Node #3 is in the lower left corner of the environment and receives data from locations 1 through 4 from its sensor. It therefore generates a pattern track integrating these locations but cannot produce the final answer because it is missing the track integrating locations 5 through 8. The diagnosis for this node is similar to the diagnosis for Node #2. The same parts of the model are instantiated and same types of failures are identified. The difference is that here it is the track 5–8 that is missing, not 1-4 as was the case in Node #2.

Diagnosis for Node #4

Node #4 is in the lower right corner of the environment and receives data in locations 3 through 6. It integrates these locations into a pattern track linking times 3 through 6 but does not produce the overall answer because it never receives the track segments 1–2 and 7–8. Diagnosis proceeds as before, generating an explanation in terms of the missing local data in locations 1–2 and 7–8, and producing a number of false MESSAGE-SENT symptoms which are sent to other nodes for diagnosis.

At the end of a few dozen cycles, all the missing spanning pattern tracks are accounted for. The cause of the problem has been identifed as the missing CKSs at each node.

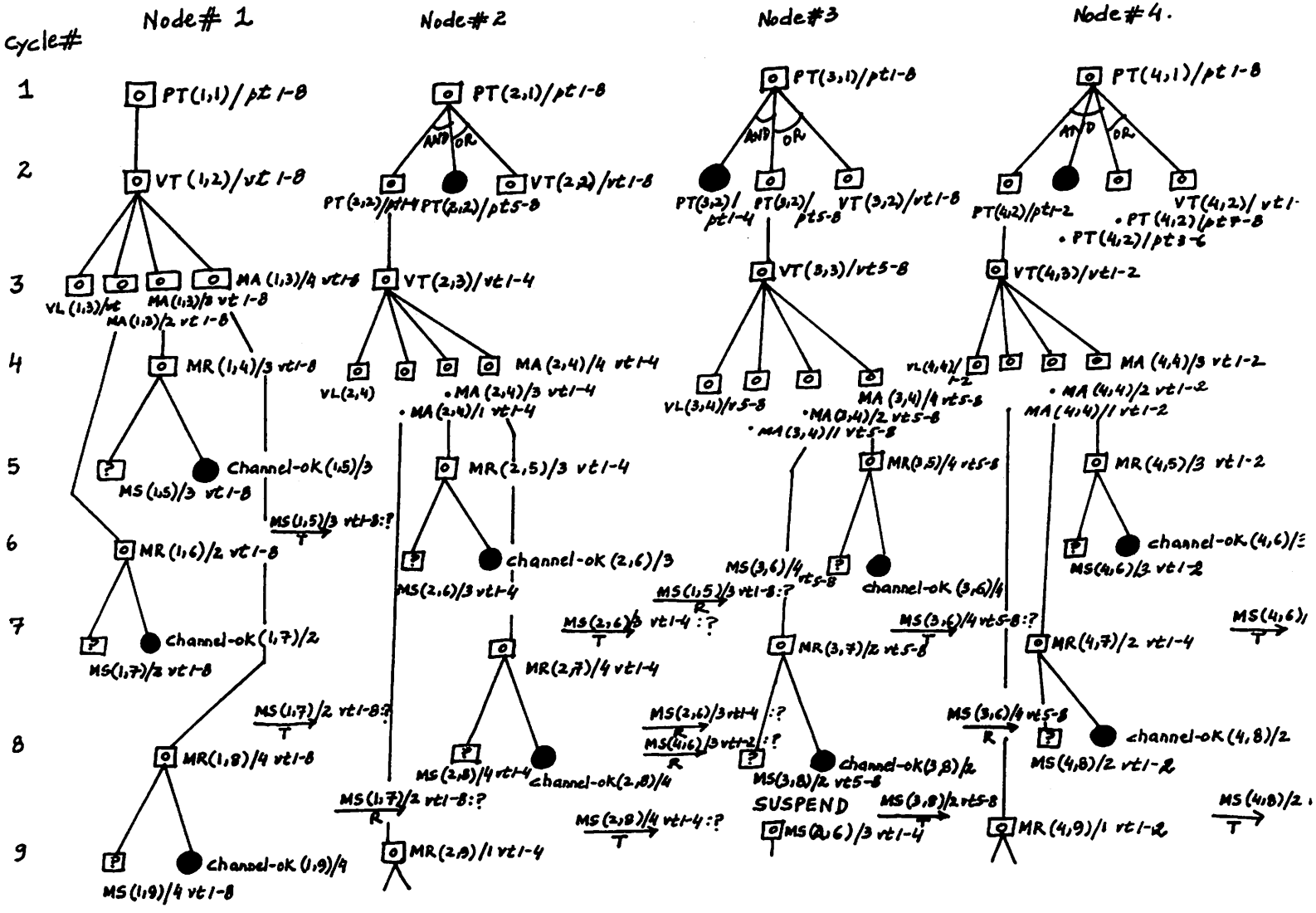
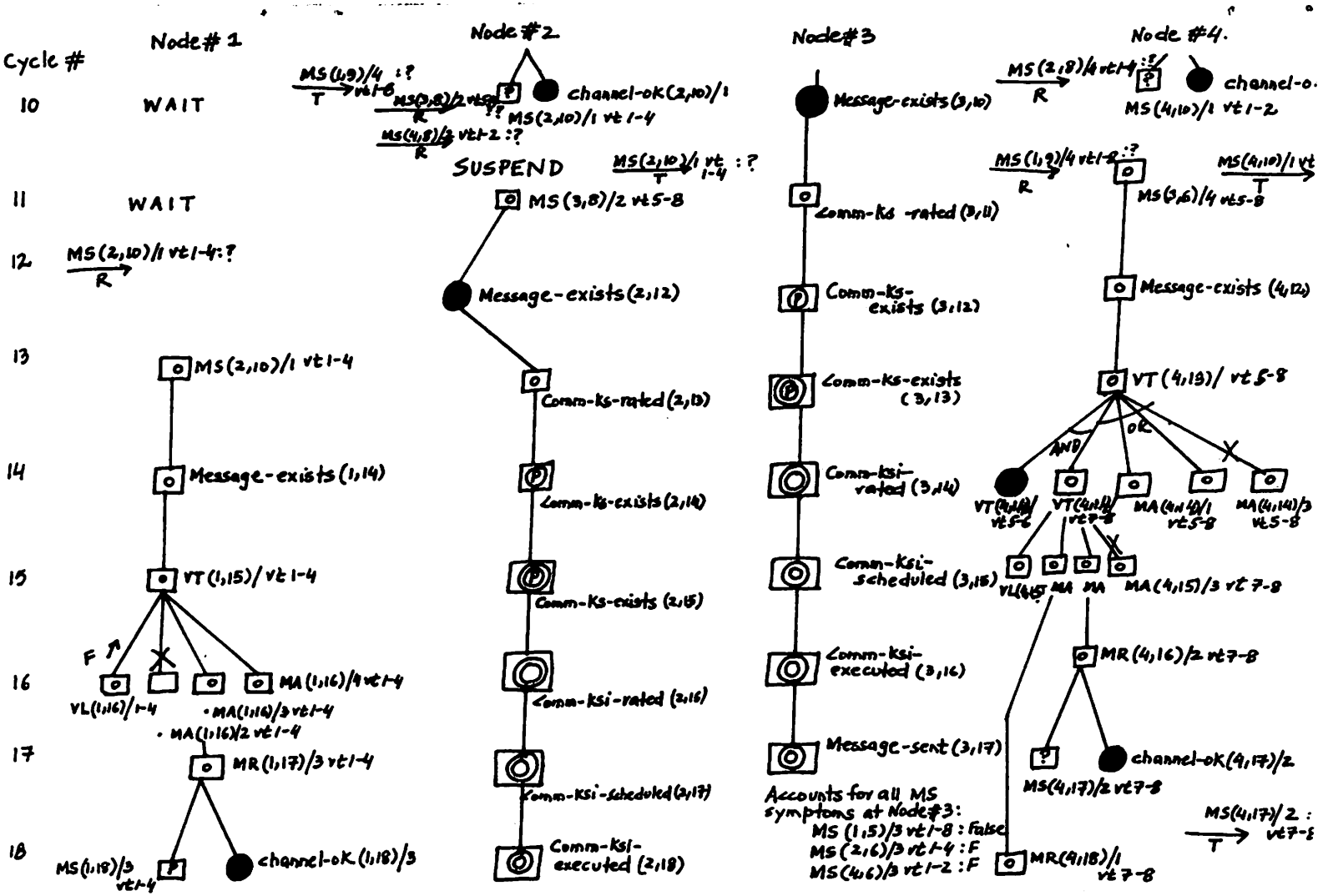
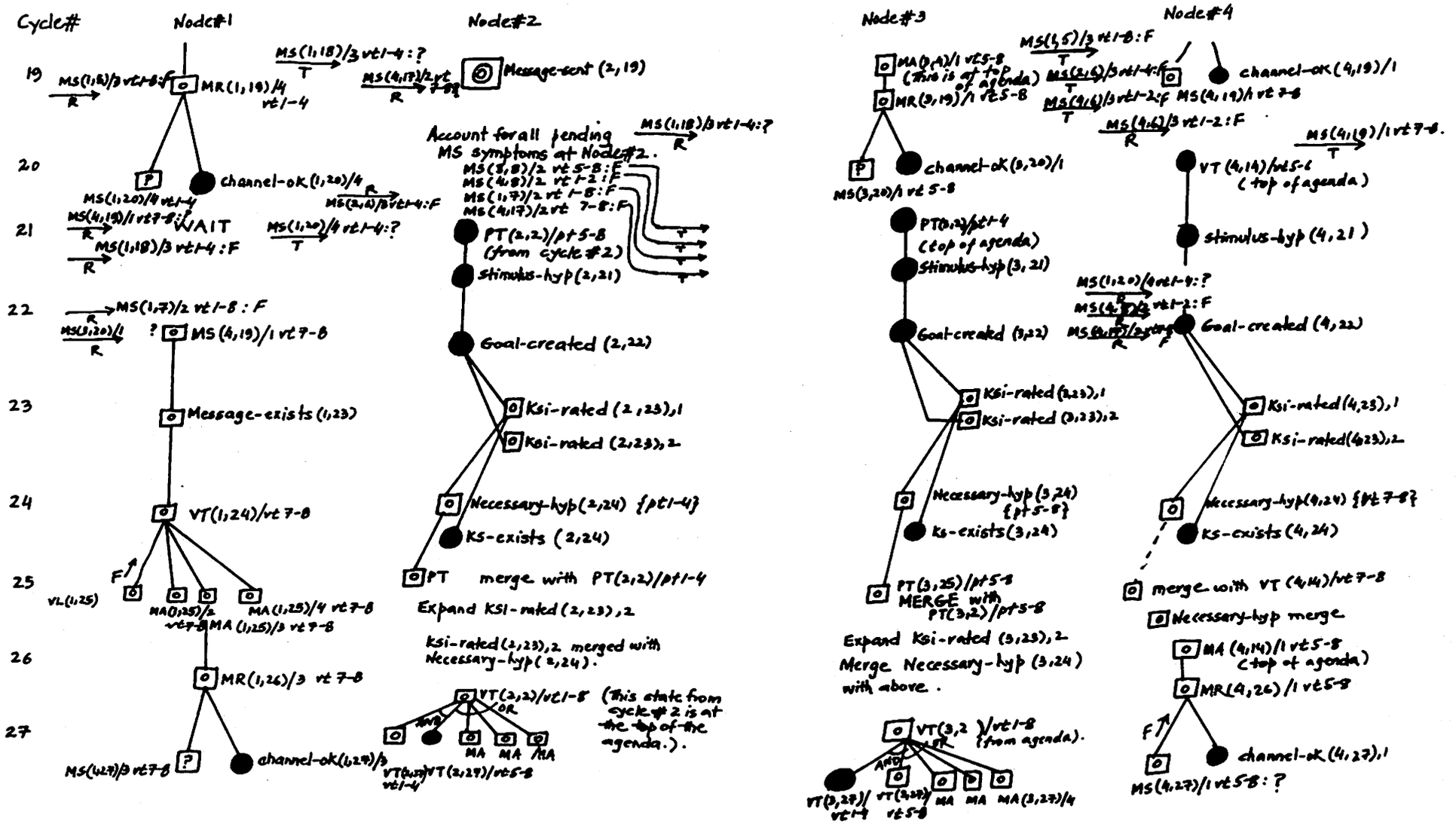


FIG. 14: A cycle by cycle Depiction of the Causal Pathways Traced by the 4 nodes.



... FIG. 14 (continued).



... FIG. 14 (continued).