Multi-Signal Flow Graphs : A Novel Approach for System Testability Analysis and Fault Diagnosis*

Somnath Deb Krishna R.Pattipati Vijay Raghavan Mojdeh Shakeri Roshan Shrestha
Dept. of Electrical and Systems Engineering,
U-157,University of Connecticut,, Storrs, CT 06269-3157.
Phone (203)-486-2890 Fax (203)-486-2447 E-Mail: krishna@sol.uconn.edu

Abstract

In this paper, we present a comprehensive methodology for a formal, but intuitive, cause-effect dependency modeling using multi-signal directed graphs that correspond closely to hierarchical system schematics and develop diagnostic strategies to isolate faults in the shortest possible time without making the unrealistic single fault assumption. A key feature of our methodology is that our models lend naturally to real-world necessities, such as system integration and hierarchical troubleshooting.

1 Introduction

Diagnosis is the process of identifying the *cause* of a malfunction (fault) by observing its *effects* at various monitoring (test) points in a system. As technology advances, there is a significant increase in the complexity and sophistication of systems. Moreover, integration and miniaturisation have sharply limited access to test points. Thus, the number of possible causes have increased while reduction in monitoring points have resulted in reduced fault observability, making it increasingly difficult to troubleshoot these systems. Consequently, system maintenance presents formidable challenges to manufacturers and end users. In this vein, Computer-aided design techniques for system modeling and diagnosis are of paramount significance.

Maintenance and design have traditionally been two separate engineering disciplines with often conflicting objectives — maximizing maintainability versus optimizing performance, size and cost. Testability has been an ad hoc, manual effort, in which maintenance engineers attempt to identify an efficient method of troubleshooting for the given product, with no control over product design. However, poor fault observability in complex systems have driven up the life-cycle maintenance cost of products to over 3–10 times that of the manufacturing cost. Evidently, significant savings in the total cost of a product can be achieved by improving the testability and maintainability of products. Testability must be engineered into the product at the design stage itself, so that optimal compromise is achieved between system maintainability and performance. This process of refining a system design to improve testability is termed Design for Testability (DFT), and is now a requirement in most defense projects. To maximize its impact, DFT must be performed at all stages of the design – from schematics – to design of subsystems – to system integration.

In this paper, we present a modeling methodology, and a software tool implementing it, that is capable of performing testability analysis of a system in every stage of its design. We envision the use of the methodology in all the stages of a product life cycle :

- In the concept phase of a design, the modeling technique enables a designer to perform system level DFT analysis by using a hierarchical generalized dependency model that is closely related to schematics. This step will also allow designers to allocate testability resources to the various subsystems for optimizing system testability.
- As subsystem designs become available, direct interfaces to the CAD/CAE databases (EDIF or VHDL description of the subsystem) will enable verification of the testability of individual subsystems.
- Using the hierarchical dependency model, a designer can integrate the subsystem models into a hierarchical model of the complete system. The designer can perform testability analysis of the system and determine if the testability goals are met.
- The analysis techniques identify enhancements that can be incorporated into a design to improve the testability of the system.

*Research supported in part by the Dept. of Economic Development of the State of Connecticut under the Yankee Ingenuity Initiative, Sikorky Aircraft and Qualtech Systems, Inc.

- The test sequencing algorithms generate near-optimal fault isolation strategies for the system, which can be used by maintenance personal in the field. This ensures that the calculated testability figures of merit for the system are indeed achieved.

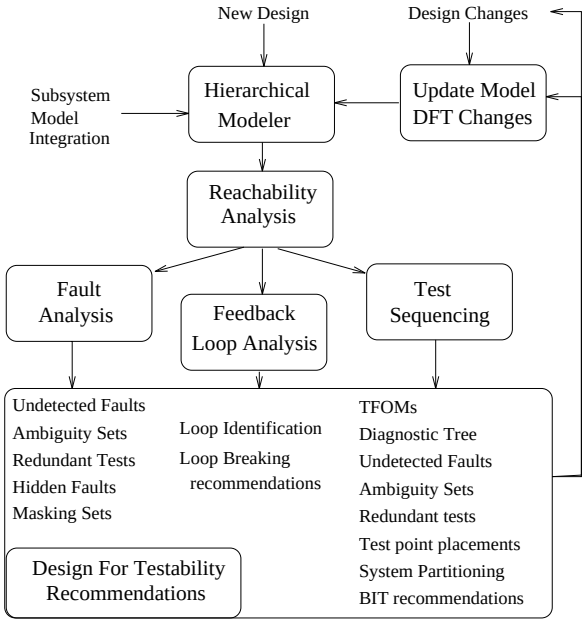

Figure 1: Designing *completely* testable systems using **TEAMS**

TEAMS, Testability Engineering And Maintenance System [1], is an X-windows based software tool, that integrates the methodology and the algorithms in an easy-to-use graphical user interface. **TEAMS** has been used for testability analysis of large systems containing as many as 50,000 faults and 45,000 test points. **TEAMS** minimizes the life-cycle cost of a system by aiding the system designer and test engineer in embedding testability features, including "built-in-test" requirements, into a system design; and by aiding the maintenance engineer by developing near-optimal diagnostic strategies (see Fig. 1). **TEAMS** is used to: (i) model individual subsystems and integrate them into system models, (ii) generate near-optimal diagnostic procedures for a variety of realistic options, and (iii) analyze and quantify testability of systems and subsystems, visually pinpoint the diagnostic inefficiencies of a system, and make recommendations towards the design of *completely* testable systems.

The paper is organized as follows. In section 2, we present the multi-signal directed graph modeling methodology, which corresponds closely to hierarchical system schematics. In section 3, we discuss the static analysis algorithms that assess the inherent testability of a system, pinpoint testability deficiencies and suggest improvements, by analyzing the topology of the system. This is followed in section 4 by the test sequencing algorithms and extensions required in testing algorithms to exploit the capabilities of multisignal models to meet real-world demands of users.

2 Multisignal Modeling

2.1 Existing Modeling Approaches

A review of the literature suggests a spectrum of modeling approaches for diagnosing faults in complex systems: quantitative (e.g., numerical simulation, ordinary differential equations), qualitative, structural and dependency models (see Fig. 2).

Quantitative models require the complete specification of system components, the state and observed variables associated with each component, and the functional relationships among the state variables [2]. However, the precise information required by these models is typically not available for complex systems and is prohibitively expensive to obtain.

Qualitative models, or simplified quantitative models, represent a physical system in terms of simple qualitative algebraic constraints and/or qualitative differential equations to simulate its qualitative behavior [4, 3]. Even qualitative simulation is too expensive for large systems (see Table 1). Both techniques require extensive modeling efforts and need information that is usually not available in the early stages of a design.

Structural models represent the connectivity and failure propagation direction in the form of a directed graph, which corresponds closely to the schematic of the system. Analysis based on structural models is simple and fast, and, consequently, can be used for large systems. Moreover, there is a direct correspondence between the nodes in a structural model and the modules of the real system, making it easy to verify these models. However, structure does not always imply function; typically, many complex functional dependencies are embedded in simple block diagrams. Thus, analysis based on structure alone is crude, and often leads to wrong diagnostic conclusions.

Dependency models represent the cause-effect relationships in the form of a directed graph, and are the primary modeling techniques employed in the current testability analysis tools. It is also referred to as *inference* modeling in the test community [5]. However, dependency models can deviate significantly from structure as more and more complex dependencies are modeled.

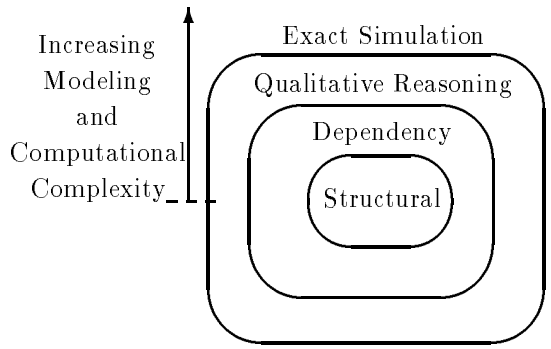

Figure 2: Spectrum of Modeling Approaches for System Fault Diagnosis

Modeling Technique	State Generation	Diagnostic Strategy Generation
Qualitative Reasoning (QRS)	1.5 hours	3.83 hours
Dependency Model (**TEAMS**)	needs true value simulator *once*. ($\sim$ 5 min.)	22 seconds (speedup = 62.7)

Table 1: Computational Requirements of Qualitative Reasoning and Dependency Models (based on blade deice system of a Helicopter with 58 components, 400 failure modes and 660 tests on a Sparc 10.)

2.2 Types of Failures

A failure is defined as any abnormal behavior of a component or of a system. We classify failures into two distinct categories :– *functional* failures and *general* failures. The two different types of failures are best illustrated via a simple example. Consider a *lossless* (passive) bandpass filter consisting of an inductor and a capacitor. If a fault in the inductor or capacitor causes a deviation in the center frequency or the Q-factor, it is considered a functional failure, i.e., a fault that affects the function it was supposed to perform. On the other hand, if the fault is a short-circuit that causes the output power to be zero (i.e., a *lossless* filter causes a *power-loss*!), this is a general failure, that is, a catastrophic failure affecting attributes beyond its normal functioning.

The four modeling techniques described in the preceding subsection differ in the way they model these failures. In qualitative and quantitative modeling techniques, the functioning of a system is modeled in great detail. Thus, they have excellent capabilities in isolating functional failures. General failures are handled as special cases of functional failures. However, since their reasoning is based on simulating the real system, the reasoning process is extremely slow and is unsuitable for large-scale systems (see Table 1).

For diagnostic purposes, we only need to model how a fault (or cause) propagates to the various monitoring points. Thus, it is sufficient to model the system in its failure space. In structural and dependency modeling, the system is modeled in terms of first-order cause–effect dependencies, i.e., how a faulty node affects its immediate neighbors. Higher-order dependencies can be inferred from first-order dependencies. Thus, dependency modeling captures the minimum necessary information for testability analysis and is the only technique that has been applied to large-scale systems.

Structural models equate the connectivity to dependency. They model general failures only, totally ignoring functional failures. Thus, if component A fails, it affects *all* attributes of all components depending on it. This obviously does not have to be true. Hence, structural models are often called "worst case" models and have poor diagnostic resolution.

Dependency modeling is a refinement on the structural modeling approach, where *failure modes* are added in an effort to model functional failures. In the filter example, the failure modes could be "out of tolerance", causing a functional failure, and "short circuit", causing a general failure. However, the failure modes are modeled based on user experience, expert input, or heuristic rules. Consequently, they do not represent an accurate or complete list of possible failure modes. *These failure modes of dependency modeling should not be confused with the failure types of multisignal modeling.* The failure modes in dependency models classify the physical faults, whereas functional and general failures in multisignal model classify the effects on system functioning. Thus, instead of enumerating the possible (range of) values of a resistor in the event of a fault in the resistor, we identify its possible effects on the function of the system. It may be tempting to map open-circuit and short circuit to general failure, and out-of-tolerance to functional failure, but this may not always be true[1].

The structural distortion in dependency models stems from the mapping of a multi-dimensional attribute or "signal" space of a physical system into a single dimensional (dependency) space. Conceptually, the structural model defines the paths along which nodes of the graph affect each other (i.e., a potential multi-signal dependency). Existing dependency modeling techniques as-

[1] In example 3, described later in section 2.5.3, a short-circuit in R_3 is a functional failure, affecting only the dc-offset of amplifier A_1.

sume dependencies to be binary. Thus, dependencies involving multiple signals are modeled as multiple single dependencies, one for each signal, just as any integer can be encoded as a string of binary digits. This is done by replicating nodes for each signal dependency, and consequently deviating from the structure. For example, a simple resistor can be split into three failure modes - open-circuit, short-circuit and out-of-tolerance. Since this deviation stems from the judgement of the modeler based on some *local* reasoning, such dependency models are subjective with limited validity. This limitation, and the resultant validation problem, has rendered dependency modeling into an *art* and cast aspersions on the usefulness of the results. Test program developers, who inherit these models from the modeler, are unable to validate these models, and are either unwilling or unable to use the results in developing test programs.

2.3 The Multisignal Dependency Modeling solution

It is evident from the preceding discussion that none of the modeling techniques are adequate for accurate fault diagnosis in large scale systems. In spite of the limitations, these modeling techniques have been successfully applied, although at a steep price, to many defense projects, to produce systems that are easy to maintain. Structural modeling is easy, but lacks the required accuracy. Dependency modeling, in an effort to add the functional failure information, also added structural distortion, subjective user input and, consequently, immense model validation problems. The other techniques require highly knowledgeable expert user input. Naturally, these techniques are not cost effective and are not applied, unless absolutely mandated (and paid for by some defense contract)!

In the following, we propose a modeling approach that lets a test engineer layer in the functional failure information on a structural model in an intuitive, easy-to-follow methodical way, without the quirks of dependency modeling. Our proposed solution to the structural distortion problem is to capture the signals (or attributes) modified by each component, and the signals (or attributes) detected by each test point. Formally, a multi-signal model consists of:

- a finite set of components $C = \{c_1, c_2, ..., c_L\}$ and a set of independent signals $S = \{s_1, s_2, ..., s_K\}$ associated with the system;
- a finite set of n available tests $T = \{t_1, t_2, ..., t_n\}$;
- a finite set of P available test points (or probe points) $TP = \{TP_1, TP_2, ..., TP_P\}$;
- each test point TP_p is associated with a set of tests, $SP(TP_p)$;
- each component c_i affects a set of signals $SC(c_i)$,
- each test t_j checks a subset of signals $ST(t_j)$, and
- the digraph $DG = \{C, TP, E\}$, where E denotes the set of directed edges specifying the structural connectivity of the system.

Conceptually, a multi-signal dependency model is akin to overlaying a set of (single-signal) dependency models on the structural model, and, hence, the model corresponds closely to the schematics of the system. Note that the "signals" correspond to the independent units in the system transfer function, or the distinct attributes that constitute the functional specifications of a system. Consequently, the number of possible signals is a small countable set and a failure in one signal, by definition of independence, does not affect the other signals. Also, since the models correspond closely to the structure, model validation and integration of individual models into system models is greatly simplified. This significantly enhances the job of the modeler.

Multisignal modeling has the benefit of capturing the necessary useful and important knowledge about the system for fault diagnosis without being bogged down by unnecessary details (which drive up the model generation cost, as in exact simulation and/or qualitative reasoning) and computationally expensive simulations and/or reasoning techniques (which makes them impractical for large-scale systems applications). Furthermore, this modeling approach does not require the explicit knowledge of failure modes in a system. This means that the modeling approach enables the detection and isolation of unanticipated failure modes. Moreover, failure modes and effects analysis can be performed by specifying the signal-failure mode association for each component, if necessary. This approach thus models all the information captured by dependency models, without the added complexity of failure modes and structural distortions.

Multisignal dependency modeling technique represents the system in the failure space. Hence, only the nature of dependency (i.e., the signal) needs to be modeled. This is in contrast to qualitative and quantitative modeling schemes, which require costly simulations and state generation. Furthermore, for troubleshooting purposes, it is unnecessary to model the exact quantitative relationships. In order to illustrate the assertion, consider a cascade of four amplifiers, having gains of 2,3, 4 and 5, with an overall gain of 120. If, due to a fault, the new gain is 60, the first stage, with a design gain of 2, should not

necessarily be implicated; the gain of any of the stages may have been reduced due to a functional failure. Thus, when the same attribute is modified by multiple components, quantitative relationships convey little, if any, information. For the same reason, signals should not be *abused* to define two signals, such as "voltage high" and "voltage low", in an effort to obtain higher diagnostic resolution. It is imperative that the signals be the basic independent variables that describe a system, and not be directly derivable from each other.

2.4 Three step guide to multisignal modeling

In the following, we provide a three-step procedure for multisignal modeling that should be adequate for most modeling needs :

1. Enter the structural model, schematic model or a conceptual block diagram. In **TEAMS**, the structural model can be automatically generated from VHDL structural models, EDIF netlists, or directly entered via the graphical user interface.

2. Add signals to the modules and test points. The set of signals can be identified from the functional specification or from the independent variables in the transfer function. For example, the signal specification of a power amplifier will include output distortion, harmonic distortion and power output. In general, any unique attribute will have an associated signal. For example, in a bus with multiple independently addressable devices attached to it, the address of each device will serve as a signal (see example 1 in section 2.5.1).

3. Update model for special situations. In the following, we identify a few special situations (not necessarily exhaustive), and the corrections necessary for them. All these correction mechanisms are available in **TEAMS**.

    - If a system has built in redundancy, (e.g., both A and B must fail for a system to fail), configure the redundant components using AND nodes.
    - If a system has different modes of operation, use special purpose nodes called SWITCHes, to model them.
    - If a system has components with built-in-self-test (BIST), enable BIST in property options of the component in the **TEAMS** model. Note that BIST can also be modeled via unique signal names. A BIST spanning multiple components is also called a "component test" in dependency modeling; it can be modeled via a unique signal.
    - If a system has replaceable digital integrated circuits, model them with their equivalent models. The equivalent models [6] are simplified models of chips that capture the necessary dependency information to detect faults in (but not isolate faults within) a chip.
    - In rare cases, a system may have dependencies that cancel out. This is equivalent to "DON'T CARE" conditions in digital circuits. These dependencies must be identified and removed.

2.5 Multisignal Modeling Examples

In the following, we illustrate the multisignal modeling concepts using three examples. Each component has two types of failure – *general* and *functional*. General failures are explicitly modeled by signal G. The functional failures are mapped to the set of affected signals.

2.5.1 Example 1: A simple bus system

This example illustrates how multisignal modeling helps preserve the relationship to original system topology. Consider a simple bi-directional (SCSI) bus system, with a controller and five independently addressable devices connected to it (Fig. 3). In normal mode of operation, the controller activates one device at a time, and transfers data to and from the device[2] independent of the other devices on the bus. Thus, the bus acts as a point-to-point link between the controller and each of the devices. This is modeled by creating a signal for each unique device address, and attaching it to the respective devices. Thus, if there are errors only while the controller is communicating with device A, the suspected components would be device A and the controller itself. This is equivalent to a functional failure in signal A. However, if the cabling and/or termination is improper, or any of the devices fail and "hang" the bus (a general failure), all communications will fail! Indeed, this is a tricky system to model using traditional (single-signal) dependency model, because the dependencies are point-to-point for functional failures, but all-to-all for general failures. Table 2 presents the binary dependency matrix (where 1 denotes a cause-effect relationship) for this system.

[2] To test a device, the controller would send specific commands to the device and compare the received data to known true responses.

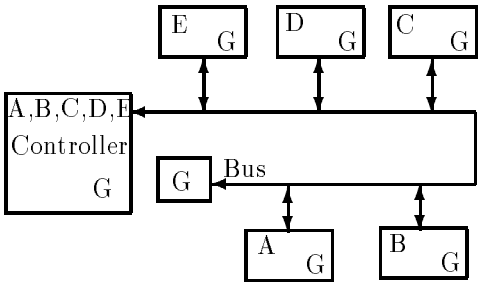

Figure 3: Simple Multisignal Model of a bus system

Possible Cause(s)	Failure in device test				
	A	B	C	D	E
Controller (G)	1	1	1	1	1
Controller (F)	1	1	1	1	1
Bus (G)	1	1	1	1	1
Device A (G)	1	1	1	1	1
Device A (F)	1	0	0	0	0
Device B (G)	1	1	1	1	1
Device B (F)	0	1	0	0	0
Device C (G)	1	1	1	1	1
Device C (F)	0	0	1	0	0
Device D (G)	1	1	1	1	1
Device D (F)	0	0	0	1	0
Device E (G)	1	1	1	1	1
Device E (F)	0	0	0	0	1

Table 2: The Dependency Matrix for the bus system example

2.5.2 Example 2: A Cassette Player

This example illustrates how multisignal modeling can be used to increase the diagnostic resolution by adding signal attributes on a structural model. Consider a cassette player, consisting of power-supply, tape head, pre-amplifier, power-amplifier and a three-way speaker system. It is assumed that there are no internal test points in the system, and, hence, its performance can only be monitored at its outputs, i.e., by listening to the sound from the three-way speaker, and by looking at the Power ON LED. A multi-signal model for the cassette player system is shown in Fig. 4, where the signals s_1, s_2, s_3, s_4, and s_5 correspond to Treble, Bass, Midrange, signal to noise ratio (SNR) at 1 Watt nominal output, and harmonic distortion at rated power, respectively. As before, each component is associated with a list of signals that it affects (including "G" for general failure). The output music and power-on LED (the test points) are associated with a list of signals that they monitor. Further, assume

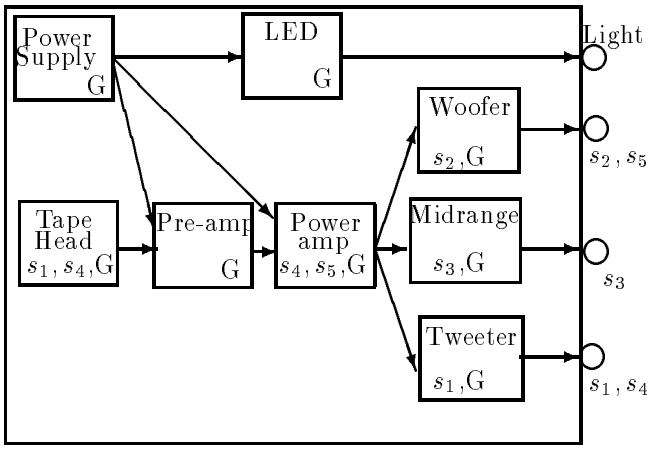

Figure 4: Simple Multisignal Dependency Model of a Cassette Player System

that we attach special instruments to measure SNR (s_4) to the tweeter output and a distortion meter that checks s_5 to the woofer. In addition, the base (s_1), treble (s_2) and midrange (s_3) are monitored at the respective speakers. Table 3 presents a binary matrix that captures the overall cause-effect or dependency matrix for this system.

	Tweeter		Woofer		Mid.	Light
	s_1	s_4	s_2	s_5	s_3	
Power (G)	1	1	1	1	1	1
LED (G)	0	0	0	0	0	1
Tape Head (G)	1	1	1	1	1	0
Tape Head (F)	1	1	0	0	0	0
Pre-amp (G)	1	1	1	1	1	0
Power-amp (G)	1	1	1	1	1	0
Power-amp (F)	0	1	0	1	0	0
Woofer (G)	0	0	1	1	0	0
Woofer (F)	0	0	1	0	0	0
Midrange (G)	0	0	0	0	1	0
Midrange (F)	0	0	0	0	1	0
Tweeter (G)	1	1	0	0	0	0
Tweeter (F)	1	0	0	0	0	0

Table 3: The Dependency Matrix for the Cassette player example

2.5.3 Example 3: An Amplifier/Filter System

In this example, we present the multisignal model of an amplifier/filter system. This example was solved using (single-signal) dependency modeling in [7, 8]. Here, we present the equivalent multisignal model for it. The circuit consists of an amplifier (of gain 2), followed by an

RC low pass filter (f_c=1 khz) and a buffer. Thus, the signals associated with the amplifier are s_1 (gain), s_2 (linearity), s_4 (slew rate) and s_5 (d.c. offset). The filter is associated with signal s_3 (cut off frequency). Figure 5 presents the schematic of the circuit overlayed with the appropriate signals. For example, the first stage will have a d.c. offset if $(R_1 \parallel R_2) \neq R_3$. Hence, R_1, R_2 and R_3 affect signal s_5. Similarly, R_4 and C_1 affect s_3, and the gain of the first stage of the amplifier is affected by (R_2/R_1) and the open loop gain of the op-amp. The system has test points P_1, TP_0, TP_1, TP_2 and J_1. The tests associated with TP_1 could be as follows :– apply a known input (1V r.m.s. 1 KHz sine wave) at P_1 and (a) measure d.c. voltage (check s_5), (b) measure the ratio of a.c. voltages at TP_0 and known input P_1 (check s_1) and (c) measure the harmonic distortion (check s_2). The slew-rate can be tested by applying high-frequency, high-amplitude sine-waves and observe the slope of the sine-wave at zero-crossings (check s_4). The same set of measurements could be performed at TP_2 and J_1. For the sake of simplicity, let us assume that we only check for s_3 at TP_2, i.e., measure the ratio of voltages at TP_2 and TP_1 for different frequencies. The dependency matrix for this example is presented in Table 4.

A key advantage of the multisignal modeling approach is that the model is independent of the tests associated with the test points. For example, the model does not have to be changed, even if none of the tests check slew-rate s_4. Even more remarkably, if a new design specification is added, say the gain-bandwidth product of the amplifier, it is sufficient to attach a new signal to the components affecting it (i.e., op-amps) and the test point monitoring it. This is in sharp contrast to the approach in [7], where the tests and the corresponding component dependencies are identified first and then a (single-signal) dependency model is derived [7, 8]. Modeling a system based on predefined tests, and then using it to improve testability is akin to the classic *chicken and egg* problem!

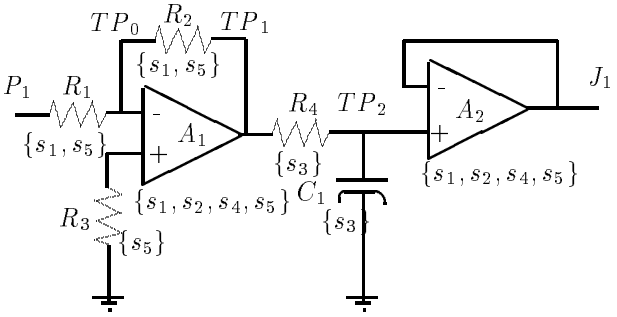

Figure 5: Multisignal Dependency model of an Amplifier/filter

	TP_1				TP_2	J_1			
	s_1	s_2	s_4	s_5	s_3	s_1	s_2	s_4	s_5
R_1 (G)	1	1	1	1	1	1	1	1	1
R_1 (F)	1	0	0	1	0	1	0	0	1
R_2 (G)	1	1	1	1	1	1	1	1	1
R_2 (F)	1	0	0	1	0	1	0	0	1
R_3 (G)	1	1	1	1	1	1	1	1	1
R_3 (F)	0	0	0	1	0	0	0	0	1
A_1 (G)	1	1	1	1	1	1	1	1	1
A_1 (F)	1	1	1	1	0	1	1	1	1
R_4 (G)	0	0	0	0	1	1	1	1	1
R_4 (F)	0	0	0	0	1	0	0	0	0
C_1 (G)	0	0	0	0	1	1	1	1	1
C_1 (F)	0	0	0	0	1	0	0	0	0
A_2 (G)	0	0	0	0	0	1	1	1	1
A_2 (F)	0	0	0	0	0	1	1	1	1

Table 4: The Dependency Matrix of the Amplifier/Filter example

2.6 Hierarchical Multisignal Modeling in TEAMS

A hierarchical multisignal dependency (similar to digraph, or information flow) graph captures the first-order cause-effect relationships between modules and test points. Modules may have (sub)modules (i.e., an embedded dependency subgraph) or be a submodule of a larger system. The lowest level (sub)modules are also called components. Thus, a system can have multiple levels of hierarchy. The modules are the failure sources, or the causes. A test node denotes a monitoring point, where the effects are observable. A link between two nodes A and B denotes that A *affects* B, or B *depends* on A. Higher-order dependencies can be inferred from these first-order dependencies. The redundancy for fault-tolerance, which may hide the failure of a component, is modeled as an AND node. An AND node with M-out-of-N redundancy indicates that *at least* M of its inputs must fail for the output of the AND node to fail. Switches model the various modes of system operation. In addition, they can be used to functionally isolate modules or break feedback loops in test mode to improve the testability of a system.

2.7 Node descriptions in the digraph model

The input requirements of the various nodes of the digraph are as follows:

1. *Module node*: Each module node i is characterized by: module name, number of inputs (I_i), number of outputs (O_i), the set of signals it affects ($SC(c_i)$), Mean-time-to-failure ($MTTF_i$) or failure rate λ_i (measured in number of failures per 1,1000, 1 million or 1 billion hours), repair and rectification times and costs. Modules with built-in-self-test (BIST) are represented by a special node to denote that BIST detects a fault within the module only. The repair times/costs are in any consistent units (e.g., costs in dollars, time in hours) selected by the user.

2. *Test node*: Each test point p has the following properties: test name, the set of tests associated with it, the set of signals detected by each test ($ST(t_j)$), test costs, test times, precedence constraints, resource requirements, setup operations required, probabilities of detection and false alarm, and whether the test is enabled or disabled. The test times/costs are in any consistent units (e.g., costs in dollars, time in hours) selected by the user.

3. *AND node*: **TEAMS** offers a continuously variable voting logic (= $VL\%$) for the AND nodes. This allows the modeling of any M-out-of-N logic, where $M = [VL * N/100]^+$ and $[x]^+$ denotes the smallest integer greater than or equal to x. The reconfiguration reaction time can be instantaneous (= zero delay) or finite.

4. *Switch node*: A switch node is characterized by the orientation of the switch. It enables the modeling of system modes and the breaking of feedback loops in testability mode.

5. *Links*: Links can be marked breakable or unbreakable for loop breaking recommendations from feedback loop analysis.

2.8 The Dependency Matrix

The directed graph model captures the first order cause-effect dependencies such as A affects B and B affects C. The global dependencies, such as A affects C, are inferred by the *reachability analysis* algorithms. Specifically, we need to ascertain which of the failure sources can be observed from each of the tests of the test points, thus enabling us to compute the dependency or fault dictionary matrix, similar to the ones presented in Tables 2–4.

A key requirement in multi-signal modeling is that the component be the smallest functionally distinct entity. This requirement relates the level of details that should be specified in a model to the diagnostic resolution that can be achieved by it. Components can still be of diverse complexity. In example 3, resistor R_2 is a component[3] whereas, in the bus system example, each device and the controller in the bus system is modeled as a component, even though it may consist of hundreds of transistors. The bus system example is a conceptual or macro view of the system where only (failures in) the controller-device communication were modeled. Consequently, the diagnostic resolution is coarse.[4] To sum up, if a component fails, it affects *all* the signals associated with it. Conversely, if any of the signals affected by the component is good, the component is fault-free[5].

It is therefore sufficient to decompose each component into only two aspects, the general failure and the functional failure. Similarly, a test point may have multiple tests associated with it. In this case, if there are L components, L_f of which have functional failures, and n tests, the dependency matrix (D-matrix) is of size $(L+L_f) \times n$, where $d_{i_{GJ}} = 1$, if there is a path from component c_i to test t_j (dependency for general failure in component i) and $d_{i_{fj}} = 1$, if $d_{i_{GJ}} = 1$ *and* $SC(c_i) \cap ST(t_j) \neq \emptyset$. The D-matrix summarizes the diagnostic information of the system and all analysis is performed using this matrix.

In order to find the reachability of test points from a given module, tokens are propagated from the module via the digraph's links to determine which other nodes are affected. When a token reaches a module, test, or a switch, copies of it are propagated to appropriate outputs of the node and to the links connected to them. The output of an AND node specifying M-out-of-N redundancy is not propagated until at least M tokens are received by it. To prevent the algorithm from entering an infinite loop when a cycle is encountered, tokens are not replicated nor propagated if a node has already been reached by another token. The algorithm terminates when tokens cannot be propagated to any new nodes. The signal information is then used to derive the multisignal dependencies, $d_{i_{fj}}$, for each test point monitoring multiple signals and each component with functional failure. This algorithm takes $O(E)$ operations (E = number of links in the graph) to find the reachability of test points from a given module. The procedure is repeated for all modules; thus, the worst-case complexity of our algorithm is $O(EL)$, where L is the number of components.

[3] If R_2 fails functionally, both s_1 and s_5 must be affected, since both are functions of R_2.

[4] One could also break up each of the devices into (sub)components – bus-driver and disk-subsystem, and achieve higher diagnostic resolution - such as communication error and media error.

[5] If the multi-fault algorithm, presented in section 4 is used, this requirement can be relaxed by assuming each signal can fail individually. This allows for more flexibility in modeling partially defined systems, where the component information is not yet available.

3 Static Fault Analysis

Static fault analysis techniques are used as a rapid means to assess the inherent testability of a system. It identifies undetectable faults, ambiguity groups, and redundant tests. An ambiguity set of faults corresponds to a set of identical rows in the D-matrix. Since they all have the same failure signature, the failure sources cannot be isolated within an ambiguity group. Similarly, redundant test sets identify tests that have identical diagnostic information, i.e, sets of identical columns in the D-matrix. In addition, Feedback loop analysis identifies the topological testability limitations of the system and makes DFT recommendations to overcome them, while hidden and masking false failure analysis sets the stage to troubleshoot systems with possibly multiple failures.

3.1 Feedback loop analysis

A system is said to have feedback loops in the sense of diagnosability, whenever there is a circular flow of diagnostic information feeding back onto itself. When we analyze dependency (information flow) models, topological circularities most often correlate with physical feedback loops. Unless these cycles are broken (by placing tri-state buffers that block the feedback of diagnostic information), additional tests will not improve the testability of a system.

Our approach to identify the feedback loops is to decompose the graph model of a system into its *strongly connected components*. A strongly connected component (SCC) is defined as the set of nodes in the directed graph in which there is a path from every node to every other node. Hence, the only nodes that appear in a *cycle* are those which are part of a strongly connected component. We identify the strongly connected components of a system graph using an algorithm due to Tarjan [9] which takes $O(E)$ operations, where E is the number of arcs in the system graph. In the parlance of testability analysis, the strongly connected components of the system graph are termed "gross feedback loops". Every gross feedback loop in the directed system graph represents an ambiguity group of components which cannot be diagnostically resolved by inserting any number of test points. Hence, it is necessary to place tri-state buffers within a strongly connected component to prevent the information feedback. This problem is equivalent to determining the minimal set of links which need to be removed in order to break the strong connectivity.

We adopt the following heuristic approach that works directly on the SCC without having to enumerate all the elementary cycles in it. Consider a link (v, w) belonging to an SCC, where v is the start-node of the link and w is the end-node. We define a Figure Of Merit (FOM) for this link as: FOM = indegree(v) $\times$ outdegree(w). Thus, FOM(v, w) represents the minimum number of cycles in which the link (v, w) appears. Hence, FOM(v, w) is indicative of the number of cycles that will be broken when link (v, w) is removed. Given a strongly connected component, we break the link that maximizes the above Figure of Merit and recompute the SCCs of the resulting subgraph. This process is repeated until the strong connectivity is completely broken. The broken links mark the potential locations of the tri-state buffers.

3.2 Hidden and masking false failures analysis

Most traditional testability analysis algorithms make the simplifying assumption that there is at most one fault in the system. However, this assumption may not hold for large systems involved in a prolonged mission with no opportunity for repair during the mission. In such cases, diagnosis based on single-fault assumption can produce wrong inferences under certain conditions. For example, we need to analyze a system for potential hidden failures, i.e., the set of failures that get masked by another failure. The diagnostic procedure must check for these additional failures when the single fault assumption is not valid. Another important problem in analyzing multiple failures is the potential for *masking false failures*. A *masking false failure* occurs when the symptoms of two or more failures add up to mimic the failure of an unrelated element. The diagnostic procedure based on single fault assumption will replace the implicated failure source and obviously fail to repair the system. Hence, identifying the hidden faults and masking false failures would help the maintenance technician in adapting the single fault diagnostic procedures to multiple failure situations.

The problem of identifying the sets of hidden failures is relatively easy to solve. The binary fault-test matrix, $D = [d_{lj}]$, consists of failure aspects $A = (a_0, a_1, \ldots, a_m)$ as row indices and tests $T = (t_1, t_2, \ldots, t_n)$ as column indices[6]. Thus, the element $d_{lj} = 1$ denotes that the failure aspect a_l is detected by test t_j. Therefore, each row in the D-matrix corresponds to a failure aspect and its observability with respect to the available tests. The set of hidden failures H_l for the failure aspect a_l is given by, $H_l = \{j | 1 \leq j \leq m, j \neq l, D_l \cup D_j = D_l\}$, where D_l denotes the l-th row of the D-matrix, m is the total number of failure aspects, and $\cup$ denotes the logical OR operation.

In contrast, the problem of enumerating the masking false failure sets for a given fault is computationally ex-

[6]Here, a_0 is a *dummy* failure aspect that represents the fault-free condition

pensive. Typically, it requires $O(2^m n)$ operations, which is impractical for even moderate values of m. Hence, in the following, we develop a new approach for enumerating only the "irreducible subsets". We define an "irreducible set" as a set of rows which when logically OR-ed would produce the row corresponding to the fault under consideration and excluding any one of the rows in this set would produce a different row pattern. This subset is irreducible in the sense that each member row of the set is indispensable for the set to qualify as a masking false failure set. Enumerating these is enough, since every other masking false failure set is a superset of the irreducible subsets.

This problem is related to one of determining the minimal hitting sets discussed in [10, 11]. Let $\hat{n}_k$ be the number of 1's in the reference row D_k, and H_k be the set of hidden faults for the reference row D_k. Let $I_k = \{j : d_{kj} = 1\}$ represent the set of columns in the reference row that contain 1's. Define a function $\hat{m} : \{1, \ldots, \hat{n}_k\} \rightarrow I_k$ that maps the set of $\hat{n}_k$ contiguous integers onto the set I_k. Further, define the sets $L_j = \{l : l \in H_k, d_{l\hat{m}(j)} = 1\}(1 \leq j \leq \hat{n}_k)$, where L_j represents the subset of hidden rows for the reference row that have a 1 in the $\hat{m}(j)$-th column position. Given the sets $L_j(1 \leq j \leq \hat{n}_k)$, it is clear that a set R is a masking set if $R \subseteq \bigcup_{j=1}^{\hat{n}_k} L_j$, such that $R \cap L_j \neq \phi(1 \leq j \leq \hat{n}_k)$, where ϕ is a null-set. If R is an irreducible masking set, then no proper subset of it satisfies the above criterion. Thus, once we determine the sets $\{L_j\}(1 \leq j \leq \hat{n}_k)$ for a given row k, we can use Reiter's algorithm [10, 11] to find all the irreducible masking subsets for the failure aspect a_k.

4 Test Sequencing Algorithms

The test sequencing problem, in its simplest form, consists of:

1. a set of failure aspects $A = (a_0, a_1, \ldots, a_m)$ associated with the system, where $a_l(1 \leq l \leq m)$ denotes one of the m potential failure aspects in the system, and a_0 is a dummy failure aspect denoting fault-free condition;

2. the conditional probability vector of the failure aspects $P = [p(a_0)p(a_1)\ldots p(a_m)]^T$; [7]

3. a set of n available tests $T = (t_1, t_2, \ldots, t_n)$ with an application cost vector $B = [b_1, b_2, \ldots, b_n]^T$ where b_j denotes the usage cost of test t_j measured in terms of time, manpower requirements, or other economic factors,

4. a vector of repair costs of the failure aspects $F = [f_1, f_2, \ldots, f_m]$,

5. a diagnostic dictionary matrix $D = [d_{lj}]$, where d_{lj} is 1 if test t_j detects a failure aspects a_l and 0 otherwise.

The problem is to devise a sequential testing strategy (in the form of a binary decision tree) such that the expected testing cost (i.e., diagnostic cost) defined by

$$J = \sum_{l=0}^{m} \left\{ \sum_{j=1}^{|\mathcal{P}_l|} b_{\mathcal{P}_l[j]} + f_l \right\} p(a_l) \qquad (1)$$

where $\mathcal{P}_l$ denotes the ordered set of indices representing the sequence of tests applied to isolate the failure aspect a_l (the optimization is over $\mathcal{P}$, the class of all such admissible ordered sets), $\mathcal{P}_l[j]$ is the j-th element of the sequence $\mathcal{P}_l$, and $|\mathcal{P}_l|$ is the cardinality of the sequence $\mathcal{P}_l$.

This problem belongs to the class of binary identification problems that arise in medical diagnosis, nuclear power plant control, pattern recognition, and computerized banking [12]. The optimal algorithms for this problem are based on dynamic programming (DP) [13] and AND/OR graph search procedures. The DP technique is a recursive algorithm that constructs the optimal decision tree from the leaves up by identifying successively larger subtrees until the optimal tree rooted at the initial node of complete ambiguity is generated. The DP technique has storage and computational requirements of $O(3^n)$ for the basic test sequencing problem.

4.1 Top-down search algorithms

The AND/OR graph search algorithms presented in [12] are top-down algorithms that replace the optimal cost-to-go by an easily computable estimate of the optimal cost-to-go. A novel feature of this approach is that the estimate (termed as the Heuristic Evaluation Function (HEF)) is derived from Huffman coding and entropy. These information theoretic lower bounds ensure that an optimal solution is found using the AO*, HS, and CF search algorithms. In addition, the top-down nature of the AND/OR graph search algorithms have enabled us to derive several near-optimal search algorithms that provide a trade-off between optimality and computational complexity. In the following, we present brief descriptions of each of these algorithms.

[7] These are conditional probabilities computed from *a priori* probabilities of failure aspects $[p_1, p_2, \ldots, p_m]^T$ based on single fault assumption [14].

4.1.1 AO^*_ϵ algorithm

The algorithm AO^*_ϵ is similar to AO^* except that the search space is reduced substantially by incorporating a higher threshold than necessary for selecting a new path in the AND/OR graph for expansion. A nice feature of this algorithm is that, it is an ϵ-optimal algorithm, i.e., guarantees that the solution found does not exceed the optimal cost by a factor $1+\epsilon$ ($\epsilon > 0$).

4.1.2 Limited search AO^*

AO^* retains all admissible tests at every OR node of ambiguity for further examination until the OR-node is solved, i.e., an optimal solution tree is found starting from that ambiguity set. In order to overcome the computational explosion of AO^* while solving large problems, limited (breadth) search AO^* retains only $MBEST$ best tests at each OR-node, where $MBEST$ is a user specified parameter. The $MBEST$ best tests are selected by ranking the tests on the basis of their information gain per unit cost [12].

4.1.3 Multi-step information heuristics

AO^* and its near-optimal variants are essentially breadth-first strategies. An alternative is to select the next best test based on a limited lookahead by a depth-first expansion of the decision tree from an OR node of ambiguity. A simple MSTEP lookahead greedy heuristic algorithm for the selection of a test at a given ambiguity node x can be devised as follows. For each test t_j from the set of available tests at the reference ambiguity set, the left/right children are computed for the pass/fail outcomes. For each child, the best possible test is selected based on the single step information heuristic and the OR node is split further into its left/right children corresponding to the pass/fail outcomes. The above computation is repeated for each child recursively until the depth of the search tree reaches MSTEP. The information gain per unit cost of the search tree is then computed, and the test that maximizes this figure of merit is selected for splitting the reference ambiguity set. Fig. 6 shows the computational performance of various test sequencing algorithms for realistic problems of varying sizes.

4.1.4 Minimax optimization

In all of the above algorithms, the cost criterion to be minimized is the expected testing cost. Minimization of expected cost can sometimes result in inordinately expensive sequences of tests to isolate faults of very low probability of occurrence. This may not be acceptable since the estimates of the MTTFs of the components are often inaccurate. In these cases, the dependence of the cost function on the underlying probability distribution can result in diagnostic strategies that are not really optimal. Minimax (minimizing the maximum testing cost) is a criterion that results in robust diagnostic strategies. We can extend the AO^* algorithm to perform minimax optimization by modifying the HEF $h(x)$ used in [12]. The HEF for the minimax criterion does not depend on the probability distribution of the failure sources over the ambiguity group x. Our experiments with this criterion have resulted in robust diagnostic trees that are well-balanced and near-optimal, even in the sense of expected cost.

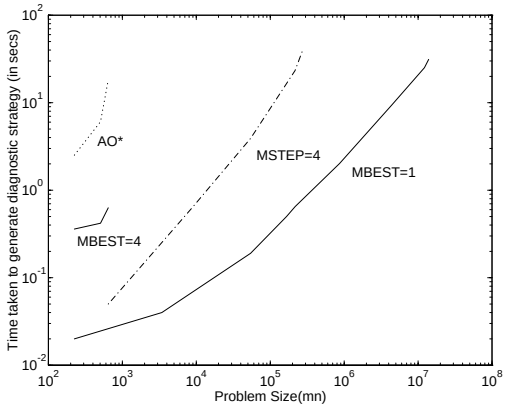

Figure 6: Computational complexity of various algorithms on a Sun Sparc10 (m=number of failure sources, n=number of tests)

4.2 Extensions to generalized testing with practical features

4.2.1 Multiple fault isolation

As discussed in Section 3.2, in many of the real world situations, the single fault assumption does not hold. In section 2.8, we also noted that it is necessary to have a multi-fault algorithm to analyze (incomplete) models that are not defined down to component level, and, hence, one or more signals associated with the lowest level module may fail individually. Even though static analysis provides a means to deal with multiple faults, there is a need for developing dynamic diagnostic strategies to isolate multiple faults that may be present in a system. Earlier research [14] produced a set of multi-fault algorithms based on enumeration of possible multi-failure combinations using the compact set notation. However, they are not suitable for problems with more than 600 aspects [14]. In the following, we outline a multi-fault test sequencing

algorithm suitable for the solution of larger problems.

Let $TS_j = \{a_i | d_{ij} = 1\}$ be the set of faults covered by test t_j, TA be the set of available tests, P be the set of known good components and $S = \bar{P}$ be the set of *suspected* faulty components. Further, let $X = TS_j \cap S$, be the set of suspected failure aspects covered by test t_j. The probability that test t_j passes is the probability that none of the failure aspects in X is faulty. This probability is given by $\Pr(t_j = \text{pass}) = 1 - \Pr(X) = \prod_{i \in X}(1 - p_i)$, where p_i is the unconditional apriori probability of aspect a_i. If test t_j passes, we update the set of known good aspects as $P = P \cup X$. Moreover, if $|X| = 1$, and test t_j fails, aspect X is definitely faulty. This is known as the *one-for-sure* condition [14]. Otherwise, if $|X| > 1$ and the test fails, no update on P and S is possible. Therefore, a measure of information content of test t_j is[8], $IC_j = (u(|X| - 1)\Pr(X) - 1)(\hat{p}_1 \log \hat{p}_1 + \hat{p}_2 \log \hat{p}_2)$, where, $\hat{p}_1 = (\Pr(X)/\Pr(S))$, $\hat{p}_2 = 1 - \hat{p}_1$, and $u(x)$ is a unit step function.

The algorithm then proceeds as follows. Initially, $P = \{a_0\}$ and $S = \{a_1, a_2, \ldots, a_m\}$. The test, t_b, with the highest information content is applied. If it passes, P and S are updated as follows : $P = P \cup \{TS_b \cap S\}$, $S = \bar{P}$. The next best test (selected from the remaining set of tests) is then applied. After every update of P, the *one-for-sure* condition is checked. If for any test t_j that failed previously, $|TS_j \cap S| = 1$, then aspect $TS_j \cap S$ is replaced and added to set P, and any previously failed tests with non-zero information content, that covered the repaired aspect, is added to the list of available tests. The process is continued until $\bar{P} = \emptyset$, or none of the available tests have non-zero information gain. In the latter case, if $\bar{P} \neq \emptyset$, $\bar{P}$ is the ambiguity group, and the diagnostic process terminates.

4.2.2 Modular diagnosis

In order to improve the availability of a system, it is often sufficient to isolate the failure source to a module. An optimized test sequence which attempts to isolate modules instead of individual failure aspects is important for field maintenance. The broad outline of the strategy for modular diagnosis is the same as that for diagnosis to aspect level. The structure of the decision tree is the same, but each of the OR-nodes in the decision tree represents an ambiguity set of modules. Also, the conditional probability distribution of the modules in the ambiguity node x is state-dependent. With these modifications, the test sequencing algorithms described in the previous sections can be easily extended to modular diagnosis.

[8] Other measures of the information content are currently under investigation

4.2.3 Precedence constraints and setup operations

The basic test sequencing algorithms can be extended to include practical features such as precedence constraints and setup operations for tests. In many practical systems, some tests (e.g., power on, safety tests) must precede others. Also, in many systems, tests have setup operations, some of which may be common among multiple tests. Formally, we denote precedence constraints via a mapping $\gamma : \{t_j\} \rightarrow \{T_j^p\}, (1 \leq j \leq n)$ and $T_j^p \subset T$. That is, for each test t_j, T_j^p which is a subset of T not containing t_j is the set of precedence constraints. Denoting the set of setup operations as $\hat{T} = \{\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_{\hat{k}}\}$, the dependence of the tests on the setup operations is described by the mapping $\psi : \{t_j\} \rightarrow \{\hat{T}_j\}, (1 \leq j \leq n)$ and $\hat{T}_j \subset \hat{T}$. That is, for each test t_j, $\hat{T}_j$, which is a subset of $\hat{T}$, is the set of setup operations. Further let $\hat{b}_k$ denote the cost of k-th setup operation.

The problem is to devise a sequential strategy such that the expected diagnostic cost

$$J = \sum_{l=0}^{m} \left\{ f_l + \sum_{j=1}^{|\mathcal{P}_l|} b_{\mathcal{P}_l[j]} + \sum_{k \in \bigcup_{j=1}^{|\mathcal{P}_l|} \hat{T}_{\mathcal{P}_l[j]}} \hat{b}_k \right\} p(a_l) \quad (2)$$

is minimized, with the precedence constraints

$$T^p_{\mathcal{P}_l[j]} \subset \{\mathcal{P}_l[1], \mathcal{P}_l[2], \ldots, \mathcal{P}_l[j-1]\}, 1 \leq j \leq |\mathcal{P}_l|. \quad (3)$$

AO* and its variants need to be modified in the following ways in order to incorporate the precedence constraints and setup operations.

1. Since the test costs are state-dependent due to the commonalities of setup operations, the estimate of the cost-to-go at a given ambiguity node x in the tree is a function of the tests used so far.

2. The set of admissible tests at a given node should not include those tests (even though they split the ambiguity set) whose precedence constraints are not satisfied, i.e., the set of admissible tests is also state-dependent.

3. If the only information yielding tests at a given node are those whose precedence restrictions are not satisfied, then they should be considered as admissible with their cost incremented by the costs of the unapplied precedence tests.

4.2.4 Rectification

The algorithms considered so far attempt to isolate the fault to the lowest level component/modules with the

available tests and then effectuate its repair. If availability of the system is the prime consideration, rectification of failure sources can be a superior alternative. Rectification, as opposed to repair, is the replacement of a potentially faulty component/module without prior diagnosis. Rectification often leads to quicker diagnosis of the most likely failure sources at the expense of increased cost due to replacement of components/modules. Rectification can be modeled as additional "pseudo" tests which detect faults in a module with cost equal to the rectification cost of the component/module. That is, rectification amounts to having an additional column in the test matrix which has all zeros except for the rows corresponding to the rectified faults.

5 Conclusion

In this paper, we present a novel, intuitive and flexible modeling methodology and provided a brief overview of the **TEAMS** software package that uses it for automatic test sequencing and testability analysis of complex hierarchically-described modular systems. The multisignal modeling methodology enables the modeler to add functional failure information on a structural model via signals. Thus, high diagnostic resolution can be achieved without the complexity of qualitative and quantitative modeling or the structural distortion of dependency modeling. This,in turn,simplifies the task of model validation. We expect that the concomitant reduction in modeling and model validation cost makes DFT cost effective for commercial applications, as well. Multisignals also enable us to model advanced testability features, such as multiple tests on a test point, test suites and component tests, without deviating from structure. In addition, the ability of **TEAMS** to import equivalent diagnostic models of modules from other sources (VHDL structural models, EDIF, Failure Environment Analysis Technique (FEAT), etc.), allows for the seamless integration of models of subsystems. The test sequencing algorithms of **TEAMS** can handle real-world features such as rectification, modular diagnosis, precedence restrictions, setup operations for tests, imperfect tests, and multiple fault isolation.

References

[1] Pattipati, K., Raghavan, V., Shakeri, M., Deb, S., and Shrestha, R., "TEAMS: Testability Engineering and Maintenance System" Invited paper, 1994 IEEE ACC, Baltimore, Maryland, pp. 1989–1996.

[2] Isermann, R., "Process Fault Detection Based on Modeling and Estimation Methods - A Survey, "*Automatica*, 1984, pp. 387-404.

[3] Kuipers, B., " Qualitative simulation: Then and Now," *Artificial Intelligence*, Vol. 59, 1993, pp.133-140.

[4] Kuipers, B., " Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge, "*International Federation of Automatic Control*, 1989, pp.571-585.

[5] Proposed IEEE - AIESTATE specification P1232, Draft 1.2, October 1992.

[6] Chreim, Samir., *Aggregate Models of Digital Systems for System Level Testability Analysis,* M.S. Thesis, Dept. of Electrical and Systems Engineering, Univ. of Connecticut, Storrs, CT 06269-3157.

[7] Dill, H.H., "Test Program Sets – A new approach," *1990 IEEE Autotestcon*, San Antonio, Texas.

[8] "Weapon System Testability Analyzer (WSTA) – Introductory level training", *IDSS software support activity, Test Systems Engineering and Technology Branch*, NUWC, Newport, RI, June 1993.

[9] Tarjan, R., "Depth-First Search and Linear Graph Algorithms," *SIAM J. Comput.*, Vol. 1, No. 2, June 1972.

[10] Reiter, R., "A Theory of Diagnosis from First Principles," *Artificial Intelligence*, Vol. 32, 1987, pp.57-95.

[11] Greiner, R., Smith, B. A., and Wilkerson, R. W., "A Correction to the Algorithm in Reiter's Theory of Diagnosis," *Artificial Intelligence*, Vol. 41, 1989/90, pp.79-88.

[12] Pattipati, K.R., Alexandridis, M.G., "Application of Heuristic Search and Information Theory to Sequential Fault Diagnosis," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. 20, No. 4, July 1990, pp.872-887.

[13] Bertsekas, D.P., *Dynamic Programming: Deterministic and Stochastic Models,* Prentice-Hall, Englewood Cliffs, NJ, 1987.

[14] Shakeri, M., Pattipati, K., Raghavan, V., and Deb. S.,"Near Optimal Sequential Testing Algorithms for Multiple Fault Isolation.", to be presented at *IEEE SMC conference*, San Antonio, Texas, Oct. 1994.