PARTITIONING LARGE DIAGNOSTIC PROBLEMS

William R. Simpson and John W. Sheppard
ARINC Research Corporation
2551 Riva Road
Annapolis, Maryland 21401

ABSTRACT

An increasing number of full system testability analyses are being handled by information and dependency modeling. Although top-level design review analyses have not presented problems, there is concern that detailed models of large systems may exceed computer and computational capabilities of the most sophisticated modeling approach. It is reasonable to assume that the communications portion of Space Station Freedom or the avionics on the Advanced Tactical Fighter (ATF) may exceed from 10,000 to 15,000 elements in diagnostic models.

Information and dependency modeling approaches are based on a set partitioning algorithm that has $O(n^3)$ complexity, thus requiring a factor of 8 or more computational power for each doubling of the model size. In large, complex systems such as those mentioned above, we may not be able to perform static analysis, much less interactive diagnosis, without breaking the problem into smaller pieces. The difficulty is dividing the system while still maintaining the desired subsystem interactions.

In this paper we describe a basic approach to modeling large diagnostic problems, such as Space Station Freedom or the ATF avionics. For this approach we present three techniques for developing submodels to create an interacting network of submodels of reasonable size. Specifically, we discuss:

- Partitioning by articulation points
- Partitioning by logical cut points
- Partitioning at arbitrary points

These techniques have been developed as part of a total integrated diagnostics package that includes testability analysis; design for testability; and a full range of diagnostic capabilities, including built-in test, embedded diagnostics, automatic test, and portable maintenance aids. We address partitioning models produced using the modeling approach as implemented in the System Testability and Maintenance Program (STAMP®) and the Portable Interactive Troubleshooter (POINTER™). These techniques, however, generally apply to any of the current formulations for complex systems diagnoses.

INTRODUCTION

The 1980s saw increasing emphasis on designing systems for field operations. Although manifested in many contexts, such as design for testability, concurrent engineering, and integrated diagnostics, emphasis is measurably shifting in the approach to the maintenance of complex systems from designing for performance to designing for performance and maintainability. Customers are demanding to know what the field maintainability of systems will be, almost before they ask about system performance. It is clear that the old method of doing business is giving way to a more structured approach. We can no longer afford the luxury of designing to performance specifications and then supplying maintenance and diagnostic procedures after other objectives have been achieved. We have seen that the result of this nonintegrated approach to design has led to retest OK (RTOK) rates in excess of 40% with field no fault found (NFF) rates at 50% and false alarm rates (FARs) often exceeding valid detections [1,2]. Detailed system specification attempted to limit the downstream liability resulting from poor maintainability, but field results were often years away from the ad hoc measures taken by designers. As a result, the Department of Defense (DoD) instituted a standard to require detailed analysis of testability issues during design [3].

Many companies developed tools to help handle the analysis task. ARINC developed two tools to assist in developing testable systems and in streamlining the maintenance process—STAMP and POINTER. STAMP is a model-based reasoning system that is used to conduct testability analysis and develop fault-isolation strategies to improve system maintenance. POINTER, which was derived from STAMP, uses the system model generated by STAMP for its knowledge base. With the system model, POINTER interactively presents test material and guides the maintenance process through diagnosis and repair. POINTER may be applied to manual, semiautomatic, and automatic fault-diagnosis problems; advanced capabilities include learning and reasoning under uncertainty [4,5]. Many other analysis systems, based on the dependency model approach, have been developed as well [6-8]. These types of tools have been in use now for about 10 years and, in the case of STAMP and POINTER, have garnered significant and even spectacular results compared with past approaches [9,10].

CH2941-3/91/0000-0329 $1.00 ©1991 IEEE

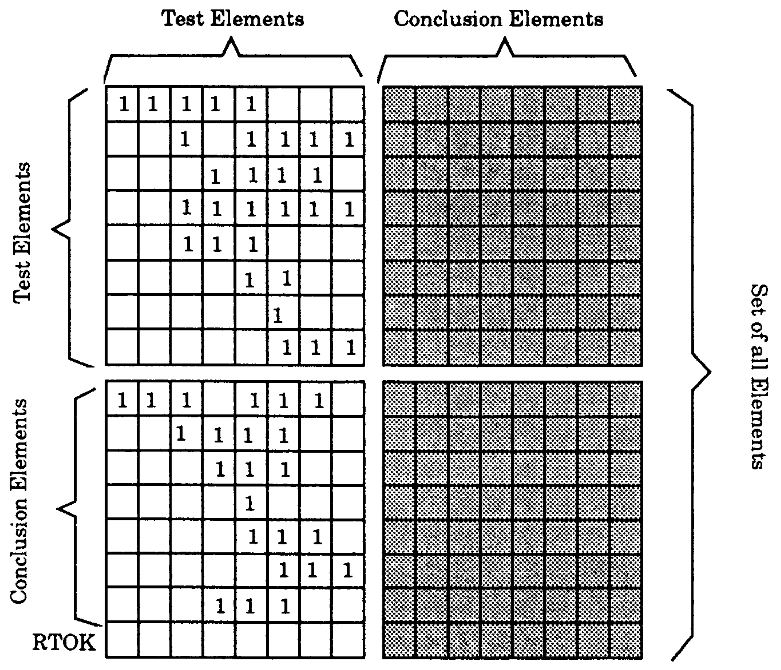
BASIC PROBLEM FORMULATION

With STAMP, the basic formulation represents the topology of the system being analyzed in a graph representation, where the tests and components are the vertices of the graph and the dependency relationships are the edges of the graph. Specifically, the representation is a bit matrix that is an adjacency matrix for test dependencies [11].

Figure 1 shows the basic matrix formulation. The bit matrix representation is compact and requires only $\theta(n^2)$ bits for storage, where n is the number of elements in the matrix, corresponding to the sum of the testable elements and the conclusion elements plus 1 (for the RTOK conclusion). Testable inputs comprise two elements under this formulation: one for the test element represented, and one for the conclusion element represented. The shaded portion of the matrix is not used because we do not consider component-to-test dependencies or component-to-component dependencies.

across this matrix and requires $O(n^3)$ time. The result of using this approach is a good approximation of an optimization problem that is known to be NP-complete [12]. On the other hand, inferences are drawn in bit-parallel fashion and are significantly faster (i.e., $O(n)$) than the logical chaining required by an expert rule representation of the system. Thus, portable maintenance aids for moderately complex systems can provide a full range of options and still respond to user input because of today's faster microprocessor chips. In fact, because models can now be considered that will require from 10,000 to 15,000 elements, it is the compactness of this algorithm that led us to investigate the application to diagnosis of a full system at a level of detail that would be difficult for the normal rule-based expert system.

Although the representation of the system is extremely compact and efficient, the very size of the model may become a problem from a computation standpoint, storage standpoint, or both. Conceptually, the problem can be partitioned into smaller pieces while still retaining the



Note: A value of 1 in a column indicates a dependency relationship.
RTOK is always empty.

Figure 1. Basic System Representation in STAMP and POINTER

Although STAMP has evolved to include a number of logical constructs that cannot be represented in this format, the basic graph representation is still an integral part of the calculational burden. In particular, the test-choice algorithm (which we have simplified for an interactive calculation) involves the computation of information entropy values desired level of interaction at the full system level. Ideally, this system-level partitioning is accomplished with one or more readily observable indications that then point to the appropriate submodel. If the submodels are partitioned at points of no overlap or at least minimum overlap, computation efficiency is maintained. We call a test that partitions the model into two nonoverlapping submodels a "clean" cut point. If the submodels are significantly smaller than the original model, a substantial reduction in storage requirements results. Figure 2 shows conceptually the limits of partitioning a model into two submodels. The partitioning into submodels can be either well-conditioned (as shown in Figure 2.a) or ill-conditioned (as shown in Figure 2.b), the major difference being in the benefit derived by partitioning. Ill-conditioned partitioning may be described as partitioning that has only a small impact on model size requirements. Figure 2.a shows a substantial reduction in the model size regardless of the outcome of the test symptom, and Figure 2.b shows a poor partitioning taking place with the symptom chosen. Given a set of readily observable indications, we may be able to accomplish only ill-conditioned partitioning. But ill-conditioned partitioning points should be avoided where possible.

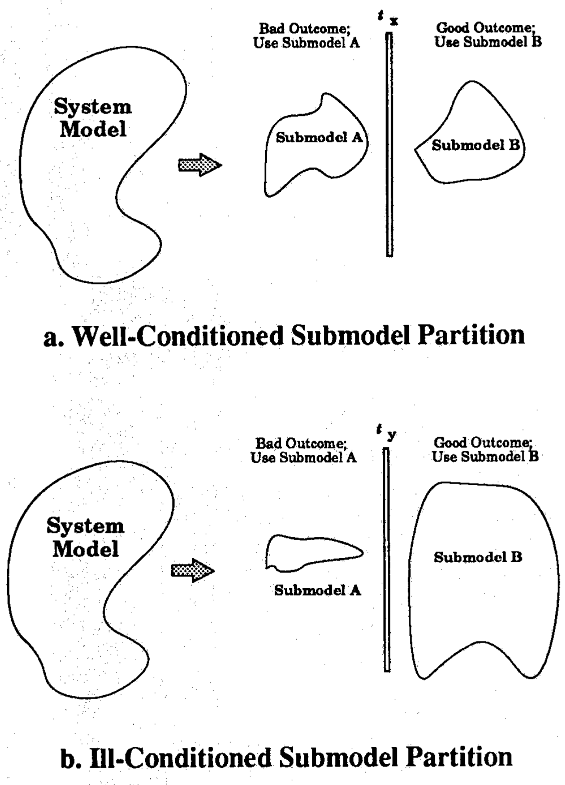

Bad Outcome; Use Submodel A
t_x
Good Outcome; Use Submodel B
System Model
Submodel A
Submodel B
a. Well-Conditioned Submodel Partition
Bad Outcome; Use Submodel A
t_y
Good Outcome; Use Submodel B
System Model
Submodel A
Submodel B
b. Ill-Conditioned Submodel Partition

Figure 2. A Representative Partition of a Larger Model into Two Submodels

We have developed three basic techniques for partitioning the model. The first, articulation-point partitioning, uses the properties of the graph representation. The second, logical cut-point partitioning, takes advantage of the logical properties of the representation. When dealing with real systems, it is recognized that graph cut points and logical cut points may not occur at readily observable indications, so the third technique, arbitrary partitioning, provides a completely arbitrary cut-point process. The latter is subject to a varying degree of submodel overlap.

PARTITIONING BY ARTICULATION POINTS

An articulation point, or cut vertex, is a vertex on a graph that can be used to cleanly separate the graph into two subgraphs. The vertices of the graph are the candidates for the articulation points, which, in our case, are the tests (see Figure 1). To derive the formulation of an articulation point, we must first examine graph *connectivity*. We define our graph, G, to be a set of vertices (tests and conclusions) and edges (dependencies) as illustrated in Figure 1. The vertices, V, are connected by the edges, E. An undirected graph is a graph in which the edges may be traversed in either direction. A directed graph is a graph in which the edges may be traversed in only one direction. A path is defined as the set of edges required to go from some vertex (say, v_i) to another vertex (say, v_j). If we assume the graph is undirected, then the graph is connected if and only if at least one path exists between any pair of vertices. If the graph is directed, then the graph is "strongly" connected if and only if at least one path exists between any pair of vertices. A directed graph may be "weakly" connected if its undirected form is connected but its directed form is not strongly connected. Any strongly connected graph is also weakly connected. One final form of connectivity that must be examined is *biconnectivity*. A graph is biconnected if and only if, for every pair of vertices in the graph, at least two independent paths exist between the vertices:

$$\forall_i \ \forall_j \bigcap_k P_{ij}^k = (v_i, v_j),$$

where P_{ij}^k is the kth path between i and j. If a graph is not biconnected, then some pair of vertices, v_i and v_j, exists such that a third vertex, v_x, is on every path between v_i and v_j. This can be represented by

$$\forall_i \ \forall_j \bigcap_k P_{ij}^k = (v_i, v_x, v_j).$$

The common vertex in the graph v_x is the articulation point.

Figure 3 shows a topological representation of a system that we use here for example purposes. In Figure 3, t is used for the test elements, and c is for conclusions. Two additional elements are given by *int* and *inu*, representing testable and untestable inputs, respectively. The testable inputs are modeled as two vertices (test part and conclusion part). The untestable inputs are modeled as single vertices. The actual process by which articulation points are uncovered considers only the test-to-test subgraph and involves the building of a depth-first spanning tree, which is an approach to systematically examining every vertex of a graph [13]. For the system of Figure 3, the articulation

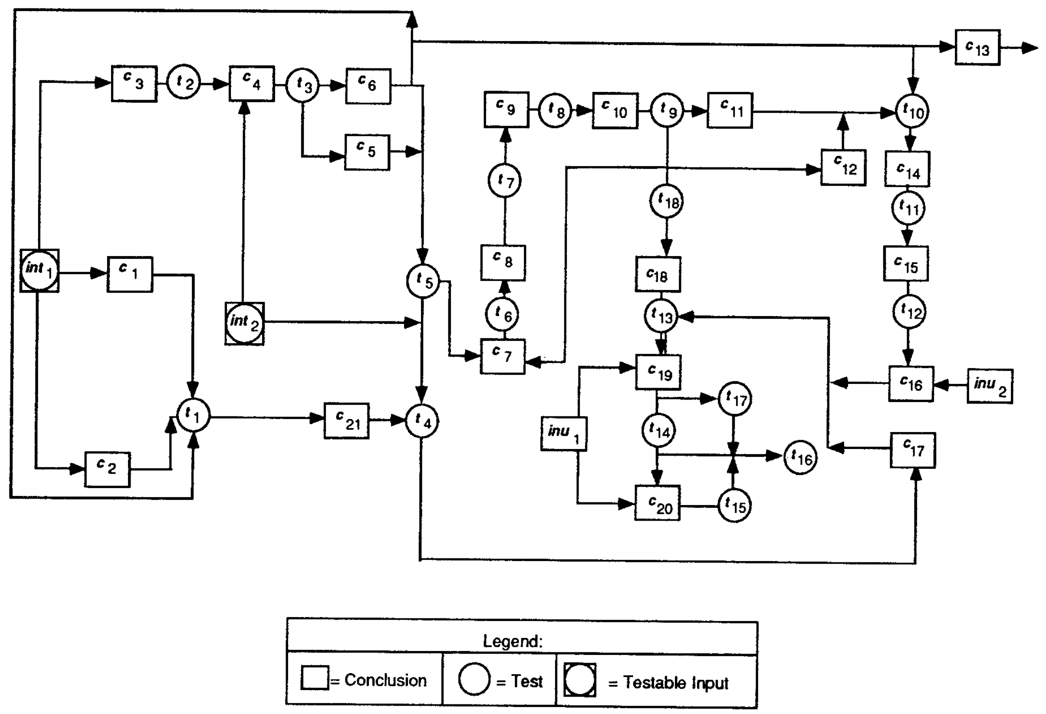


Figure 3. An Example System for Illustration Purposes

points are identified as t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, int_1, and int_2. Given the articulation points, the system model can be partitioned along these points to form small submodels. Once the model is partitioned, two computationally intense processes are simplified.

First, in analyzing a system, we determine all higher order dependencies for a test through a process called *closure* (see the next section, "Partitioning by Logical Cut Points"). This algorithm requires $O(n^3)$ time. The time required for closure can be reduced if we first partition the model, close each submodel, construct a high-level model representing the submodel interaction, and close this high-level model.

Second, as mentioned in "Basic Problem Formulation," the task of selecting a test to evaluate also requires $O(n^3)$ time. To choose a test, we proceed in a top-down fashion in which we can choose from among the articulation points. This process continues until a single submodel is isolated; that submodel forms the basis for further diagnosis. The resulting diagnostic strategy may be less efficient than considering the entire model at once, but computation time and storage space requirements can both be significantly reduced.

PARTITIONING BY LOGICAL CUT POINTS

The articulation-point technique relies heavily on the *graphical properties* of the representation. The logical cut-point technique relies heavily on the *logical properties* of the representation [14]. This technique involves identifying tests that are piece-wise serial. A serial system is one in which there is a test between every component and the system is represented by a single flow path. Figure 4 shows a simple serial system. In the serial system, each test is a clean cut point in that it divides the overall problem into two independent subproblems. The graphical form, when ordered, is an upper triangular matrix. Each test feeds all downstream elements and is fed by all upstream elements. Real systems, of course, seldom exhibit this serial nature, although significant parts of them may (see Figure 3). A test is considered to be piece-wise serial if it exhibits the properties of a test in a serial system. That is, when the system is ordered, the test feeds all downstream elements and depends upon all upstream elements.

One way to identify serial subsystems within a larger system would be to order the system and examine each test to see if the piece-wise serial property exists. This requires the logical chaining of inferences to check for the existence of relationships that occur beyond the nearest neighbor. It is sufficient, however, to examine the higher order relationships to determine whether an upstream or downstream relationship exists between the test being

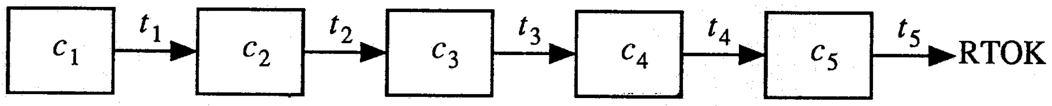


Figure 4. A Serial System for Illustration Purposes

considered and every other test. For graphically represented systems, the higher order relationships may be obtained through a process called transitive closure [15]. Transitive closure replaces the chaining of rules that must be applied to examine the higher order relationships and is based on the transitivity of logical implication:

$$[(P \rightarrow Q) \wedge (Q \rightarrow R)] \rightarrow (P \rightarrow R),$$

where P, Q, and R are logical premises, and $\rightarrow$ indicates implication.

An additional form of closure, logical closure, checks for the presence of higher order relationships using the inference metarules of the inference engine being used [16]. The primary metarules used in STAMP and POINTER are provided in Table 1. Rules 4 and 5 of that table provide the basis for logical closure and may provide additional higher order dependencies due to secondary implications. In particular, a secondary implication results as follows: Given two tests, t_i and t_j, if we examine the set of conclusions upon which t_i depends, A, and the set of conclusions upon which t_j depends, B, and find that set B is a subset of set A, then we conclude that t_i depends upon t_j.

If an upstream or downstream relationship exists between the test being considered and every other test, then the test being considered satisfies the piece-wise serial requirement for at least one ordering of the system. This ordering can be given as

$$t_j = LBP_k \textit{ iff } \forall_i \exists \hat{L}_{ij},$$

where LBP_k is the logical cut point, and $\hat{L}$ indicates a higher order direct logical relationship between every other test (either a feed or dependency). For the system of Figure 3, the logical cut points are identified as t_3, t_{13}, t_{14}, t_{15}, t_{16}, and t_{17}. It is interesting to note at this point that the two techniques for obtaining clean cut points gave different, but overlapping, answers.

PARTITIONING AT ARBITRARY POINTS

The techniques examined to this point center on determining the vertices (tests) around which clean cuts can be made. That is, the larger model can be broken into two independent submodels. Nature or design may not be kind enough to allow us to use any of these identified cut points. There is no guarantee that the partitioning techniques described will identify any cut points, and there is no guarantee that the identified cut points will be useful in terms of model reduction or will contain information that can be made available at the beginning of a diagnostic problem. We may be forced to choose some arbitrary test (arbitrary to the model representation, but significant in that it represents a gauge reading, panel light, or some other easily observable phenomenon).

Arbitrary partitioning is achieved by actually exercising the inference engine [17]. The basic inference metarules used within STAMP and POINTER (Table 1) can be applied to an arbitrary outcome, and the resultant submodels can be developed by collecting known information from the larger model. For example, partitioning at an arbitrary t_x would involve examining a *good* outcome of t_x and applying rules 2, 4, and 5 of Table 1. Each of the elements determined by these rules would be eliminated from the overall model, and the resulting submodel would be the one to use when a *good* outcome of t_x is observed. Similarly, we would examine a *bad* outcome of t_x by applying rules 3, 4, and 5 of Table 1. Each of the elements determined by these rules would be eliminated from the overall model, and the resulting submodel would be the one to use when a *bad* outcome of t_x is observed. Thus, the two submodels that result from an evaluation of t_x are determined. The resulting submodel development may not be independent (that is, the two submodels may contain one or more elements in common), but they retain the desired model integrity.

SUMMARY OF TECHNIQUES

We have examined three techniques for approaching the problem of diagnosis of extremely large systems. Each of the techniques can be applied recursively; that is, each of the resulting submodels can again be divided into submodels. This procedure would potentially allow a 16,000-element model to be broken into two 8,000-element models, four 4,000-element models, or eight 2,000-element models. (This latter size model is comfortably worked with on a regular basis in STAMP and POINTER.)

Table 2 considers each test in the example system given in Figure 3 as a cut point. It can be seen that not all of the clean cut points are uncovered. From Table 2, it appears that none of the clean cut points occur at an optimal point. The original matrix of Figure 3 has 46

Table 1. STAMP and POINTER Inferencing Metarules*

Rule	Condition	Action
1	t_j untestable	t_j untestable is the only inference.
2	t_j good	Every element upon which t_j depends is good.
3a	t_j bad	Every element which t_j feeds is bad.
3b	t_j bad	Every conclusion element which t_j does not depend on is good.
3c	t_j bad	Every element upon which t_j does not depend or feed is usually† not needed.
4	After rules 1-3	Every unknown element t_p that depends on all of the unknown elements is bad.
5	After rules 1-3	Every unknown element t_p that depends on only known good elements is good.

*Inference rules due to asymmetric, conditional, and linked outcome tests are not shown in table.
†Unless not evaluating the test under condition causes an additional ambiguity group.

elements, and an optimal cut would put 22 or 23 elements in the largest submatrix. This does occur with t_{10}, but that test is not a clean cut point. The submodels for this problem would have to be derived by the arbitrary partitioning technique.

FUTURE DIRECTIONS

The techniques described in this paper are only a few of many possible approaches for partitioning complex diagnostic problems. For example, in addition to identifying articulation points, it may be possible to identify a set of minimal vertex cuts, where a vertex cut is a set of vertices such that removal of all of the vertices disconnects the graph. If we specify some $k \geq 1$, then we can identify all vertex cuts of size k or less, which includes the articulation points as vertex cuts of size $k = 1$. For a given k, the time required to identify the k-vertex cut is $O(n^k)$.

Table 2. Examination of Tests as Cut Points

Test	Largest Submatrix	Articulation Point	Logical Cut Point	Clean Cut
t_1	34	No	No	Yes
t_2	42	No	No	Yes
t_3	38	No	Yes	Yes
t_4	30	No	No	Yes
t_5	35	No	No	No
t_6	25	No	No	No
t_7	25	No	No	No
t_8	25	No	No	No
t_9	25	No	No	No
t_{10}	23	No	No	No
t_{11}	25	No	No	No
t_{12}	27	No	No	No
t_{13}	35	Yes	Yes	Yes
t_{14}	39	Yes	Yes	Yes
t_{15}	42	Yes	Yes	Yes
t_{16}	42	Yes	Yes	Yes
t_{17}	39	Yes	Yes	Yes
t_{18}	25	No	No	No
int_1	44	Yes	No	Yes
int_2	44	Yes	No	Yes

In addition to applying the vertex cut technique, it is also possible to identify minimum-edge cuts in the graph. An edge cut is a set of edges such that the removal of all of the edges disconnects the graph. A minimal, nonempty edge cut is also called a bond. By applying a method such as the Ford-Fulkerson method [18] for identifying maximum flow in a network, we can find the minimum cut of the network. Because the method has polynomial time solutions, the approach is also computationally tractable. Determining if a graph is k-edge connected for arbitrary k is known to be NP-complete.

CONCLUSION

The partitioning techniques described allow the development of responsive diagnostic models of arbitrarily large size if enough readily observable information sources are used to start the diagnostic problem. These techniques are part of an overall package of techniques used in the diagnostic and testability information modeling of complex systems. Tools exist that combine smaller models, trace dependencies, and port models between machines and software systems as well as a variety of computer-aided design (CAD) interface tools.

REFERENCES

[1] Labit, M. L., et al., *Special Report on Operational Suitability (OS) Verification Study Focus on Maintainability*, Publication 1751-01-02-2395, ARINC Research Corporation, Annapolis, Maryland, February 1981.

[2] Aeronautical Radio, Inc., *Avionics Maintenance Conference Report—San Diego, 1987*, Publication 87-087/MOF-34, Annapolis, Maryland, August 1987.

[3] Naval Electronics Systems Command (ELEX-8111), *Testability Program for Electronic Systems and Equipments*, MIL-STD-2165, Washington, D.C., January 1985.

[4] Sheppard, J. W., and W. R. Simpson, "Uncertainty Computations in Model-Based Diagnostics," to appear in *AUTOTESTCON '91 Conference Record*, Anaheim, California, September, 1991.

[5] Sheppard, J. W., "Learning Diagnostic Information Using a Matrix-Based Approach to Knowledge Representation," Master of Science Thesis, G. W. C. Whiting School of Engineering of The Johns Hopkins University, October 1989.

[6] DePaul, R. A., Jr., "Logic Modeling as a Tool for Testability," *AUTOTESTCON '85 Conference Record*, Uniondale, Long Island, New York, October 1985.

[7] Franco, J. R., "Experiences Gained Using the Navy's IDSS Weapon System Testability Analyzer," *AUTOTESTCON '88 Conference Record*, Minneapolis, Minnesota, September 1988.

[8] Cantone, R. A., and P. Caserta, "Evaluating the Economical Impact of Expert Fault Diagnosis Systems: The I-CAT Experience," *3rd IEEE International Symposium on Intelligent Control*, Arlington, Virginia, August 1988.

[9] Simpson, W. R., and J. W. Sheppard, "Experiences with a Model-Based Approach to the Fault Detection and Isolation of Complex Systems," *Symposium on Artificial Intelligence Applications in Military Logistics*, Williamsburg, Virginia, March 1990.

[10] Simpson, W. R., "Active Testability Analysis and Interactive Fault Isolation," *AUTOTESTCON '87 Conference Record*, San Francisco, California, November 1987.

[11] Simpson, W. R., and B. A. Kelley, "Multidimensional Context Representation of Knowledge-Based Information," *1987 Data Fusion Symposium*, Laurel, Maryland, June 1987.

[12] Hyafil, L., and R. L. Rivest, "Constructing Optimal Binary Decision Trees is NP-Complete," *Information Processing Letters*, Vol. 5, No. 1, May 1976, pp. 15-17.

[13] Sheppard, J. W., "Notes on Partitioning by Articulation Points," STAMP Technical Note 346, ARINC Research Corporation, Annapolis, Maryland, November 1988.

[14] Simpson, W. R., "Notes on Partitioning by Logical Break Points," STAMP Technical Note 350, ARINC Research Corporation, Annapolis, Maryland, July 1989.

[15] Sheppard, J. W., "Notes on Closure," STAMP Technical Note 340, ARINC Research Corporation, Annapolis, Maryland, July 1988.

[16] Simpson, W. R., "Notes on Logical Closure," STAMP Technical Note 340A, ARINC Research Corporation, Annapolis, Maryland, November, 1988.

[17] Simpson, W. R., "Notes on Direct Matrix Partitioning," STAMP Technical Note 352, ARINC Research Corporation, Annapolis, Maryland, August 1989.

[18] Corman, T. H., C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill Book Company, New York, 1990, pp. 587-600.