Fault Isolation in an Integrated Diagnostic Environment

WILLIAM R. SIMPSON
JOHN W. SHEPPARD
Arinc Research Corporation

In part 5 of their series on integrated diagnostics,[1-4] the authors use the information flow model to determine diagnostic strategies. Of particular concern is optimizing diagnosis to minimize some objective cost function. They discuss a technique that can include multiple cost criteria such as test time, skill level, and failure frequency, as well as information value.

A MODEL-BASED APPROACH to integrated diagnostics permits engineers to design complex systems that make effective use of test resources. The information flow model, which we detailed in part 2 of this series, consists of two primitive elements: information sources, or tests, and fault isolation conclusions. We use the terms *tests* and *information sources* interchangeably because a test is any source of information about the system's state of health. Thus, tests include traditional stimulus-response pairs associated with direct probing, the observation of functional behavior, the detection of physical anomalies such as smoke or charred mountings, and boundary-scan data.[5] Fault isolation conclusions include specific failure modes for components, chips, cards, assemblies, and entire subsystems and systems. Further, the set of conclusions in any model may include all these conclusion types.

The first step an engineer must take in modeling a system is to determine the level of diagnosis appropriate for that system. If necessary, the engineer then partitions the system into a hierarchy of subsystems (implying a hierarchy of models) that achieves the desired level of diagnosis. Next, the engineer must determine sets of tests and conclusions that will isolate faults to each line-replaceable unit, board, or individual component. If the system is still in the design phase, the engineer must apply judgment to hypothesize appropriate sets. As the design matures, the diagnostic hierarchy becomes more fixed. Thus, model development parallels the design process.

In constructing a diagnostic model, the most important and most difficult step is determining the dependency relationships among tests and between tests and conclusions. Dependency determination approaches include simulation, failure-modes-and-effects analysis, dataflow analysis, logic flow analysis, and traditional, manual circuit analysis. In all of these approaches, the engineer examines each test and asks two important questions:

- What inferences can be drawn from a *passed* outcome?
- What inferences can be drawn from a *failed* outcome?

Our second article explained how to derive dependencies from the answers to these questions.[2]

In addition to determining dependen-

0740-7475/93/0300-0052$03.00 © 1993 IEEE

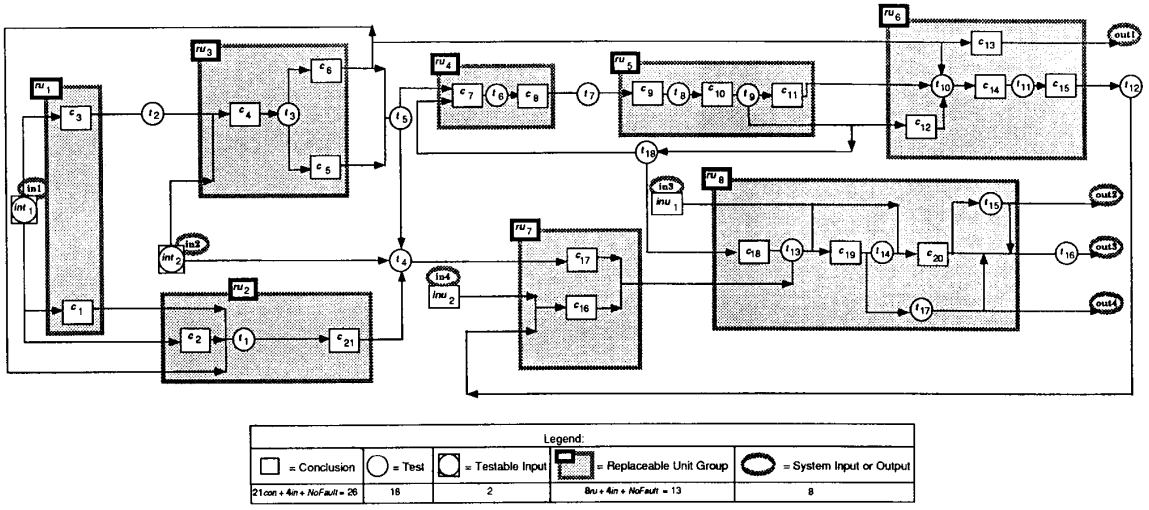


Figure 1. *Dependency diagram of antitank missile launcher case study.*

cies, the engineer must specify the following information for the model, also covered in the second article:

- hierarchical grouping of tests
- hierarchical grouping of conclusions
- cost weights for tests
- probability data for conclusions
- required test sequence
- test inference types
- multiple failures

Once the model has been constructed, an iterative process of analysis and redesign begins. First, the engineer analyzes system testability.[3,4] From this analysis, the engineer derives design recommendations and determines ways to improve testability and eliminate redundancy. The process continues until the design is final or further improvements would not be cost-effective.

Following the testability analysis cycle, the engineer has several diagnostic options. The appropriate choices depend on the type of system and the level of diagnosis required. For example, if the diagnostic capability is to be embedded, the engineer can use the model to derive built-in test procedures or to develop a comprehensive on-board maintenance system. If the system will be tested by automated test equipment, the engineer can derive the test executive's test sequence from the model or can replace the test executive with a model-based reasoner. For manual diagnosis, analysts can construct efficient fault trees from the model for a technical manual, either paper or electronic. In most cases, the engineer will combine these diagnostic options to create the most efficient form of maintenance architecture.

This article describes how to use the information flow model to conduct efficient fault isolation. We use the antitank missile launcher case study described throughout the series to present the basic fault isolation approaches, derive an information theory approach to optimization, and apply factors for multicriterion optimization. Figure 1 is a dependency diagram for the case study. Figure 2 (next page) is an overview of the model-based process.

Fault isolation theory

Fault isolation can be described mathematically as a partitioning process. For a given system we consider several conclusions, at least one of which is the correct conclusion. We call this set of conclusions **F**.[3] At some point in testing, we have a set of conclusions considered feasible solutions to the diagnostic problem. We call this set **G**. If we have not performed any testing, **G** is equivalent to **F**. Performance and evaluation of a test will allow us to infer two subsets of conclusions: those that are feasible, **G**, and those that are no longer feasible, **H**. Note that reasoning with certainty requires a conclusion to be in either **G** or **H** but not in both sets. (Reasoning under uncertainty relaxes this assumption and will be discussed in a later article in the series.) Thus, given **G** and **H**, it should be clear that

$$\mathbf{G} \cup \mathbf{H} = \mathbf{F} \text{ and}$$
$$\mathbf{G} \cap \mathbf{H} = \varnothing \text{ (the empty set)}$$

Initially, **H** has no members. A test outcome implies that a list of conclusions is no longer feasible, either direct-

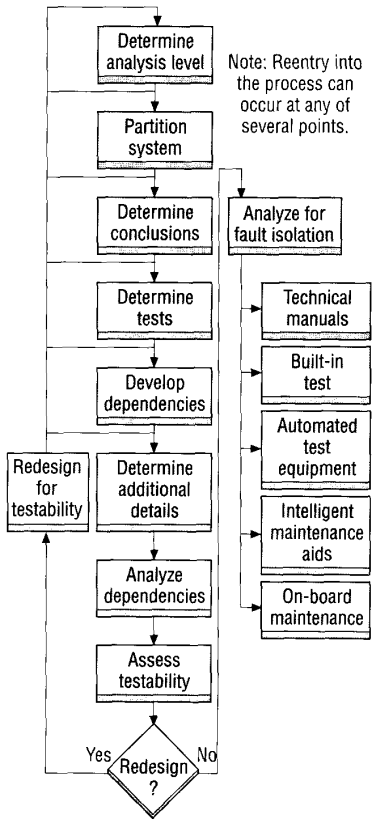


Figure 2. *Model-based integrated diagnostic process.*

ly, by determining that the conclusions are false, or indirectly, by determining that some other set of conclusions must contain the answer. As we proceed with testing, **G** should decrease in size, and **H** should increase in size. Ultimately, a strategy isolates a failure when any of the following is true:

- **G** consists of a single conclusion.
- **G** consists of an ambiguity group that we cannot partition further.
- We have no more tests to evaluate.
- No conclusions remain in **G**.

The process for determining which elements belong in **G** and which elements belong in **H** is called inference,

Tests

Tests	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	int_1	int_2
t_1	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
t_2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_3	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_4	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
t_5	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_6	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_7	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_8	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_9	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
t_{10}	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0
t_{11}	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0
t_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0
t_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
t_{14}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0
t_{15}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
t_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
t_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0
t_{18}	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
int_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0
int_2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0

(a)

Conclusions

Conclusions	int_1	int_2	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}	c_{17}	c_{18}	c_{19}	c_{20}	c_{21}	inu_1	inu_2	No Fault
int_1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
int_2	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
c_1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
c_2	1	0	0	1	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
c_3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_4	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_5	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_6	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_7	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_8	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_9	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{10}	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{11}	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{12}	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{13}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c_{14}	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{15}	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0
c_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
c_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
c_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0
c_{20}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
c_{21}	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0
inu_1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0
inu_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0
No Fault	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(b)

Figure 3. *Closed dependency matrices for case study: test-to-test dependency matrix* ***(a)****; test-to-conclusion dependency matrix* ***(b)****. 1 = dependency; 0 = no dependency.*

Table 1. Inference metarules for the information flow model.

1. IF test t_i is untestable, THEN no inference is available except that t_i is untestable.
2. IF test t_i has a passed outcome, AND IF test t_i is not negative inference, THEN every member of the information source set **I** that is not otherwise inferred as passed or failed and upon which t_i depends is inferred to have a passed outcome.
3. IF test t_i has a passed outcome, AND IF test t_i is not negative inference, THEN every member of the fault isolation set **F** upon which t_i depends is inferred to be a member of the infeasible set **H** and not a member of the feasible set **G**.
4. IF test t_i has a failed outcome, AND IF test t_i is not positive inference, THEN every member of the information source set **I** that is not otherwise inferred passed or failed and that depends upon t_i is inferred to have a failed outcome.
5. IF test t_i has a failed outcome, AND IF test t_i is not positive inference, THEN every member of the fault isolation set **F** upon which t_i does not depend is inferred to be a member of the infeasible set **H** and not a member of the feasible set **G**.
6. IF test t_i has a failed outcome, AND IF test t_i is not positive inference, THEN every member of the information source set **I** (t_q) that is not otherwise inferred passed or failed and that does not depend upon t_i, and for which t_i does not depend upon t_q is inferred to be not needed UNLESS the lack of evaluation of this test will cause additional ambiguity in the feasible set **G**.
7. IF any member of **I** (t_p) that is not otherwise known to be passed or failed contains only members of **F** as dependencies that are also members of **H**, THEN t_p is inferred to have a passed outcome.
8. IF any member of **I** (t_p) that is not otherwise known to be passed or failed contains all the members of **G** as dependencies, THEN t_p is inferred to have a failed outcome.
9. IF test t_i has a failed outcome, AND IF the failed outcome of test t_i is cross-linked to t_q, THEN t_q is inferred to have a passed outcome, and inference metarules 1 through 8 are applied to t_q.
10. IF test t_i has a passed outcome, AND IF the passed outcome of test t_i is cross-linked to t_q, THEN t_q is inferred to have a failed outcome and inference metarules 1 through 8 are applied to t_q.
11. IF test t_i has a failed outcome, AND IF the failed outcome of test t_i is cross-linked to Test Group A, THEN each test t_q that is a member of Test Group A is inferred to have a passed outcome, and inference metarules 1 through 8 are applied to each such t_q.
12. IF test t_i has a passed outcome, AND IF the passed outcome of test t_i is cross-linked to Test Group A, THEN each test t_q that is a member of Test Group A is inferred to have a failed outcome, and inference metarules 1 through 8 are applied to each such t_q.
13. IF test t_i has a passed outcome, AND IF the passed outcome of test t_i is linked to Test Group A or t_q as untestable, THEN t_q or each test t_k that is a member of Test Group A is inferred to have an untestable outcome, and inference metarule 1 is applied to t_q or each such t_k.
14. IF test t_i has a failed outcome, AND IF test t_i is linked to Test Group A or t_q as untestable, THEN t_q or each test t_k that is a member of Test Group A is inferred to have an untestable outcome, and inference metarule 1 is applied to t_q or each such t_k.

and the software that performs this process is called an inference engine.

Inference using matrix representation

Figure 3 shows the closed dependency matrices for the case study. (We addressed the development of the matrices, including special test paradigms, in part 2 of the series.) In Figure 3, test dependency is represented by a 1 and nondependency by a 0. Assuming that dependency is symmetric, we can use the matrix structure for direct inference. (If special test paradigms require special inference, we must use other forms of logical inference.) We label the columns of the dependency matrices "depends" and the rows "feeds," and we use the depends and feeds as an inference mechanism.

For example, in Figure 3b, t_8 depends on int_1, int_2, c_3 c_4, c_5, c_6, c_7, c_8, c_9, and c_{10}. If t_8 passes, each fault isolation conclusion will be infeasible (in **H**). In Figure 3a, t_8 feeds t_6, t_7, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, and t_{18}. If t_8 fails, each feed will also fail, and each of the depends will be feasible (in **G**, unless they were otherwise determined infeasible). We can confirm these inferences by examining Figure 1.

A rule used to determine which inferences can be drawn is called an inference metarule (a rule that handles rules). Table 1 lists one possible set of inference metarules for the case study information flow model. These are the rules we will use for our analyses.

We can verify most of the metarules by carefully inspecting the case study and its matrices. Two points are of interest. First, rule 6 is complex to determine,

requiring a special algorithm to satisfy the UNLESS portion. Second, rules 9 through 14 describe inferences not included in the matrices (rules 1 through 8 describe inferences inherent in the matrices). Such inferences include linked-outcome tests and asymmetric tests. Note that all the rules assume a single failure; we discuss the impact of this assumption in the next section.

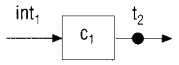


Figure 4. *Simple fault isolation example.*

Developing fault isolation strategies

The problem we discuss here is how to choose tests and how to determine the order in which to perform the tests to localize a fault to a given level. When we conduct diagnosis, we do not know which element has failed, so we assign an unknown outcome to each member of the feasible set **G**. In addition, we assign an unknown outcome to each test we will consider. A goal of diagnosis is to minimize the average number of tests needed to isolate the fault. An entire fault isolation strategy comprises a combination of sequences that form a decision tree. The leaves of the tree are the fault isolation conclusions, and the internal nodes of the tree are tests. Each test has a branch for each of its outcomes.

One might ask, "Why all the concern with efficient search methods? With high-speed computers, why don't we just compute all the possible test sequences and choose a set that solves the required diagnostic problem according to whatever efficiency criteria we specify?" To understand why this is not possible, we must first understand the nature of the calculations involved and the complexity of the possible solution set.

To demonstrate, Figure 4 presents a simple, solvable problem. In the example system, **F** has three members (int_1, c_1, and *No Fault*), and **I** has two members (int_1 and t_2). Table 2 lists the 18 possible test sequences for the system. We reach the conclusions in the table by applying the metarules in Table 1. We derive a simple equation that provides an upper bound on the number of possible sequences (Theorem 1 in the box at right provides a proof):

$$f(m, |\mathbf{I}|) = m^{|\mathbf{I}|}|\mathbf{I}|! \quad (1)$$

where $|\mathbf{I}|$ is the number of tests, m is the number of outcomes of each test, and $f(m, |\mathbf{I}|)$ is the number of test sequences that can be constructed.

An entire fault isolation strategy consists of a combination of test sequences selected for their compatibility. We can make one tree from sequences 1 through 9 and a second tree from sequences 10 through 18 because every sequence in a tree must start with the same information source. If we choose sequences 1, 3, 4, 5, 6, 7, and 9, we have a viable tree. We exclude sequences 2 and 8 because they are subsumed by sequence 4. Figure 5 shows the resultant tree structure.

Table 2. *Example test sequences and outcomes.*

Sequence	Conclusion	Comments
1. int_1=passed t_2=passed	*No Fault*	Unlike sequence 10, this sequence cannot be terminated early.
2. int_1=failed t_2=passed	Inconsistency	With single-failure assumption, we could terminate after int_1.
3. int_1=passed t_2=failed	c_1	
4. int_1=failed t_2=failed	int_1	See sequence 2.
5. int_1=untested t_2=passed	*No Fault*	
6. int_1=untested t_2=failed	int_1 or c_1	
7. int_1=passed t_2=untested	c_1 or *No Fault*	
8. int_1=failed t_2=untested	int_1	See sequence 2.
9. int_1=untested t_2=untested	int_1 or c_1 or *No Fault*	No testing has been performed.
10. t_2=passed int_1=passed	*No Fault*	With single-failure assumption, we could terminate after t_2.
11. t_2=passed int_1=failed	Inconsistency	See sequence 10.
12. t_2=passed int_1=untested	*No Fault*	See sequence 10.
13. t_2=failed int_1=passed	c_1	
14. t_2=failed int_1=failed	int_1	Unlike sequence 4, this sequence cannot be terminated early.
15. t_2=failed int_1=untested	int_1 or c_1	
16. t_2=untested int_1=passed	c_1 or *No Fault*	
17. t_2=untested int_1=failed	int_1	Unlike sequence 8, this sequence cannot be terminated early.
18. t_2=untested int_1=untested	int_1 or c_1 or *No Fault*	See sequence 9.

Proofs of Equations 1, 2, and 3

Theorem 1

Given a set of $|\mathbf{I}|$ tests, each test having m outcomes, then at most $f(m, |\mathbf{I}|)$ test sequences can be constructed, where

$$f(m, |\mathbf{I}|) = m^{|\mathbf{I}|} |\mathbf{I}|! \quad (1)$$

Proof

By induction on $|\mathbf{I}|$.

For the base case, suppose $|\mathbf{I}| = 1$. Only one test exists for any sequence, but that test has m outcomes. Thus, we can construct m sequences. Note that

$$f(m, 1) = m^1 1! = m$$

For the inductive step, assume that the inductive hypothesis holds for test sets of sizes ranging from $|\mathbf{I}| = 1$ to $|\mathbf{I}| = n - 1$. Assume that $|\mathbf{I}| = n$. Clearly, we can select any of the n tests to start a sequence, and that test has m outcomes. Thus, we have mn starts of sequences. After we choose the initial test, the subsequences are all of length $n - 1$. Therefore, by the inductive hypothesis, there are $f(m, n-1) = m^{n-1}(n-1)!$ subsequences. Combining these subsequences with the initial choice, we have

$$\begin{aligned} f(m,n) &= (m \cdot n)\,[m^{n-1}(n-1)!] \\ &= n\,[m^n (n-1)!] \\ &= m^n\, n\,(n-1)! \\ &= m^n\, n! \end{aligned}$$

QED

Theorem 2

Given a set of $|\mathbf{I}|$ tests, each test with m possible outcomes ($|\mathbf{I}| > 1$, $m > 0$), at most $g(m, |\mathbf{I}|)$ trees can be constructed, where, assuming $g(m, 1) = 1$,

$$g(m, |\mathbf{I}|) = m^{|\mathbf{I}|-2}\,|\mathbf{I}|! \quad (2)$$

Proof

By induction on $|\mathbf{I}|$.

For the base case, when $|\mathbf{I}| = 1$, it is trivially true that there can be only one tree. This is a singularity in our analysis. Consider $|\mathbf{I}| = 2$. Now we can select either of the two tests as the root (start) of the fault tree. Each of the subtrees for the m outcomes is determined (that is, there is only one subtree for each member of **I**). This means there are only two possible trees for $|\mathbf{I}| = 2$. Note that

$$\begin{aligned} g(m, 2) &= m^{2-2}\,2! \\ &= m^0\,2! \\ &= 2 \end{aligned}$$

For the inductive step, assume that the inductive hypothesis holds for test sets of sizes ranging from $|\mathbf{I}| = 2$ to $|\mathbf{I}| = n - 1$. Assume $|\mathbf{I}| = n$. Clearly, we can select any of the n tests as the root of the tree. There are then m subtrees of the root (one for each outcome). We know from the inductive hypothesis that each subtree is one of $g(m, n-1) = m^{n-1-2}(n-1)!$ possibilities. Therefore, we have $(mn)m^{n-3}(n-1)!$ possible fault trees (in other words, $m^{n-2}\,n! = g(m,n)$ trees).

QED

Theorem 3

Given a set of conclusions, $\mathbf{C} \cup \mathbf{IN}$, plus a special conclusion referred to as *No Fault*, the number of possible multiple conclusions that we can draw is $h(c, d)$, where

$$h(c,d) = 1 + \sum_{i=1}^{d} \binom{c}{i} \quad (3)$$

and c is $|\mathbf{C} \cup \mathbf{IN}|$, and d is the maximum number of conclusions to be considered, $d \le c$.

Proof

Theorem 3 describes a simple combinatorial expansion. Note that *No Fault* is treated separately because it must occur as a single conclusion, accounting for the 1 in Equation 3. Clearly, the other combinations are simply the sum of c choose i, which is given by the binomial coefficient. *QED*

We can also derive an equation that provides an upper bound on the number of possible trees to be considered (see Theorem 2 in box for a proof):

$$g(m, |\mathbf{I}|) = m^{|\mathbf{I}|-2}\,|\mathbf{I}|! \quad (2)$$

where $g(m, |\mathbf{I}|)$ is the number of trees that can be constructed.

In the example, $m = 3$, and $|\mathbf{I}| = 2$, so that $f(3, 2) = 18$, and $g(3, 2) = 2$. Table 3 (next page) shows the combinatorial

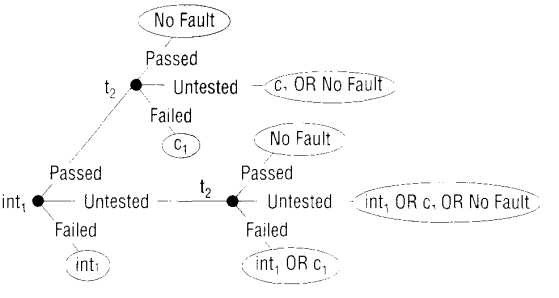


Figure 5. One resultant tree for example in Figure 4.

Table 3. Complexity contribution of test numbers to number of sequences and fault trees.

Number of tests |T|	Number of sequences $f(m, \|T\|)$	Number of fault trees $g(m, \|T\|)$
2	18	2
3	162	18
4	1,944	216
5	29,160	3,240
⋮	⋮	⋮
15	1.88×10^{19}	2.09×10^{18}
⋮	⋮	⋮
100	4.81×10^{205}	5.35×10^{204}

Table 4. Failure conclusions when considering d or fewer multiple failures.

c (Components that can fail)	d (Number of failure conclusions considered) 1	2	3	...	c
2	3	4	—	...	4
3	4	7	8	...	8
4	5	11	15	...	16
5	6	16	26	...	32
6	7	22	42	...	64
⋮	⋮	⋮	⋮	⋮	⋮
15	16	121	576	...	32,768
⋮	⋮	⋮	⋮	⋮	⋮
100	101	5,051	166,751	...	1.27×10^{30}

growth of complexity with the number of tests. Note that Equations 1 and 2 do not have the number of conclusions, |F|, as a factor, but have only information sources, |I|, and number of outcomes, m. This point is important and we will elaborate upon it later, when we discuss how to choose tests.

Most testability and fault isolation tools, as well as mathematical derivations of fault isolation paradigms, assume a single failure in the system, accurate tests, and a binary test outcome. The reason we make these restrictive assumptions is the number of combinations that would result mathematically if we considered all possible alternatives. We have just seen some of the problems related to the number of test outcomes considered and the number of resultant tree structures. A third simple equation reveals the combinatorial growth associated with comprehensive multiple-failure analysis (see Theorem 3 in box for a proof):

$$h(c,d) = 1 + \sum_{i=1}^{d} \binom{c}{i} \quad (3)$$

where c is the number of components, d is the number of failures to be considered at one time, and $h(c, d)$ is the resulting number of complex conclusions to be considered.

If we have two components and wish to consider all combinations of failures, there are four possible conclusions: component 1, component 2, both components, and *No Fault*. With three components, these numbers increase to eight conclusions; with four components, there are 16 possible conclusions. Table 4 shows the number of failure conclusions possible, assuming d or fewer failures.

The phenomenal computational complexity associated with multiple failures and multiple test outcomes illustrates the complex nature of the diagnostic problem. Our restrictive assumptions reduce the domain of the problem to a more manageable level and yet provide reasonable testability of the system. The data in Table 4 also illustrate just how critical multiple-failure analysis is to identifying likely multiple failures to be included in the model.

Choosing tests

Any problem that grows in complexity to the degree just described is intractable. Exhaustive search, therefore, is not practical, and we must consider alternative diagnostic approaches. We are looking for a method for choosing tests to minimize combinatorial growth and still produce efficient strategies.

Directed search. We will use the case study (Figure 1) to begin developing a sense of how to choose a test to evaluate. The case study system has four outputs—one coming from c_{13} and three coming from ru_8. One of the latter outputs is tested by t_{16}. If we perform t_{16} and it passes, we conclude *No Fault* or c_{13}. The metarules in Table 1 will declare that every other conclusion is false and move them into the infeasible set, **H**. In addition, the expected outcome of every test in **I** will be determined. If we perform t_{16} and it fails, we reduce the size of the feasible set by removing *No Fault* and c_{13}, leaving 24 conclusions in **G**.

Next, we follow the path from t_{16} back toward the system input. This brings us

to the next test—either t_{14} or t_{17} (the inference engine infers that t_{15} has failed). If either of these tests is performed and passes, only c_{20} remains in the feasible set. On the other hand, if either t_{14} or t_{17} fails, c_{20} becomes infeasible, leaving 23 conclusions in **G**. This process can continue until a test passes, thus "bracketing" the answer.

Technicians typically use this type of search, called directed search, when they understand how the system is built but do not have good guidance for fault isolation. Directed search almost always results in an answer if inference is performed with care, but resulting fault trees tend to be extremely inefficient. For example, if c_{20} failed, we would need only two tests to diagnose the fault, and if *No Fault* was in the system, we could arrive at that conclusion in only one test. But diagnosing c_1 required 10 tests.

Conversely, we note four inputs, two of which are testable. If we evaluate the tests of the inputs, we may immediately find a source of trouble. But if the tests pass, we have accomplished little to reduce the size of **G**, and we are faced with the dilemma of what to do next. Technical documentation occasionally specifies pretesting of inputs and sometimes outputs, but pretesting must be coupled with a clever search pattern. Table 5 (page 62) shows that the directed search and half-interval search with pretest are the most inefficient strategies for the case study.

Partitioning. A way to reduce the work load associated with testing is to concentrate on partitioning **F** into the two sets, **G** and **H**. If we assume **G** = **F**, we want to transfer part of **G** to **H**. Initially, **H** is empty, but after any test, **G** is reduced and **H** is enlarged—how much depends on the test and its outcome. Assuming binary-outcome tests, the optimal approach to searching for a fault is a binary search. For a binary search to work in diagnosis, we need to locate and evaluate the test that moves half of **G** into **H**. This is called the half-interval approach. Assuming that we can locate such a test, a *passed* test outcome would tell us that the half of our system that lies upstream (that is, the half on which the test depends) has no problem, and the answer must lie downstream. A *failed* outcome would imply the opposite. But either way, **G** decreases by |**G**|/2. Unfortunately, locating the "middle" of the system is not a straightforward exercise.

We might consider other partitioning approaches. For example, an exponential approach would select a test that is 63% from the system input.[6] Other approaches combine strategies. For example, we can select an initial test using the half-interval approach. As long as tests pass, we use the half-interval approach. Once a test fails, however, we change the test selection approach to the directed search. Each approach is based on a heuristic that makes it the most applicable in certain circumstances. We prefer to use a single approach that performs well on the general problem.

Information theory approaches. If we have a good model, such as that shown in Figure 3, and a set of metarules, such as those listed in Table 1, we can determine **G** and **H** for each test outcome and choose the test that best approximates the half-interval technique. We call this an adaptive approach because we select the test on the basis of how well it partitions the current set of unknown information sources. In other words, this approach adapts to the current state by using tests rather than conclusions.

If we examine Figure 3b, we see that t_{10} will split the 26 conclusions in half. This test is a good choice, but an exhaustive computation of the case study reveals that the best tree does not begin with t_{10}. Actually, we did not perform an exhaustive search (which would involve approximately 10^{23} trees), but we found better trees. To understand why this is so, note that some members of **F** are indistinguishable from each other (in other words, they are ambiguous). In fact, if we examine Equations 1 and 2, we realize that the conclusion set is not relevant to determining the best fault tree.

The key to efficient diagnosis lies in the information sources (the tests). One way to determine when the fault is isolated is to determine the outcomes (measured or inferred) of all the tests.

In developing our approach to optimizing the diagnostic process, we adapted Shannon's information theory.[7] Tests impart information about the diagnostic state of a system. If we choose tests that minimize the amount of uncertainty in the system, we can efficiently test the system. Our approach, entropy-directed search, consists of selecting information sources that provide the maximum reduction of system entropy (uncertainty) independently of the test outcomes. We expand this approach by weighting the tests' information value for appropriate test and conclusion factors. When we use multiple factors, we can emphasize one over another. The result is an optimization algorithm that, though not globally optimal, produces good and reasonable solutions, often at or near globally optimal limits.

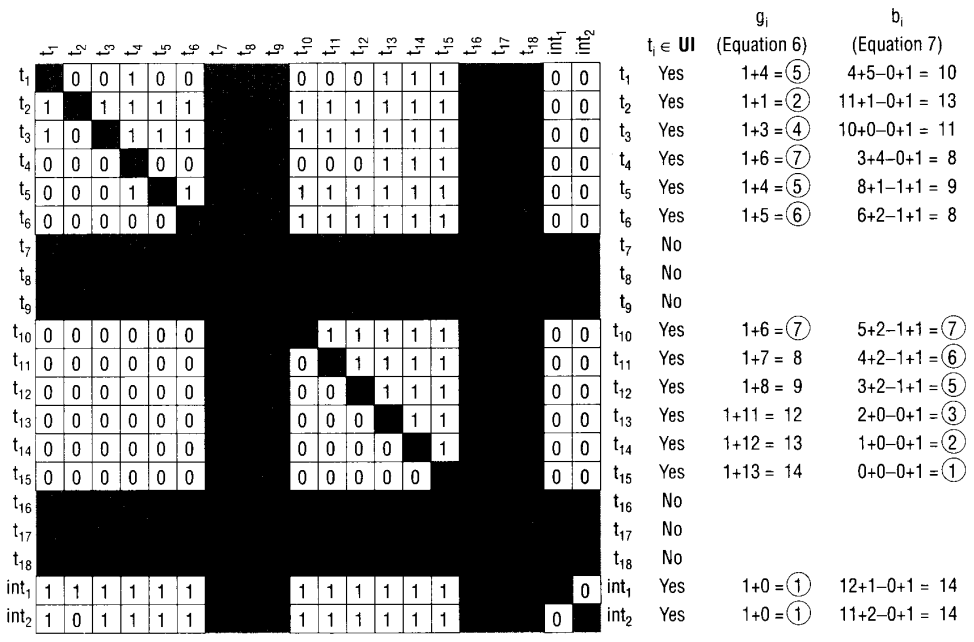
According to Shannon, the amount of information that we can obtain from an information source can be expressed as follows:

$$I(t) = \Sigma\,(-\,p \log_2 p) \qquad (4)$$

where the sum is over the range of all the possible outcomes, and p is the probability that an outcome will occur. Given that a test has a binary outcome, Equation 4 can be expanded as follows:

$$I(t) = -p(t = \text{fail}) \log_2 p(t = \text{fail}) - p(t = \text{pass}) \log_2 p(t = \text{pass}) \qquad (5)$$

Now we assume that the outcomes of a test are equiprobable: $p(t = \text{fail}) =$



	$t_i \in$ UI	g_i (Equation 6)	b_i (Equation 7)
t_1	Yes	1+4 = ⑤	4+5–0+1 = 10
t_2	Yes	1+1 = ②	11+1–0+1 = 13
t_3	Yes	1+3 = ④	10+0–0+1 = 11
t_4	Yes	1+6 = ⑦	3+4–0+1 = 8
t_5	Yes	1+4 = ⑤	8+1–1+1 = 9
t_6	Yes	1+5 = ⑥	6+2–1+1 = 8
t_7	No		
t_8	No		
t_9	No		
t_{10}	Yes	1+6 = ⑦	5+2–1+1 = ⑦
t_{11}	Yes	1+7 = 8	4+2–1+1 = ⑥
t_{12}	Yes	1+8 = 9	3+2–1+1 = ⑤
t_{13}	Yes	1+11 = 12	2+0–0+1 = ③
t_{14}	Yes	1+12 = 13	1+0–0+1 = ②
t_{15}	Yes	1+13 = 14	0+0–0+1 = ①
t_{16}	No		
t_{17}	No		
t_{18}	No		
int_1	Yes	1+0 = ①	12+1–0+1 = 14
int_2	Yes	1+0 = ①	11+2–0+1 = 14

Figure 6. *Information counts in the case study test-to-test matrix. Circled values are minimum information counts; min-max occurs at* t_4 *and* t_{10}.

$p(t = \text{pass}) = 0.5$. This assumption provides the maximum reduction in system entropy. Ideally, any test chosen should approach this limit. As a result, the consequences of using a 0.5 probability value are minimized; however, we can modify this probability with a weighting process described later. Thus,

$$I(t) = -(0.5)\log(0.5) - (0.5)\log(0.5)$$
$$= -(0.5) * (-1) - (0.5) * (-1)$$
$$= 0.5 + 0.5 = 1$$

In evaluating a test, we learn 1 bit of information for its outcome and 1 bit of information for each unique test we infer. (We defined a unique test in part 3 of the series.) Thus, we can reduce the entropy calculation to simple counting.[8] As early as 1960, Johnson addressed the problem of constructing optimal diagnostic decision trees using information measurement.[9] Hartman et al. and Addis subsequently performed additional work applying information theory and the design of diagnostics.[10,11] Chang and Kautz applied a closely related counting approach to the specific problem of diagnosing digital combinational circuits.[12,13]

For each test in the model whose outcome we do not know (through either testing or inference), we compute the amount of information, assuming the test passes and then assuming the test fails. In all these calculations we assume that we will identify a single conclusion. However, the single conclusion may be a specified multiple failure. We limit information counting to tests. Conclusions are a by-product of the inference process. We compute the information value for a test, t_i, according to these rules, which precisely follow the inference metarules in Table 1:

1. Assume t_i passes, and count the number of tests that can be inferred. This is the number of unique tests upon which t_i depends plus one (for the test itself).
2. Assume t_i fails, and count the number of tests that can be inferred. This is the number of tests that depend on t_i plus the number of tests that can be considered not needed plus one (for the test itself). The *not needed* calculation consists of examining test t_j to see if it either depends on t_i or is depended on by t_i. If neither is true, and if eliminating t_j does not create a new ambiguity, t_j is not needed.
3. Multiply the pass value and the fail value by a weight computed as described in either Equation 16 or Equation 19. Let the minimum between the weighted pass value and the weighted fail value be the test's information value.
4. Choose the test with the highest weighted information value.

When several tests have the same information value, we can apply several tie-

breakers, discussed later in this article.

To compute the tests' information values, we must define several variables. First, let g_i represent the information learned from t_i's passing. Then

$$g_i = 1 + \sum_{j=1}^{|\mathbf{I}|} \alpha_{ij}$$

$$\alpha_{ij} = \begin{cases} 1; \left(\hat{\mathbf{D}}_{ij} = 1\right) \wedge \left(i \neq j\right) \wedge \left(t_j \in \mathbf{UI}\right) \\ \qquad \wedge \left(t_j = \text{unknown}\right) \\ 0; \text{otherwise} \end{cases} \quad (6)$$

where $\hat{\mathbf{D}}$ is the closed matrix and $\hat{\mathbf{D}}_{ij}$ is the element at the ith row, jth column. Note that we do not count $i = j$ because that is represented by the 1 in Equation 6. Note also that we consider only relationships to unique tests ($t_j \in \mathbf{UI}$) and currently unknown tests. This counting algorithm follows from inference metarule 2.

Now let b_i represent the information learned from the failure of t_i. We can then compute the sum of four terms (corresponding to inference metarules 4 through 6):

$$b_i = \sum_{i=1}^{|\mathbf{I}|} \beta_{ij} + \sum_{i=1}^{|\mathbf{I}|} \gamma_{ij} - \sum_{i=1}^{|\mathbf{I}|} \delta_{ij} + 1 \quad (7)$$

where

$$\beta_{ij} = \begin{cases} 1; \left(\hat{\mathbf{D}}_{ji} = 1\right) \wedge \left(i \neq j\right) \wedge \left(t_j \in \mathbf{UI}\right) \\ \quad \wedge \left(t_j = \text{unknown}\right) \\ 0; \text{otherwise} \end{cases} \quad (8)$$

$$\gamma_{ij} = \begin{cases} 1; \left(\hat{\mathbf{D}}_{ij} = 0\right) \wedge \left(\hat{\mathbf{D}}_{ji} = 0\right) \wedge \left(i \neq j\right) \\ \quad \wedge \left(t_j \in \mathbf{UI}\right) \wedge \left(t_j = \text{unknown}\right) \\ 0; \text{otherwise} \end{cases} \quad (9)$$

and

$$\delta_{ij} = \begin{cases} 1; \left(\gamma_{ij} = 1\right) \wedge \begin{pmatrix} \text{if } t_j \text{ is deleted,} \\ \text{ambiguities in} \\ \mathbf{G} \text{ increase} \end{pmatrix} \\ 0; \text{otherwise} \end{cases} \quad (10)$$

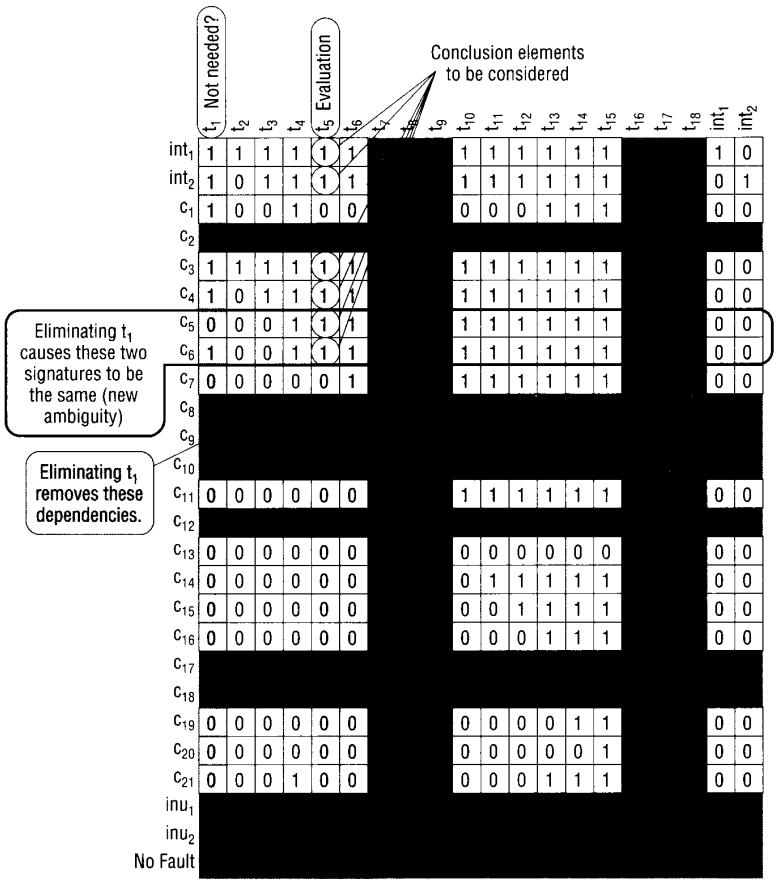


Figure 7. *Test-not-needed calculation.*

Equations 6 and 7 are similar and represent a column scan and a row scan, respectively. Equation 9 covers cases for which no explicit dependency relationships exist, and Equation 10 covers the UNLESS portion of inference metarule 6. The computation of Equation 10 is identical with the excess test computation described in part 4 of this series.

Figure 6 shows the information breakdown for the selection of the first test of the case study, with the calculations in Equations 6 and 7. Figure 7 illustrates the *test-not-needed* computation. Here we consider t_5 as having failed, and the figure shows the appropriate counting information. Equation 9 identifies t_1 as neither depending on t_5 nor being depended on by t_5, but Equation 10 removes t_5 from the count because its elimination increases ambiguity.

We make the actual test choice with a min-max algorithm as follows:

$$\phi_i = \min\{g_i, b_i\}, \quad (11)$$

$$\psi = \max_{i=1}^{|\mathbf{I}|} \phi_i \quad (12)$$

We choose t_i such that $\phi_i = \psi$. This process, derived from the minimax theorem

Table 5. Search strategies and their results for the case study.

Strategy	Description	First test	Average number of tests	Variance in number of tests
Directed	Perform all tests sequentially beginning with first known fault or output. Skip tests whose outcome can be inferred.	t_{16}	6.1	5.06
Half-interval	Test at middle of system. Proceed from outputs to inputs after a passed test, and from inputs to outputs after a failed test.	t_8	4.2	0.56
Half-interval directed	Proceed with half-interval until failed test outcome is met; then proceed with directed search.	t_8	4.3	0.76
Exponential	Same as half-interval but with a 63% partition.	t_{10}	4.5	1.60
Exponential directed	Same as half-interval directed but with a 63% partition.	t_{10}	4.7	1.96
Half-interval with pretest*	Same as half-interval but with pretest of inputs and outputs.	int_1	6.1	5.45
Entropy-directed	Method outlined in this article.	t_4	4.0	0.00

*Any of the preceding strategies can be utilized after a pretest of inputs and outputs.

of game theory, assures that the test chosen will have a robust value regardless of its outcome. If several tests have the same information value, we can apply a number of tiebreakers. For example, in the case study t_4 and t_{10} tie, so to break the tie we choose the test with the highest maximum information value as follows: For t_4, $g_4 = 7$ and $b_4 = 8$. For t_{10}, $g_{10} = 7$ and $b_{10} = 7$. Thus, max $\{g_4, b_4\}$ > max $\{g_{10}, b_{10}\}$, so we select t_4. Additional tiebreakers include counting information in the conclusion set, counting first-order inferences, and selecting the test with the largest index value. The last method has the added advantage of always breaking the tie.

Comparison of methods

Table 5 summarizes the search strategies discussed in this article and shows the average and variance of the number of tests in the diagnostic trees for the case study. The entropy-directed and half-interval approaches perform better than the other approaches; in fact, the entropy-directed approach actually yields the theoretical minimum values for binary tests for the case study. Unfortunately, we cannot prove optimality because generating optimal binary decision trees is known to be NP-complete, and we are forced to use a local search mechanism.[14] In one case, where we have a fully serial system, we can prove that the entropy-directed approach is optimal. For this case, the entropy-directed approach becomes equivalent to the half-interval approach. Further, for a large number of real systems, the entropy-directed approach has provided results at or very near the theoretical minimum number of tests.

Multicriterion optimization

So far we have assumed that factors associated with test times, test costs, and failure frequencies are either unavailable or unimportant. In real problems, of course, these factors may be more important than the number of tests conducted. For example, four tests may require several hours, while several dozen tests may take just a few minutes. The latter case may be preferable if it is just as accurate as the former, requires the same level of expertise, and all other factors are equal. We may wish to include multiple factors directly in the computation, such as the least costly isolation on premium shifts and the quickest isolation on normal shifts (the latter factor allows more work to be performed on normal shifts).

Clearly, several factors can influence the process by which we choose tests. Normally, for multicriterion optimization we develop a common factor, such as dollar value, and reduce each pertinent parameter to the common element. Our process uses information value as the common element, scaling that value in terms of relative worth.

We empirically derived a weighting scheme and verified it by application.[15–17] The weighting scheme is based on two types of parameters:

- parameters directly related to test worth, such as test cost, test time, and skill level
- parameters indirectly related to test worth through the conclusions in the analysis, such as component failure frequency and component criticality

Using these weights, we modified the min-max algorithm to operate on weighted information value:

$$\phi_i^* = \min\{w_i' g_i, w_i'' b_i\} \quad (13)$$

$$\psi^* = \max_{i=1}^{|\mathbf{t}|} \phi_i^* \quad (14)$$

We choose t_i such that $\phi_i^* = \psi^*$. We use w_i' for the weighting applied to a *passed* test outcome and w_i'' for the weighting applied to a *failed* outcome because in general the two weights may be different. Note that we can combine multiple types of weights by multiplying weights together. Of course, the terms in the product must first be normalized or standardized. Thus, a weight, w_i, is simply computed as

$$w_i = \prod_{j=1}^{n} w_{ji}^* \quad (15)$$

where w_{ji}^* is the normalized weight of type j associated with test t_i, and n is the number of weighting types.

Weighting for direct parameters. For the direct parameters, we use the convention that a larger value indicates a less desirable test. This is consistent with the normal range of values assigned to weights, such as test cost or test time. Thus, we expect the value of information provided by a test to decrease as the direct parameter increases. In other words, the weight applied should be inversely proportional to the direct parameter:

$$w_i = \kappa_1 / d_i \quad (16)$$

where d_i is the value of the direct parameter, κ_1 is a normalization constant, and w_i is the weight that will be applied to the information measures. For the direct parameters, κ_1 is given by

$$\kappa_1 = \left(\sum_{i=1}^{|\mathbf{t}|} \left(\frac{1}{d_i} \right) \right)^{-1} \quad (17)$$

Thus w_i will always be a number between 0 and 1.

Weighting for indirect parameters. For the indirect parameters, we apply the weights to the conclusions in the model and derive weights for the tests. Therefore, we use the convention that a larger value indicates higher priority in considering the conclusion. This scaling is consistent with the normal range of values assigned to weights, such as failure rate or criticality. Using indirect parameters, we determine the weight applied to the test through the dependency relationships stored in the matrix. First, we compute the test contribution, tc_i:

$$tc_i = \sum_{i=1}^{|\mathbf{F}|} a_j$$
$$\text{where } a_j = \begin{cases} e_j; \hat{\mathbf{D}}_{ji} = 1 \\ 0; \text{otherwise} \end{cases} \quad (18)$$

where e_j is the indirect parameter value attached to conclusion c_j. More simply, tc_i is the sum of all e_j that apply to the test. Then

$$w_i = \kappa_2 \, tc_i \quad (19)$$

where κ_2 is a normalization constant determined by

$$\kappa_2 = \left(\sum_{i=1}^{|\mathbf{t}|} tc_i \right)^{-1} \quad (20)$$

This assures that w_i will have a value between 0 and 1.

Weighted test choices for case study. Table 6 (next page) presents the computed weights to be applied in fault-isolating the case study. We took the data for test times, skill levels, and failure rates from part 2 of our series.[2] We computed the normalizers with Equations 16 (for direct weights) and 19 (for indirect weights). We multiplied the information counts by these weights, as shown in Table 7.

This formulation considers each parameter equally. That may not be at all desirable, and an analyst may wish to emphasize one parameter over another. For example, in the last column of Table 7, where all test weights are treated equally, t_{11} is chosen. Table 6 shows that t_{11} has a high skill level, and we may wish to emphasize the skill level requirement and reduce the skill level of the first test chosen. Because we have normalized the weights, we can use an emphasis factor that applies an exponent to the parameter to be emphasized. Then we can compute the weighted information value of a test as follows:

$$val_i = \left(\prod_{j=1}^{n} \left(w_{ji}^* \right)^{\eta_j} \right) \left(\min\{g_i, b_i\} \right) \quad (21)$$

where w_{ji}^* is the normalized weight of type j for test t_i, η_j is the emphasis exponent applied to weights of type j, and n is the number of weight types. We then choose t_i such that

$$val_i = \max_{j=1}^{|\mathbf{t}|} (val_j)$$

For example, suppose the three weights to be applied are skill level, test time, and failure rate, corresponding to types 1, 2, and 3, respectively. Then, if $\eta_1 = 2$, $\eta_2 = 1$, and $\eta_3 = 1$ (emphasizing the skill level factor), the test choice of the first column of Table 7 changes from t_{11} to t_{18}. Note that t_{18} has a considerably lower skill level.

Constructing the fault tree. The test choice algorithm is central to constructing a fault tree, but it is not complete; it must be combined with a problem reduction strategy and a tree traversal strategy. Figure 8 (page 66) shows the nodal-pivoting tree traversal strategy we use to develop fault trees. This technique, based on a pre-order (depth-first) tree traversal, is computationally efficient and can easily be adapted to tests with multiple outcomes. The technique is described in detail by Cormen, Leiserson, and Rivest.[18] The nodal pivot point

Table 6. *Case study weight computations.*

Test	Skill	Time	Skill weight	Time weight	Failure rate sum for test	Failure rate weight	Conclusion	Failure rate
							c_1	5
							c_2	5
							c_3	100
t_1	3	2.00	0.0537	0.0179	330	0.03415	c_4	100
t_2	3	2.20	0.0537	0.0162	110	0.01138	c_5	100
t_3	5	2.40	0.0322	0.0149	220	0.02276	c_6	100
t_4	4	1.50	0.0403	0.0238	530	0.05484	c_7	5
t_5	4	1.30	0.0403	0.0275	420	0.04346	c_8	5
t_6	3	3.00	0.0537	0.0119	440	0.04553	c_9	5
t_7	5	1.00	0.0322	0.0357	440	0.04553	c_{10}	5
t_8	3	2.00	0.0537	0.0179	440	0.04553	c_{11}	5
t_9	4	0.50	0.0403	0.0714	440	0.04553	c_{12}	5
t_{10}	6	0.60	0.0268	0.0595	450	0.04656	c_{13}	5
t_{11}	7	0.10	0.0230	0.0357	550	0.05619	c_{14}	100
t_{12}	2	0.90	0.0806	0.0397	650	0.06725	c_{15}	100
t_{13}	3	1.20	0.0537	0.0298	785	0.08122	c_{16}	5
t_{14}	4	1.60	0.0403	0.0223	800	0.08273	c_{17}	5
t_{15}	2	1.50	0.0806	0.0238	900	0.09312	c_{18}	5
t_{16}	3	2.00	0.0537	0.0179	900	0.09312	c_{19}	5
t_{17}	3	1.50	0.0537	0.0238	800	0.08273	c_{20}	100
t_{18}	3	0.30	0.0537	0.0119	440	0.04553	c_{21}	100
int_1	2	1.00	0.0806	0.0357	10	0.00104	int_1	10
int_2	3	1.00	0.0537	0.0357	10	0.00104	int_2	10
							inu_1	10
							inu_2	10

Normalizers

Skill (Equation 17): 0.1610

Time (Equation 17): 0.0357

Failure rate sum for test (Equation 20): 0.0001035

is where we assign and process test outcomes, using the inference metarules in Table 1. We repeat the test choice process, using the reduced set of unknown tests and feasible answers, until we can draw conclusions.

The remaining problems include storing intermediate answers (the reduced problem set) to allow backtracking, and terminating the diagnostic sequence. We accomplish the former through recursion or an explicit stack structure, which we will not detail here.

We terminate a diagnostic sequence through the inference process; Table 8 gives applicable inference metarules. In our experience, rule 15 is the only one that terminates a sequence; the metarules in Table 1 force rule 15 to be satisfied at the same time as rules 16 and 17. Under rule 15, the entire diagnostic process considers only tests and scarcely looks at the conclusions until it declares termination. Only inference metarule 6 and second-level tiebreakers use the conclusion set at all, in determining which tests to use and in which sequence to use them. Thus, entropy-directed search differs significantly from other approaches to diagnosis, which heavily emphasize conclusions in the development of test strategies.

THE THEORETICAL EXPLANATION in this article lays a solid foundation on which we can approach system diagnosis. The techniques described here have been applied in a number of cases with solid cost benefits—and sometimes spectacular results (order-of-magnitude improvements). The first article of the series described some of these real-world applications.

We have, however, ignored many aspects of real problems. For example, diagnosis frequently begins with our knowledge of specific symptoms a system is exhibiting. These symptoms provide valuable information that can significantly reduce the number of faults to be considered (and the number of tests to be performed). We would like to include this information in generating fault trees.

Furthermore, in our last two articles we posed several testability problems that we can address (and maybe even solve) by means of existing tests and the information flow model. For example, we can use excess and redundant tests to identify false alarms as they occur. For us to use these tests, fault trees must take them into account, which means that certain inferences may need to be suppressed.

Finally, we can use the group constructs in the model to guide hierarchical diagnosis and to further control test selection, adapting the optimization and inference procedures to provide maximum flexibility and realistic testing. We

Table 7. Weighted test choice.

Test	MIN* $\{g_i, b_i\}$	Weight: Minimum skill	Weight: Minimum time	Weight: Most frequently failing item	Weight: Least time for most frequent failure	Weight: Minimum skill, least time for most frequent failure
t_1	5	0.2685	0.0895	0.17075	0.0030564	0.0001641
t_2	2	0.1074	0.0324	0.02276	0.0003687	0.0000198
t_3	4	0.1288	0.0596	0.09104	0.0013565	0.0000437
t_4	**7†**	0.2821	0.1666	**0.38388†**	0.0091363	0.0003682
t_5	5	0.2015	0.1375	0.21800	0.0059950	0.0002416
t_6	6	0.3222	0.0714	0.27318	0.0032508	0.0001746
t_7	6**	0.1932	0.2142	0.27318	0.0097525	0.0003140
t_8	6**	0.3222	0.1074	0.27318	0.0048899	0.0002626
t_9	6**	0.2418	0.4284	0.27318	0.0195051	0.0007861
t_{10}	7	0.1876	0.4165	0.32592	0.0193922	0.0005197
t_{11}	6	0.1380	**2.1420†**	0.33714	**0.1203590†**	**0.0027683†**
t_{12}	5	0.4030	0.1985	0.33625	0.0133491	0.0010759
t_{13}	3	0.1611	0.0894	0.24366	0.0072611	0.0003899
t_{14}	2	0.0806	0.0446	0.16546	0.0036898	0.0001487
t_{15}	1	0.0806	0.0238	0.09312	0.0022163	0.0001786
t_{16}	1**	0.0537	0.0179	0.09312	0.0016668	0.0000895
t_{17}	2**	0.1074	0.0476	0.16546	0.0039379	0.0002115
t_{18}	6**	**0.3222†**	0.7140	0.27318	0.0325084	0.0017457
int_1	1	0.0806	0.0357	0.00104	0.0000371	0.0000030
int_2	1	0.0537	0.0357	0.00104	0.0000371	0.0000020
Best test	t_4	t_{18}	t_{11}	t_4	t_{11}	t_{11}

** Value from Figure 6*

*** Value taken from unique element of redundant group[3]*

† Indicates test choice

will discuss these and additional issues in part 6 of this series, in which we will develop specific fault trees for the case study. D&T

References

1. W.R. Simpson and J.W. Sheppard, "System Complexity and Integrated Diagnostics" (part 1 of series), *IEEE Design & Test of Computers*, Vol. 8, No. 3, Sept. 1991, pp. 16-30.
2. J.W. Sheppard and W.R. Simpson, "A Mathematical Model for Integrated Diagnostics" (part 2), *IEEE Design & Test of Computers*, Vol. 8, No. 4, Dec. 1991, pp. 25-38.
3. W.R. Simpson and J.W. Sheppard, "System Testability Assessment for Integrated Diagnostics" (part 3), *IEEE Design & Test of Computers*, Vol. 9, No. 1, Mar. 1992, pp. 40-54.
4. J.W. Sheppard and W.R. Simpson, "Applying Testability Analysis for Integrated Diagnostics" (part 4), *IEEE Design & Test of Computers*, Vol. 9, No. 3, Sept. 1992, pp. 65-78.
5. C. Maunder and R. Tulloss, eds., *The Test Access Port and Boundary-Scan Architecture*, IEEE Computer Society Press, Los Alamitos, Calif., 1990.
6. M.L. Cramer et al., *Application of Advanced Testability Design and Fault Isolation Analysis*, USAAVRADCOM 83-D-28, Applied Technology Laboratory, U.S. Army Research and Technology Laboratories (AVSCOM), Fort Eustis, Va., 1984.
7. C.E. Shannon, "A Mathematical Theory of Communications," *Bell System Technical J.*, Vol. 27, 1984, pp. 379-423.
8. B.A. Kelley and W.R. Simpson, "The Use of Information Theory in Propositional Calculus," *1987 Data Fusion Symposium*, Johns Hopkins Univ., Laurel, Md., 1987, pp. 125-136.
9. R.A. Johnson, "An Information Theory

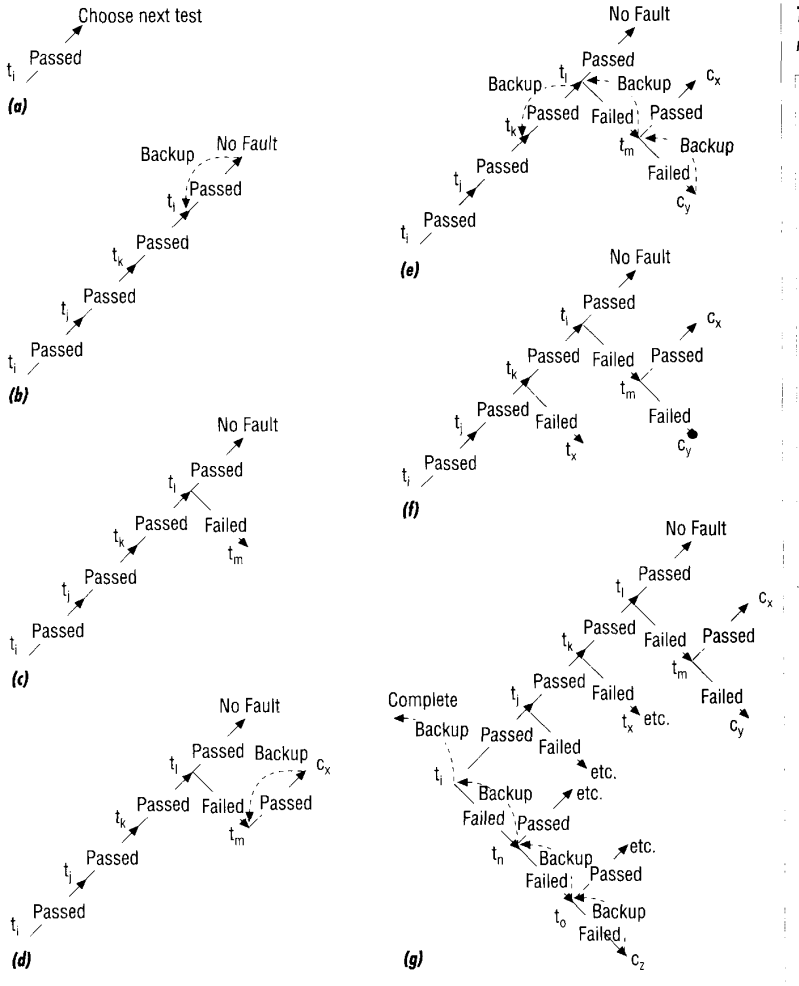


Figure 8. Nodal-pivoting tree traversal strategy (in sequence a-g).

Table 8. *Inference metarules for the termination of fault isolation.*

15. IF no test t_i is unknown, THEN terminate—the answer is the remaining feasible set **G**.
16. IF only a single answer remains in the feasible set **G**, THEN terminate—the answer is the remaining feasible set **G**.
17. IF every member of the feasible set **G** is ambiguous with every other member of the feasible set, THEN terminate—the answer is the remaining feasible set **G**.
18. IF a backup move is ever initiated from the first node of a fault tree, THEN terminate—fault tree is complete.

Approach to Diagnostics," *Proc. Sixth Annual Conf. Reliability and Quality Control*, 1960, pp. 102-109.

10. C.P. Hartman et al., "Application of Information to the Construction of Efficient Decision Trees," *IEEE Trans. Information Theory*, Vol. IT-28, No. 4, July 1982, pp. 565-577.
11. T.R. Addis, "Knowledge Refining for a Diagnostic Aid (An Example of Applied Epistemics)," *Int'l J. Man-Machine Studies*, Vol. 17, 1982, pp. 151-164.
12. H.Y. Chang, "An Algorithm for Selecting an Optimum Set of Diagnostic Tests," *IEEE Trans. Electronic Computers*, Vol. EC-14, Oct. 1965, pp. 706-711.
13. W.H. Kautz, "Fault Testing and Diagnosis in Combinational Digital Circuits," *IEEE Trans. Computers*, Vol. C-17, Apr. 1968, pp. 352-366.
14. L. Hyafil and R.L. Rivest, "Constructing Optimal Binary Decision Trees Is NP-Complete," *Information Processing Letters*, Vol. 5, No. 1, May 1976, pp. 15-17.
15. B. Pickerall et al., *Testability Analysis and Integrated Test Procedure Development for Four Modules of the NS-20 Missile Guidance Set Control Subassembly*, Tech. Report 08645-16-90-WD-ODG-0102-TP, Arinc Research Corp., Ogden, Utah, 1991.
16. B. Cash, *M1 Testability Improvement Program*, Tech. Report 3573-02-01-4472, Arinc Research Corp., Annapolis, Md., 1987.
17. B. Pickerall, *Testability Analysis and Revised Procedures for Overall Combat System Operability Test (OCSOT) USS Dale (CG-19)*, Tech. Report 3599-01-01-4528, Arinc Research Corp., Annapolis, Md., 1987.
18. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill, Cambridge, Mass., 1990.

The authors' biographical sketches and photos appear with part 4 of their series, in the September 1992 issue.

Direct questions or comments to the authors at Arinc Research Corp., Advanced Research and Development Group, 2551 Riva Rd., Annapolis, MD 21401; e-mail: sheppard@csjhu.edu or william_r_simpson@mcimail.com.