registers used to hold results of CORDIC computation will ultimately overflow for sufficiently large $n$.

It should be noted that the CORDIC scheme was not designed for sinusoid generation over an unlimited range, but rather for evaluation of trigonometric functions with bounded arguments. The complete CORDIC algorithm includes a prenormalization scheme which, however, requires prefixing of the upper limit for $n$ in (4).

## Proposed Alternate Two-Multiply Scheme

The algorithm proposed in this correspondence involves a modification of (4), such that each $c$-value is computed from the previous $c$-value and the *current* $s$-value

$$\begin{aligned} s_{n+1} &= Mc_n + s_n \\ c_{n+1} &= c_n - Ms_{n+1}, \qquad n = 0,1,2,\cdots. \end{aligned} \tag{7}$$

For purposes of comparison, we substitute the expression for $s_{n+1}$ from the first line of (7) into the second line of (7) to obtain

$$\begin{aligned} s_{n+1} &= Mc_n + s_n \\ c_{n+1} &= (1 - M^2)c_n - Ms_n, \qquad n = 0,1,2,\cdots. \end{aligned} \tag{8}$$

Expressions (8) are similar to the CORDIC algorithm (4), except for the additional term $(-M^2)$ in the coefficient of $c_n$ in the second line.

If we define $\rho$ by

$$M = 2 \sin(\rho/2) > 0 \tag{9}$$

the system (8) yields, analogously to (3) and (6)

$$\begin{aligned} s_n &= C \sin(\rho_0 - \rho/2 + n\rho) \\ c_n &= C \cos(\rho_0 + n\rho), \qquad n = 0,1,2,\cdots \end{aligned} \tag{10}$$

for

$$\begin{aligned} C &= \sqrt{s_0^2 + 2s_0c_0 \sin(\rho/2) + c_0^2}/\cos(\rho/2) \\ \rho_0 &= \tan^{-1}(s_0/(c_0 \cos(\rho/2)) + \tan(\rho/2)). \end{aligned}$$

The expressions $s_n$ and $c_n$ in (10) have constant amplitude, dependent only on initial values $(s_0,c_0)$ and on the factor $M$. Thus the algorithm is stable.

The phase separation of the $s$- and $c$-components differs from orthogonality by an offset $\rho/2$. This is essentially the price paid for having both stability and arithmetic economy. However, the generation of sinusoids in real time requires the value of only one of the components $s$ or $c$; the other is only a computing aid and is seldom needed explicitly. Thus, for *this* application, the phase shift inherent in the algorithm causes no problem. For applications that require both components and still demand orthogonality, such as generation of tangent values, the algorithm could perhaps still be used, in conjunction with additional hardware, since the amount of phase shift is a known constant. However, this would seem to negate the feature of economy which made the algorithm attractive originally.

Note that the amplitude $C$ yielded by this method, while constant, is always greater than $A = \sqrt{s_0^2 + c_0^2}$. Thus initial values $s_0$ and $c_0$ must always be adjusted to avoid overflow. Once this is done, however, there will be no overflow except that due to accumulated numerical roundoff errors.

## Summary

A comparison of the three schemes discussed here follows:

| | Trig. Identities | Unnormalized CORDIC | Proposed Scheme |
|---|---|---|---|
| Multiplies/cycle | 4 | 2 | 2 |
| Stable? | Yes | No | Yes |
| Phase deviation from orthogonality | 0 | 0 | $\rho/2$ |
| Increment angle | $\theta = \sin^{-1}M$ | $\emptyset = \tan^{-1}M$ | $\rho = 2\sin^{-1}(M/2)$ |
| Amplitude ($A = \sqrt{s_o^2 + c_o^2}$) | A | $A/(\cos \emptyset)^n$ | $\sqrt{\frac{A^2 + 2s_o c_o \sin(\rho/2)}{\cos(\rho/2)}}$ |
| Initial phase ($\theta_o = \tan^{-1}(s_o/c_o)$) | $\theta_o$ | $\theta_o$ | $\tan^{-1}(\tan\theta_o/\cos(\rho/2) + \tan(\rho/2))$ |

## References

[1] L. Levine, *Methods for Solving Engineering Problems Using Analog Computers.* New York: McGraw-Hill, 1964, pp. 114–115.

[2] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, pp. 330–334, Sept. 1959.

# Diagnosis of Intermittent Faults in Combinational Networks

ISRAEL KOREN AND ZVI KOHAVI

***Abstract*—Detection of intermittent faults was considered in a number of publications [1], [2]. Recently, Kamal [3] presented an approach to the diagnosis of intermittent faults in combinational networks. The methods developed so far, however, do not yield a minimal diagnosis experiment. Moreover, complete diagnosis is not obtained even in cases where it is possible.**

**In this correspondence we present a method for locating intermittent faults in combinational networks. This method does yield a minimal diagnosis experiment and, in addition, the maximal diagnostic resolution possible is obtained.**

**We also present a simpler alternative procedure that yields only nearly minimal experiments, but for which the required computation time is considerably shorter.**

***Index Terms*—Combinational networks, dynamic programming, fault diagnosis, intermittent faults, sequential decision tree, weighting function.**

## I. Introduction and Basic Assumptions

The problem considered in this correspondence is diagnosis of intermittent faults, and a method for locating faults of this type is presented. This method employs a simple statistical model for intermittent faults which is based upon the following assumptions.

1) At most, one out of several intermittent faults exists in the network (the single-fault assumption).

2) The faults are well behaved [1], i.e., during the application of a test the network behaves as if it is fault-free or as if a permanent fault exists.

3) The faults are signal independent, i.e., the occurrence of

Manuscript received December 23, 1975; revised December 1, 1976.

I. Koren is with the Department of Electrical Engineering and Computer Science, University of California, Santa Barbara, CA 93106.

Z. Kohavi is with the Department of Computer Science, University of Utah, Salt Lake City, UT 84112, on leave from the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa, Israel.

a fault is independent of the signal values present in the network.

Two statistical models for describing the behavior of intermittent faults were proposed in the literature. The first one is a Markov model introduced by Breuer [1]. According to this model, a network containing an intermittent fault is modeled by a two-state first-order Markov process. In the first state *FP* the fault is present and in the second state *FN* the fault is not present. The second model introduced by Kamal and Page [2] is a special case of the first one. In this case, the transition probabilities between the two states are assumed to be equal; thus the first-order Markov process turns out to be a zero-order process. Because of the complexity of the diagnosis problem discussed in this correspondence, we shall use the latter simpler model.

The network under consideration may have one out of $n$ intermittent faults denoted $f_1,f_2,\cdots,f_n$. To these faults we add $f_0$ to denote the fault-free network. Each fault has an *a priori* probability of existing in the network:

$$p_i = \Pr\{f_i\}, \qquad i = 0,1,\cdots,n$$

where,

$$\sum_{i=0}^{n} p_i = 1$$

(see in [5] a possible assignment of *a priori* probabilities).

Suppose a fault $f_i$ exists. We apply a test covering (i.e., detecting) this fault. The fault may be present during the test, causing the *failure* of the test; or the fault may not be present, causing the *success* of the test. Given that the intermittent fault $f_i$ exists in the network, the conditional probability that fault $f_i$ will be present when a test covering it is applied, is denoted

$$e_i = \Pr\{A \text{ test covering } f_i \text{ fails}/f_i \text{ exists}\}, \qquad i = 1,2,\cdots,n.$$

For $f_0$ we define $e_0 = 1$. Clearly, $e_i = 1$ if $f_i$ is a permanent fault.

We assume that for the $n + 1$ possible faults a fault matrix $R$ can be constructed. This fault matrix consists of $n + 1$ rows corresponding to the different faults and $m$ columns corresponding to $m$ possible tests.

The fault matrix is defined in the following way:

$$r_{ij} = \begin{cases} 1, & \text{if } f_i \text{ is covered by } t_j \\ 0, & \text{otherwise.} \end{cases}$$

## II. Sequential Decision Tree

The tests for detecting and locating a fault in a network can be arranged either as a sequential decision tree or as a fixed (preset) set of distinguishing tests [4], [5]. We prefer sequential diagnosis, because the average number of tests required to locate a fault is usually smaller than the number of tests required using the method of fixed diagnosis. This average number of tests serves as a cost function for the decision tree, i.e.,

$$C = \sum_{i=0}^{n} l_i p_i$$

where $l_i$ is the number of tests required to locate the fault $f_i$ with *a priori* probability $p_i$.

The sequential decision tree contains $n + 1$ terminal edges corresponding to the $n + 1$ possible faults, and $n$ vertices corresponding to the $n$ tests that are required in order to distinguish between the $n + 1$ faults [6]. (An example of a decision tree is given in Fig. 1.) We assign the numbers 1 through $n$ to the $n$ vertices in the following way: the first vertex is labeled 1, the rest of the vertices are labeled in such a way that along each path in the tree starting at vertex 1 and ending at a terminal edge the labels are in an increasing order. The $n$ tests corresponding to these $n$ vertices are denoted $t_1,t_2,\cdots,t_n$. The decision tree is described by a matrix $A$ containing $n + 1$ rows corresponding to the $n + 1$ faults and $n$ columns corresponding to the $n$ tests. This matrix is defined in the following way:

$$a_{ij} = \begin{cases} 1, & \text{if the path for } f_i \text{ includes } t_j \\ 0, & \text{otherwise.} \end{cases}$$

The $n$ tests are selected out of the $m$ possible tests in the fault matrix $R$. Some of these $n$ tests (corresponding to different paths in the decision tree) may be identical. In order to simplify notation and without loss of generality, we will assume that the tests $t_1,t_2,\cdots,t_n$ are distinct and that the first $n$ tests in the fault matrix $R$ are the tests $t_1,t_2,\cdots,t_n$.

If the possible faults in the network are permanent, then the application of a test $t_j$ at the $j$th vertex distinguishes the faults covered by $t_j$ from the faults not covered by $t_j$. If the faults are intermittent, it may occur that a fault covered by $t_j$ exists in the network but is not present when $t_j$ is applied. Hence, a wrong conclusion might be drawn regarding the existence of the fault. In order to minimize such wrong conclusions, each test is applied repeatedly so that the probability of a wrong conclusion is smaller than some prespecified $\epsilon$. For a given $\epsilon$ we shall next determine the required number $M_j$ of repetitions of the test $t_j$. Clearly, if in any application of $t_j$ a failure is observed, the repeated application of $t_j$ can be terminated and a definite conclusion can be drawn. The number of times that $t_j$ is actually repeated will be designated $m_j$. This number is a random variable satisfying

$$1 \le m_j \le M_j.$$

The maximal required number $M_j$ of repetitions will be determined from the following inequality:

$$\Pr\left\{\begin{matrix}\text{a fault covered by } t_j \text{ at vertex } j \text{ exists} \\ \text{but } t_j \text{ succeeds } M_j \text{ times}\end{matrix}\right\} \le \epsilon. \tag{1}$$

Applying the principle of total probability we obtain

$$\sum_{f_k \text{ is covered by } t_j} \Pr\{t_j \text{ succeeds } M_j \text{ times}/f_k \text{ exists at vertex } j\} \cdot \Pr\{f_k \text{ exists at vertex } j\} \le \epsilon$$

or

$$\sum_{k=1}^{n} a_{kj} r_{kj} \Pr\{t_j \text{ succeeds } M_j \text{ times}/f_k \text{ exists at vertex } j\} \cdot \Pr\{f_k \text{ exists at vertex } j\} \le \epsilon. \tag{2}$$

A solution of inequality (2) will yield the value of $M_j$ as a function of $\epsilon$. The probabilities in (2) are evaluated in the following lemmas of which the first one, Lemma 1, appears in [3] and is repeated here for the sake of completeness.

*Lemma 1:* $\Pr\{t_j \text{ succeeds } M_j \text{ times}/f_k \text{ exists}\} = (1 - e_k)^{M_j}$.

*Proof:* Since the outcomes of $M_j$ applications of $t_j$ are independent of each other and the probability of a success in each such application of $t_j$ is $1 - e_k$ ($f_k$ is covered by $t_j$) the lemma follows.

*Lemma 2:* $\lim_{\epsilon \to 0} \Pr\{f_k \text{ exists at vertex } j\} = (p_k/P_j)$, where $P_j = \Sigma_{i=0}^{n} a_{ij} p_i$ is the sum of probabilities of the faults at vertex $j$.

*Proof:* See the Appendix.

Lemma 2 enables us to derive in an efficient manner algorithms for generating minimal decision trees for diagnosis of intermittent faults. The major contribution of Lemma 2 is that it enables us to compute the *a posteriori* probabilities of the faults at each vertex regardless of the previously applied tests and their outcome. In [2] it is shown that different applications of a test may result in different *a posteriori* probabilities; and the minimal decision tree (minimal expected value of the cost function) could be generated only by exhaustive comparison of all possible decision trees. Lemma 2 overcomes this problem.

Substituting the results of the previous lemmas in equation (2) yields the following inequality for determining $M_j$:

$$\sum_{k=1}^{n} a_{kj} r_{kj} (1 - e_k)^{M_j} \frac{p_k}{P_j} \le \epsilon. \tag{3}$$

Matrix A and inequality (3) define the structure of the sequential decision tree. The value of the cost function $C$ for this

decision tree depends upon $l_i$, which is the number of tests required to locate the fault $f_i$. To evaluate $l_i$ consider the path for $f_i$ in the decision tree. This path contains tests covering $f_i$ and tests which do not cover $f_i$. The number $l_i$ is the sum of the number of applications of those tests in the path covering $f_i$ and the number of applications of those tests in the path not covering $f_i$. Since each test $t_j$ covering $f_i$ is applied $m_j$ times and each test $t_j$ not covering $f_i$ is applied $M_j$ times, we obtain

$$l_i = \sum_{j=1}^{n} a_{ij}(m_j r_{ij} + M_j \bar{r}_{ij}) \tag{4}$$

where

$$\bar{r}_{ij} = 1 - r_{ij}.$$

The fact that $m_j$ is a random variable implies that $l_i$ and the cost function $C$ are also random variables. And since $C$ is a random variable it cannot be minimized directly, and therefore we shall minimize its expected value. This expected value equals

$$E\{C\} = \sum_{i=0}^{n} p_i E\{l_i\} = \sum_{i=0}^{n} p_i E\{l/f_i\}$$

$$= \sum_{i=0}^{n} p_i \sum_{j=1}^{n} a_{ij}(r_{ij} E\{m_j/f_i\} + \bar{r}_{ij} M_j). \tag{5}$$

*Lemma 3:* $E\{m_j/f_i\} = 1/e_i\,[1 - (1 - e_i)^{M_j}]$.

*Proof:*

$$E\{m_j/f_i\} = \sum_{k=1}^{M_j} k(1 - e_i)^{k-1} e_i + M_j(1 - e_i)^{M_j}$$

$$= \frac{1 - (M_j + 1)(1 - e_i)^{M_j} + M_j(1 - e_i)^{M_j+1}}{1 - (1 - e_i)} + M_j(1 - e_i)^{M_j}$$

$$= \frac{1}{e_i}[1 - (1 - e_i)^{M_j}]. \qquad \text{Q.E.D.}$$

Substituting the result of Lemma 3 in (5) yields

$$E\{C\} = \sum_{i=0}^{n} \sum_{j=1}^{n} a_{ij} p_i \left( r_{ij} \frac{1}{e_i}[1 - (1 - e_i)^{M_j}] + \bar{r}_{ij} \cdot M_j \right). \tag{6}$$

## III. Minimal Decision Trees

A *minimal decision tree* is defined as a tree that minimizes the expected value of the cost function given by (6). The problem of generating minimal decision trees has been considered in several papers; e.g., Garey [9] considered the problem of identifying permanent faults, Matula [7] and Black [8] considered the problem of searching a target that may not be detected in a single search. The latter problem is similar to our problem; however, only a restricted case, where $R = I$ (identity matrix), was considered in [7], [8]. In this case, the decision tree turns out to be a single path so that no wrong conclusion might be drawn.

In this correspondence we generate the minimal decision tree by employing the dynamic programming method whose computational efficiency is greater than the direct enumeration method. In the following we define an objective function that will be computed recursively.

Consider a decision tree of the type shown in Fig. 1. Suppose that a number of tests have already been applied; consequently, a number of possible faults have been distinguished. Let $F$ be any subset of faults out of the $n + 1$ possible faults which remained to be distinguished at a vertex $\alpha$. Let $l_{\alpha_j}$ be the number of tests required to locate fault $f_{\alpha_j}$ out of the faults in $F$. We define the following objective function:

$$C(F) = \min \sum_{f_{\alpha j} \epsilon F} l_{\alpha_j} p_{\alpha_j}.$$

The minimization is made over all $m$ tests given in the fault matrix. This objective function is computed recursively in the following way. If $F$ contains a single fault, then $C(F) = 0$. For $F$ containing more than one fault, suppose that test $t_i$ is applied. Denote by $F_1^i(F_0^i)$ the subset of all faults covered (not covered) by test $t_i$. Then the objective function is

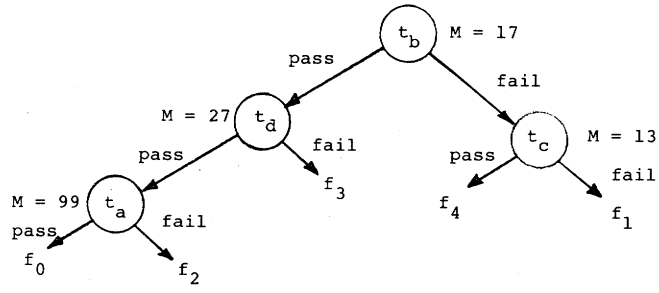


Fig. 1. A minimal decision tree for intermittent faults.

$$C(F) = \min_{t_i} \{\Delta C(F_0^i, F_1^i) + C(F_0^i) + C(F_1^i)\}. \tag{7}$$

The first term in this equation is the contribution of the test $t_i$ to the cost function (to simplify notation the subscript $\alpha_j$ is omitted)

$$\Delta C(F_0^i, F_1^i) = M \sum_{f_j \epsilon F_0^i} p_j + \sum_{f_j \epsilon F_1^i} E\{m/f_j\} p_j$$

$$= M \sum_{f_j \epsilon F_0^i} p_j + \sum_{f_j \epsilon F_1^i} \frac{1}{e_j}[1 - (1 - e_j)^M] p_j. \tag{8}$$

The maximal number of repetitions of $t_i$ is determined by substituting appropriate values into inequality (3)

$$\sum_{f_j \epsilon F_1^i} (1 - e_j)^M \frac{p_j}{P} \le \epsilon \tag{9}$$

where

$$P = \sum_{f_j \epsilon F} p_j.$$

Such a computation is performed for every possible subset $F$; and when $F$ contains all $n + 1$ possible faults, $C(F)$ equals the expected value of the cost function defined in (6).

## IV. Nearly Minimal Decision Trees

The computation time required for generating a minimal decision tree using the dynamic programming method increases rapidly. Hence, a procedure requiring a smaller computation time is desired, even if the generated decision tree will not necessarily be minimal. In this section we develop such a procedure for selecting distinguishing tests according to a weighting function [4], [5]. Using this procedure the decision tree will be constructed starting at the first vertex, contrary to the dynamic programming method in which the decision tree is constructed starting at the terminal vertices. At each vertex a local optimization is performed, minimizing a weighting function. As a weighting function we select the contribution $\Delta C(F_0^i, F_1^i)$ of this vertex to the cost function.

We define a binary vector $B$ of length $n + 1$ indicating which of the $n + 1$ possible faults is contained in $F$, i.e.,

$$b_j = \begin{cases} 1, & \text{if } f_j \epsilon F \\ 0, & \text{otherwise.} \end{cases}$$

Substituting this vector yields the following expression for $\Delta C$:

$$\Delta C = M \sum_{j=0}^{n} b_j \bar{r}_{ji} p_j + \sum_{j=0}^{n} b_j r_{ji} \frac{1}{e_j} p_j [1 - (1 - e_j)^M]. \tag{10}$$

Similarly, we obtain the following inequality for determining $M$:

$$\sum_{j=0}^{n} b_j r_{ji} (1 - e_j)^M p_j \le \epsilon \cdot P \tag{11}$$

where

$$P = \sum_{j=0}^{n} b_j p_j.$$

Since $M$ is determined from inequality (11) which requires many numerical iterations, the computation of $\Delta C$ is complicated. Hence, employing $\Delta C$ as a weighting function requires a large amount of computations, and a simpler weighting function is desired. Experience in generating decision trees for diagnosis of intermittent faults using APL programs leads to a conclusion that an upper bound of $\Delta C$ can serve as a weighting function yielding satisfactory results. The selection of such a weighting function is justified because a test minimizing the upper bound will usually minimize $\Delta C$ as well.

Equation (10) for $\Delta C$ consists of two terms. Let us denote them $\Delta_1$ and $\Delta_2$, respectively. Hence

$$\Delta C = \Delta_1 + \Delta_2.$$

In order to derive an upper bound for $\Delta C$, two different bounds are derived for the two terms $\Delta_1$ and $\Delta_2$. First we denote

$$e_{\min} = \min_{f_j \epsilon F_1'} e_j = \min_{b_j r_{ji}=1} e_j. \tag{12}$$

Let $P_1(P_0)$ designate the sum of probabilities of the faults from $F$ that are covered (not covered) by the test $t_i$. Hence

$$P_1 = \sum_{j=0}^{n} b_j r_{ji} p_j, \quad P_0 = P - P_1 = \sum_{j=0}^{n} b_j \bar{r}_{ji} p_j. \tag{13}$$

The second term $\Delta_2$ in (10) satisfies

$$\begin{aligned}\Delta_2 &= \sum_{j=0}^{n} b_j r_{ji} \left(\frac{1}{e_j}\right) p_j[1 - (1 - e_j)^M] \\ &\le \frac{1}{e_{\min}} \sum_{j=0}^{n} b_j r_{ji} p_j [1 - (1 - e_j)^M] \\ &= \frac{1}{e_{\min}} \left[ \sum_{j=0}^{n} b_j r_{ji} p_j - \sum_{j=0}^{n} b_j r_{ji} p_j (1 - e_j)^M \right].\end{aligned}$$

The first term inside the brackets equals $P_1$, while the second term appears in inequality (11) for determining $M$. The minimal value for $M$ is achieved when the equality in (11) is satisfied. Consequently,

$$\Delta_2 \le \frac{1}{e_{\min}} [P_1 - \epsilon P]. \tag{14}$$

In order to derive a bound for the first term $\Delta_1$, we rewrite equality (11) and derive, using $e_{\min}$, an upper bound for it.

$$\epsilon P = \sum_{j=0}^{n} b_j r_{ji} (1 - e_j)^M p_j \le (1 - e_{\min})^M \sum_{j=0}^{n} b_j r_{ji} p_j.$$

Substituting $P_1$ yields

$$\epsilon P \le (1 - e_{\min})^M P_1$$

or

$$M \le \frac{\ln (\epsilon P / P_1)}{\ln (1 - e_{\min})}. \tag{15}$$

Consequently, the upper bound for $\Delta_1$ is

$$\Delta_1 = MP_0 = M(P - P_1) \le (P - P_1) \frac{\ln (\epsilon P / P_1)}{\ln (1 - e_{\min})}. \tag{16}$$

Using the two bounds stated in (14) and (16), we obtain the following upper bound for $\Delta C$:

$$\Delta C \le (P - P_1) \frac{\ln (\epsilon P / P_1)}{\ln (1 - e_{\min})} + \frac{1}{e_{\min}} (P_1 - \epsilon P). \tag{17}$$

This upper bound will serve as a weighting function for selecting the distinguishing tests. At each vertex of the decision tree, the value of the upper bound for each possible test will be evaluated. The test minimizing this bound will be selected as a distinguishing test at this vertex.

The decision tree generated by using the upper bound as a weighting function is not necessarily a minimal decision tree. However, it is nearly minimal when the contribution $\Delta C$ of each vertex to the cost function is significant. The contribution $\Delta C$ is proportional to $M$ and hence proportional to the values of the various conditional probabilities $e_i$ and to the values of the probabilities $p_i$ relative to the value of $\epsilon$. In the case where $e_i \ll 1$ and $(p_i/\epsilon) \gg 1$, the term $\Delta C$ in the objective function to be minimized (7) is more sensitive to the selected test than the sum of the next two terms $C(F_0) + C(F_1)$. Hence, the decision tree generated by using the upper bound for $\Delta C$ as a weighting function is nearly minimal in this case as shown in the following example.

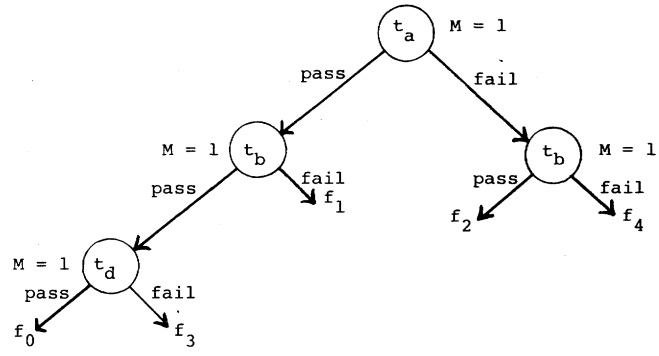


Fig. 2. A minimal decision tree for permanent faults.

*Example:* A network may have one out of five intermittent faults $f_0, f_1, f_2, f_3, f_4$ with *a priori* probabilities: $p_0 = 0.4$, $p_1 = p_2 = 0.2$, $p_3 = p_4 = 0.1$, and conditional probabilities: $e_0 = 1$, $e_1 = 0.6$, $e_2 = 0.1$, $e_3 = 0.3$, $e_4 = 0.5$. The fault matrix for this network is

$$R = \begin{matrix} \\ f_0 \\ f_1 \\ f_2 \\ f_3 \\ f_4 \end{matrix} \begin{matrix} \begin{matrix} t_a & t_b & t_c & t_d & t_e \end{matrix} \\ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \end{matrix}.$$

Using the dynamic programming method stated in Section III, a minimal decision tree was generated for $\epsilon = 10^{-5}$. This decision tree is shown in Fig. 1. The expected value of the cost function for this decision tree is $E\{C\} = 72.2$. The same decision tree was generated using the upper bound as a weighting function. However, using the latter method, the computation time was considerably smaller (by a factor of 10). The results of these two methods remain identical for other values of the conditional probabilities; e.g., $e_0 = 1$, $e_1 = e_4 = 0.9$, $e_2 = 0.1$, $e_3 = 0.3$.

Different decision trees were generated by the two methods for the following values of the conditional probabilities: $e_0 = 1$, $e_1 = e_2 = e_3 = e_4 = 0.999999$; that is, when the faults are approximately permanent. The minimal decision tree generated by the dynamic programming method is shown in Fig. 2. The value of the cost function for this decision tree is $C = 2.5$. The decision tree generated by the weighting function method is not minimal and the value of the cost function is $C = 3.1$.

It is worthwhile to note that the minimal decision tree for diagnosis of intermittent faults, shown in Fig. 1, is different from the minimal decision tree for diagnosis of permanent faults, shown in Fig. 2. The value of the cost function of the decision tree shown in Fig. 1 for permanent faults is $C = 2.6$, while the minimal value as shown before is $C_{\min} = 2.5$.

## V. Conclusions

A procedure for sequential diagnosis of intermittent faults in combinational networks has been developed in this correspondence. Using this procedure we apply each distinguishing test repeatedly a number of times so that the probability of *not* detecting an existing intermittent fault is smaller than some prespecified $\epsilon$. Lemmas 1 and 2 establish the relationship between

the maximal number $M$ of applications of each test and the value $\epsilon$.

The cost function used equals the average number of distinguishing tests required to locate a fault. We present a procedure employing dynamic programming for generating a sequential decision tree which minimizes the previously mentioned cost function. Although this method yields minimal decision trees, its limitation lies in the great amount of required computation. To overcome this limitation we next present a second procedure yielding only "reasonably minimal" decision trees, but at a considerable savings in the amount of required computation.

## Appendix

*Proof of Lemma 2*

Since for $j = 1$ we have $P_1 = 1$, the lemma is true in this case. We consider now the case $j = 2$. The first distinguishing test $t_1$ separates the set of $n + 1$ faults into two subsets, $F_1$ consisting of all faults covered by $t_1$ and $F_0$ consisting of all faults not covered by $t_1$. The maximal number $M_1$ of applications of $t_1$ is determined by the following inequality:

$$\sum_{f_i \epsilon F_1} (1 - e_i)^{M_1} \cdot p_i \leq \epsilon. \tag{A.1}$$

The subset of faults in the second vertex can be either $F_0$ or $F_1$. Consider first the case of $F_0$. In this case, each fault $f_k \in F_0$ satisfies

$$\Pr\{f_k \text{ at vertex } 2\} = \Pr\{f_k/t_1 \text{ succeeds } M_1 \text{ times}\}.$$

Applying Bayes' formula we obtain

$$\frac{\Pr\{t_1 \text{ succeeds } M_1 \text{ times}/f_k\} \cdot \Pr\{f_k \text{ at vertex } 1\}}{\sum_{i=1}^{n} \Pr\{t_1 \text{ succeeds } M_1 \text{ times}/f_i\} \cdot \Pr\{f_i \text{ at vertex } 1\}}$$

$$= \frac{p_k}{\sum_{f_i \epsilon F_0} p_i + \sum_{f_i \epsilon F_1} \Pr\{t_1 \text{ succeeds } M_1 \text{ times}/f_i\} \cdot p_i}. \tag{A.2}$$

From (A.1) the second term in the denominator satisfies

$$\sum_{f_i \epsilon F_1} \Pr\{t_1 \text{ succeeds } M_1 \text{ times}/f_i\} \cdot p_i = \sum_{f_i \epsilon F_1} (1 - e_i)^{M_1} p_i \leq \epsilon.$$

Substituting in (A.2) results

$$\lim_{\epsilon \to 0} \Pr\{f_k \text{ at vertex } 2\} = \frac{p_k}{\sum_{f_i \epsilon F_0} p_i} = \frac{p_k}{P_2}. \tag{A.3}$$

This completes the proof for $F_0$.

Consider now the case of $F_1$. If $F_1$ is the subset of faults in the second vertex, each fault $f_k \epsilon F_1$ satisfies

$$\lim_{\epsilon \to 0} \Pr\{f_k \text{ at vertex } 2\} = \lim_{\epsilon \to 0} \Pr\{f_k \text{ at vertex } 1/D\} \tag{A.4}$$

where $D$ is the event "the test $t_1$ was applied at most $M_1$ times and a failure was observed in the last application." Equation (A.4) is justified by the following argument. From (A.1) we find that $M_1 \to \infty$ when $\epsilon \to 0$. Hence, applying $t_1$ repeatedly $M_1$ times ensures that a fault $f_k$ covered by $t_1$ will cause a failure of $t_1$.

By employing Bayes' formula in (A.4), we obtain

$$\Pr\{f_k/D\} = \frac{\Pr\{D/f_k\} \Pr\{f_k \text{ at vertex } 1\}}{\sum_{f_i \epsilon F_1} \Pr\{D/f_i\} \Pr\{f_i \text{ at vertex } 1\}}.$$

Denote by $\bar{D}$ the complement of $D$; i.e., the event "$t_1$ was applied $M_1$ times and no failure was observed." Thus

$$= \frac{[1 - \Pr\{\bar{D}/f_k\}]p_k}{\sum_{f_i \epsilon F_1} [1 - \Pr\{\bar{D}/f_i\}]p_i}.$$

Substitution of Lemma 1 to this expression yields

$$= \frac{[1 - (1 - e_k)^{M_1}]p_k}{\sum_{f_i \epsilon F_1} [1 - (1 - e_i)^{M_1}]p_i} = \frac{p_k - (1 - e_k)^{M_1} p_k}{\sum_{f_i \epsilon F_1} p_i - \sum_{f_i \epsilon F_1} (1 - e_i)^{M_1} p_i}.$$

And since

$$(1 - e_k)^{M_1} p_k \leq \sum_{f_i \epsilon F_1} (1 - e_i)^{M_1} p_i \leq \epsilon$$

we obtain

$$\lim_{\epsilon \to 0} \Pr\{f_k \text{ at vertex } 2\} = \frac{p_k}{P_2}. \tag{A.5}$$

This completes the proof for $j = 2$. The extension to the case where $j > 2$ is now obvious.

## References


[1] M. A. Breuer, "Testing for intermittent faults in digital circuits," *IEEE Trans. Comput.*, vol. C-22, pp. 340–351, Mar. 1973.

[2] S. Kamal and C. V. Page, "Intermittent faults: A model and a detection procedure," *IEEE Trans. Comput.*, vol. C-23, pp. 770–725, July 1974.

[3] S. Kamal, "An approach to the diagnosis of intermittent faults," *IEEE Trans. Comput.*, vol. C-24, pp. 502–505, May 1975.

[4] H. Y. Chang, E. G. Manning, and G. Metze, *Fault Diagnosis of Digital Systems.* New York: Wiley, 1970.

[5] I. Koren and Z. Kohavi, "Sequential fault diagnosis in combinational networks," *IEEE Trans. Comput.*, vol. C-26, pp. 334–342, Apr. 1977.

[6] D. E. Knuth, *Fundamental Algorithms.* Reading, MA: Addison-Wesley, 1968.

[7] D. Matula, "A periodic optimal search," *Amer. Math. Monthly*, vol. 71, pp. 15–21, Jan. 1964.

[8] W. L. Black, "Discrete sequential search," *Informat. Contr.*, vol. 8, pp. 159–162, Apr. 1965.

[9] M. R. Garey, "Optimal binary identification procedures," *SIAM J. Appl. Math.*, vol. 23, pp. 173–186, Sept. 1972.


# An Algorithm for Grey-Level Transformations in Digitized Images

ASHOK T. AMIN

***Abstract*—Grey-level transformations on digitized images are variously used to minimize storage requirements or to enhance low-level contrast information. A graph-theoretic approach is proposed to implement these transformations. In particular, an efficient algorithm is given for transformation leading to almost uniform grey-level distribution.**

***Index Terms*—Data compression, digitized images, grey-level transformations, image enhancement, uniform distribution.**

## I. Introduction

An image $S$ is characterized by a grey-level function $g$ that assigns each point $(x,y) \in S$ a grey-level $g(x,y)$. Computer pro-

Manuscript received September 3, 1976; revised March 8, 1977.

The author was with the University of Illinois at Chicago, Chicago, IL. He is now at 227 Custer Avenue, Evanston, IL 60202.