

# ***AN AI BASED APPROACH TO AUTOMATIC FAULT DIAGNOSIS FOR MIXED DIGITAL/ANALOGUE CIRCUITS***

***T. S. Arslan<sup>1</sup>, L. Bottaci<sup>1,2</sup> and G. E. Taylor<sup>1</sup>***

## **Abstract**

Two techniques are applied to a fault diagnosis problem. The data consists of functional test reports for faulty PCBs. The first technique uses an automatically generated decision tree. The second technique seeks to mimic the problem solving behaviour of the expert fault finder and uses a knowledge-base of test reports and associated matching rules. The two techniques are described and compared with respect to performance and implementation difficulty.

## **1 The Fault Diagnosis Problem**

This report describes work done at Hull University in investigating a problem faced by the test engineering group at a local company. The group has access to a large amount of data, related to the testing of one of their locally manufactured PCBs. However, it was felt that insufficient use was being made of this data, especially with respect to fault finding. The data is in the form of reports produced by one of their ATEs. Each report contains a number of failed tests for some particular board. A single report might describe from 1 up to 50 or more failed tests. Typically, a report describes in the region of 8 failed tests but a significant number mention only a single failed test. Each faulty board is repaired and the report of failed tests, together with other test equipment, is used by the fault finder to locate the faulty component or components. It is found that about five percent of the reports indicate multiple faults and that five percent are ambiguous, i.e. two or more distinct faults described by identical failure reports.

Fault finding is a time consuming task which causes a delay in PCB production. For this reason the possibility of using a knowledge-based system to assist with fault finding was considered. The system should accept the test failure report for a given faulty board and output the most likely faulty components.

Two solutions are considered. The first solution uses an automatically generated decision tree [1] and the second uses a automatically generated fault dictionary.

## **2 The Automatically Generated Decision Tree**

### **2.1 The Algorithm**

In this method, the existing data is processed to produce a decision tree. The leaves of the tree represent individual faults, the higher nodes represent groups of faults with the root node representing all possible faults. Each node of the tree is associated with a discrimination function which is used to determine which subnode group of faults is indicated by a given failure report.

- 
- i) Department of Electronic Engineering*
  - ii) Department of Computer Science*

In the first stage, a binary tree of fault groups is constructed as follows. The set of all possible faults is divided into two groups such that one contains the most reliably identifiable faults and the other group contains all the other faults. Each of these groups is again split in two according to the same criteria. The process is repeated until at the leaves of the tree, each group consists of a single fault. The method ensures that the faults which are most easily identified are located at the top of the tree. The rationale for this is that errors in identification are more serious the higher up the tree they occur. In the second stage, discrimination or recognitions functions are added to each node of the tree. Given two groups of faults, an algorithm [2] is used to generate a function which recognizes examples of one group, but not of the other. The function, which is in the form of a logical expression of passed and failed tests is not unique, in fact there is a combinatorially large number of them. In particular, some functions will be more general with respect to the training data than others. As a result of some experiments on a small sample of data, we found that the more general functions, i.e. those which have smaller numbers of terms and recognize a relatively large number of examples, performed best. The algorithm was therefore set up to perform a moderate amount of searching for the best function of this sort.

## **2.2 Results**

The fault report data, some 250 reports, was partitioned into training data and test data. The ambiguous reports were removed since it is not possible to generate a recognition function for them. We produced a number of decision trees by varying the training data and tested each one using the test data. This section compares the different trees produced with respect to their size and construction time and also their reliability in identifying faults.

To build a decision tree using the entire training data required about 4 hours on a SUN 3/260 (M68030 based machine), and occupied about 400K bytes. The tree performed badly with the test data. The failure to construct a small reliable tree suggests the data is complex. For this reason we decided to experiment with various simplifications of the training data.

The training and test data were reduced to the eight most commonly occurring faults. Although there are many more than eight faults, these eight do account for about 25 percent of the data. This tree was much smaller at 34K bytes and required about 3 hours to build. It could identify the fault associated with a given report providing the report was relatively long (10-20 failed tests) and differed from some known report (i.e, a report in the training data) by only a few tests. The tree performed badly with shorter tests.

The board under test consists of a number of very similar modules. In the next experiment, faults were grouped by ignoring the module in which they occur, so that for example, fault3 in module2 is regarded as identical to fault3 in module7. In fact all information about the module in which a failed test occurs was removed from the data. This seems sensible since each module is subjected to the same set of tests. Once the data was grouped in this way, the ten most common fault groups were used to construct a decision tree. This tree occupied 73K bytes and required 30 minutes to build. This tree performed well with the test data, although there were some near misses.

The above experiment was repeated by taking the fourteen most common faults and the five most common faults. In general, the fewer the number of faults covered, the smaller the resulting decision tree and the greater the reliability. However, none of the trees produced appeared to be very reliable and it was felt that a different approach could lead to a better solution.

### 3 The Fault Dictionary Technique

This approach was designed to mimic the problem solving methods used by the fault finders who interpret new reports on the basis of their similarity with previous reports. Additional data was also available at this stage making a total of 400 reports.

In this technique, each test report is stored, together with its identified fault, in a knowledge-base. When a new test report is presented, the program searches the knowledge-base for a report which best matches the new test report. This method makes use of a number of heuristics for performing the match. The fault dictionary is updated as each new fault is confirmed by the fault finder.

#### 3.1 The Knowledge-Base (KB)

The knowledge-base consists of three parts (*Test Pattern KB*, *Test Significance KB* and a *Rule-Base*).

##### 3.1.1 Test Pattern KB

This is the main knowledge-base and holds the information about each fault. The knowledge base contains a list of *Fault Entries*. Each fault entry contains two parts : a *Fault Number*, which indicates a fault in a specific component, and a *Test Patterns* part, which is derived from the test reports previously encountered for this fault.

The term *Test Pattern* refers to a sequence of failed tests occurring in a single test report.

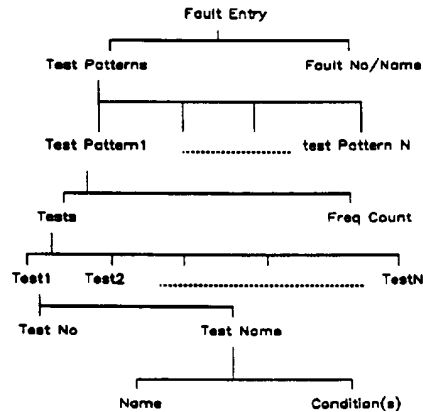


Figure 1 : The structure of a fault entry

Each single *Test Pattern* in turn consists of two parts : a *Frequency Count*, which is the number of times that a specific test pattern causes the failure of a given component, and a *Tests* part, which contains all the tests which form the test pattern.

Finally, each single *Test* has a *Test Number* and a *Test Name*. The *Test Name* consists of the name of the test and the condition(s) under which the test was applied. The structure of a fault entry is shown in figure 1.

##### 3.1.2 Test Significance KB

This knowledge base stores statistical information regarding the occurrence of each test with respect to each fault.

### 3.2 Searching The Data base (Matching)

When a failure report is presented to the program, the tests in the report are compared to each test pattern of a given fault entry. This process is repeated for all the fault entries in the Test Pattern KB. The system outputs the fault entries which produced the closest matches.

When comparing the report to each pattern, scores for the following are noted :

- 1) The number of common tests.
- 2) The number of common tests relative to the total number of tests.
- 3) The sum of the *Significances* of the common tests relative to the sum of the *significances* of all the tests.

The Significance of a single test is the relative frequency of the test for the fault in question compared to the relative frequencies of all the faults. The significance is obtained from the information in the Test significance KB.

- 4) The frequency count of a pattern.

#### 3.2.1 The Rule-Base

The choice of the "best" match is performed by comparing the scores of the different patterns. About 80 rules [3] are used to compare the rules. With each rule in the rule-base specifying which of two sets of scores is best. Each rule takes into account the relative importance of each of the four scores and this varies according to the values of the scores. The rules were obtained by inspection of the output of the matching process. This was an iterative procedure in that an initial rule-base was used to produce a set of results. The results were then used to improve the rules. This procedure was repeated a number of times.

### 3.3 Results

Currently about 65% of the reports presented to the system are correctly identified as the best match. If the best three matches are considered, then about 75% of the reports are identified correctly. There does not seem to be much room for improvements in these figures since the majority of the reports incorrectly identified are either ambiguous or not similar to any of the previously seen reports.

### 4 Conclusion

Here, we compare the two techniques by considering their advantages and disadvantages. The advantages of the decision tree technique are:

- 1) The tree is simple to build.
- 2) The identification of new reports is very fast.

The disadvantages are:

- 1) Difficulty in handling ambiguous data.
- 2) The identification is unreliable, unless, the data used are severely restricted.
- 3) The result of the identification is a single fault only.

- 4) The tree will become out of date and will require re-building periodically.

The advantages of the fault dictionary technique are:

- 1) Can handle ambiguous data.
- 2) The result of the identification produces a number of the most likely faults.
- 3) Identification is reasonably reliable over all the data.
- 4) The fault dictionary is continuously updated by new fault information and hence it is always up to date.

The disadvantages are:

- 1) More complex to build than the decision tree.
- 2) Identification is slower than the decision tree, about 2 seconds.

### 5 Future Work

currently, we are experimenting with the use of *Inexact matching* of individual test descriptions as a means of dealing with reports which are not similar to any previously seen report. We are also investigating the use of neural networks for the construction of the rule-base.

### 6 References

1. Michalski R.S., Carbonell J.G. and Mitchell T.M.: "Machine Learning: an AI Approach", Tioga Publishing Company, Palo Alto, 1983.
2. Michalski, R.S. and Chilauski, R.L.: "Knowledge Acquisition by Encoding Expert Rules versus Computer Induction from examples: A Case Study Involving Soybean Pathology", International Journal of Man-Machine Studies Vol 12 pp 63-87, 1980.
3. Waterman, D.A. and Hayes-Roth, F.: "Pattern-Directed Inference Systems", Academic Press, 1978.
4. Rumelhart D.E., McClelland J.L. and the PDP Research Group: "Parallel Distributed Processing", Vol I and II, MIT Press, 1987.