THE UNIVERSITY OF HULL

Application of Artificial Intelligence Techniques to Probeless Fault Diagnosis of Printed Circuit Boards

being a Thesis submitted for the Degree of
Doctor of Philosophy in
Electronic Engineering & Computer Science

in the University of Hull

By

Tughrul Sati Arslan, BEng(Hons) DipEng

January 1994

To Mum & Dad

In gratitude for their Care, Support, and Trust.

Acknowledgments

I would like to thank my supervisor, Professor Gaynor. E. Taylor, for her guidance , support, and sharing her valuable experience in circuit design and test which has been an inspiration to me. I would also like to express my gratitude for her guidance throughout the years of my undergraduate and postgraduate studies in Hull. It has been an honour and a valuable experience to work with Professor Taylor.

I am grateful for the supervision provided by Dr. L. Bottaci and his continuous guidance and support with this work. His efficient direction to understanding the fundamentals of LISP in the early months of this work was an important base to explore and implement a number of methods/ideas which helped me in understanding AI and building/improving the system developed here.

I would like also to thank Ericsson Telecommunications Ltd. (Scunthorpe) for their technical and financial assistance. Specifically, thanks are due to Mr. D. Clayphan, Mr. D. Shaw, and all the staff in the test engineering department.

Finally, I would like to thank all the staff and the researchers in the Department of Electronic Engineering for providing me with memorable years.

Abstract

This thesis describes investigations which led to the development of a failure diagnosis expert system for printed circuit boards which exploits functional test data. The boards considered are highly complex mixed signal (analogue and digital) systems. The data is output from automatic test equipment which is used to test every board subsequent to manufacture.

The use of a conventional machine learning technique produced only limited success due to the very large search space of failure reports. This also ruled out the use of some conventional knowledge-based approaches. In addition, there was a requirement to track changes in printed circuit board design and manufacture which also ruled out some techniques.

Our investigations lead to the development of a system which tracks changes by learning in a more restricted search space derived from the original space of reports. The system performs a diagnosis by matching a failure report with information about previously seen reports. Both exact and inexact matching were investigated. The matching rules used are heuristic. The system also uses basic circuit connectivity information in conjunction with the matching procedure to improve diagnostic performance especially in cases where matching fails to identify a unique component.

Publications

i. *An AI based approach to automatic fault diagnosis for mixed digital/analogue circuits.*
T. S . Arslan, L. Bottaci, G.E.Taylor.
IEE Colloquium, Nov 1990, London.

ii. *An Intelligent Fault Diagnosis System for Mixed analogue/digital PCBs.*
T. S . Arslan, L. Bottaci, G.E.Taylor.
Pacific Test Workshop, May 1992, Canada.

iii. *A Fault Dictionary Based Expert System for Failure Diagnosis in a Multiple PCB Environment.*
T. S . Arslan, L. Bottaci, G.E.Taylor.
Journal of Engineering Application Of Artificial Intelligence, Vol. 6, No. 5, pp. 447-456, 1993.

List of Terminologies

AI	**Artificial Intelligence**
ASIC	**Application Specific Integrated Circuit**
ATE	**Automatic Test Equipment**
ATPG	**Automatic Test Pattern Generation**
BUT	**Block Under Test**
CAT	**Connectivity Analysis Table**
CCFR	**Component-Component Frequency Record**
CPN	**Component Position Number**
CUT	**Circuit Under Test**
FC	**Final Component CPN (chosen using connectivity)**
GTR	**General Test Representation**
HDL	**Hardware Description Language**
IC	**Integrated Circuit**
InC	**Intermediate Component CPN (selected by matching)**
KB	**The knowledge-base**
KCL	**Kirchhoff's Current Law**
LB	**Lower Bound of a score range**
LIC	**Line Interface Card**
Match-pattern	**The significant sequence of failed tests in a report**
MBE	**Multiple Board Environment**
PCB	**Printed Circuit Board**
PF	**Periodic Frequency of a score range**
RB	**The rule-base**
RBP	**Range Building Period**
RL	**Rule Location in the rule-table**
RMB	**Range Manipulation Buffer**
RRL	**Reverse Rule Location in the rule-table**
S, P, G, F	**The matching scores**
SB	**Suspect Buffer**
SBFT	**Single Boolean Function Tree**
SC	**Statistical Count of a SR in statistical-ranges**
SF	**Suspect fault**
SR	**Secondary Rule**
SRRU	**Smallest Repair Replaceable Unit**

STD	**Standard Deviation value**
TBFT	**Twin Boolean Function Tree**
Test-pattern	**Any sequence of failed tests in a report**
TF	**Correct Faulty component Position Number**
UB	**Upper Bound of a score range**
UI	**User Interface**
UUT	**Unit Under Test**
VLSI	**Very Large Scale Integration**
VSA	**Version Space Algorithm**

Table of Contents

CHAPTER FOUR - THE AUTOMATICALLY GENERATED DECISION TREE METHOD

CHAPTER FIVE - THE DEVELOPMENT OF A FAULT DICTIONARY BASED SYSTEM

CHAPTER SIX - ANALYSIS OF SPECIAL CASES IN THE DATA

Table of Figures

Table of Tables

CHAPTER ONE
INTRODUCTION

1.1 The Need for Test and Diagnosis

One of the complex issues involved in designing an electronic product is that of isolating defective products immediately after fabrication. Such products are identified by *testing* all the products coming off the manufacturing line.

The testing procedure involves applying a set of input stimuli which will result in a different reponse for the faulty and fault-free circuits. In the past, these input stimuli were generated manually; however, as circuit complexities increased manual methods were replaced (fully or partially) by test pattern generation programs and the testing procedures are performed by computer-based equipment, termed automatic test equipment (ATE).

Test pattern generation programs could follow different strategies. An obvious approach is to exercise the circuit so as to check that it performs its intended function. A program developed in this way is known as ***functional test program***. For example, a functional test of a two input NOR gate could take the form of four test patterns corresponding to the four lines of the truth table that describes its operation. The alternative to the functional test program is the ***structural test program*** in which the attention is directed towards defects in circuits; these being actual physical failures that result in a functional failure.

Most of the test strategies above provide a go/no-go decision, which is the first requirement for production testing. However, there are occasions on which a go/no-go test is not sufficient; if the unit under test (UUT) is to be repaired, diagnostic information is required in addition, so as to locate the fault to the smallest replaceable element in the circuit. Even with an unrepairable unit (an integrated circuit), diagnostic information is useful as an aid to monitoring the performance of the fabrication process.

The seriousness of the problem of testing and diagnosing electronic circuits has become apparent relatively recently, largely as a result of developments in microelectronic technology. The two main effects of this technology are that component sizes have been dramatically reduced while the complexity of units at all levels (chip, PCB, system) has been dramatically increased. Both of these effects can be reflected in the increase in the number of transistors commercially integrated on a single chip, which has risen from 2 to more than 500,000 in just over two decades [PUC92]. The increased complexity, mentioned above, makes an intuitive approach to testing ineffective; while the reduction of component size, reflected in the amount of circuitry that can be accommodated on a single chip, means that testing based on the use of measuring instruments to trace signal flow through the circuits is very restricted on PCB circuitry [WIL86].

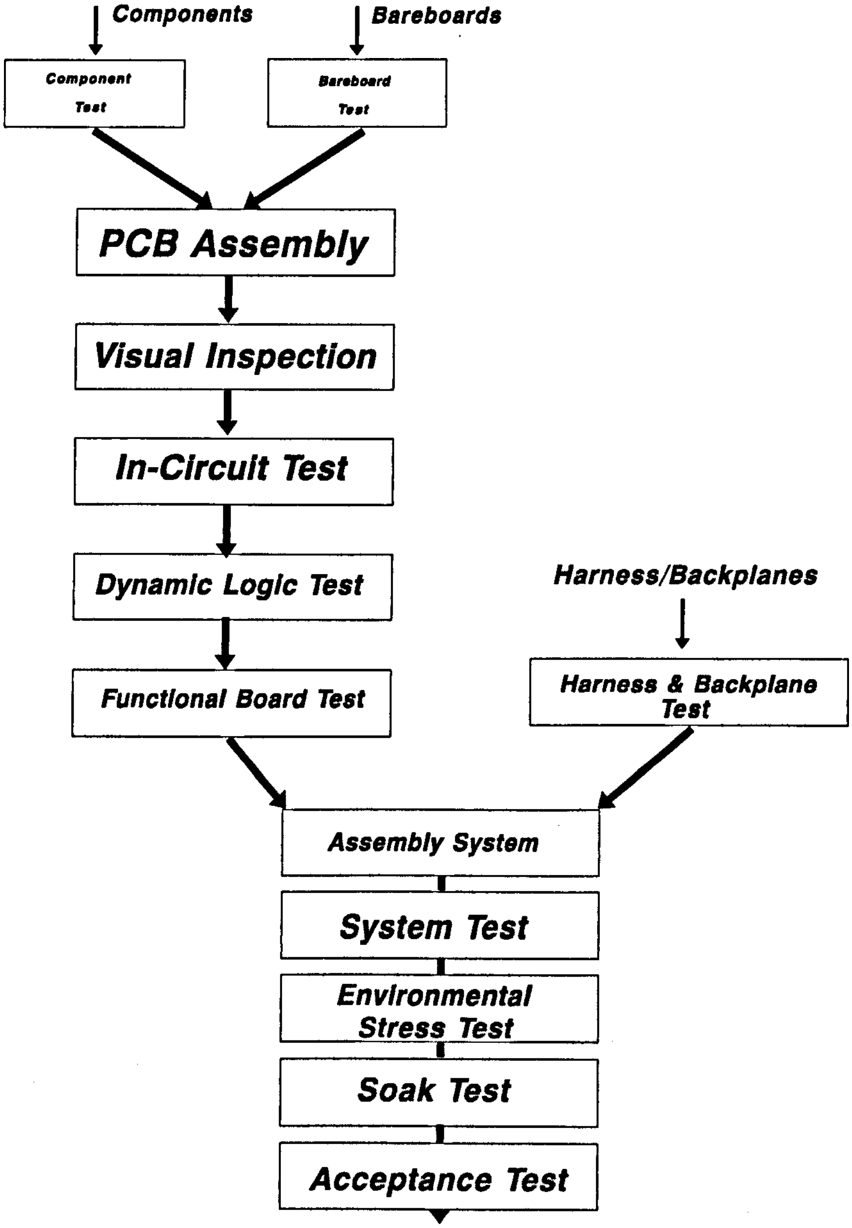


Figure 1-1: Computer integrated testing

Figure 1-1 is a general illustration of the test stages involved in producing a PCB product [MAN92]. The following are brief descriptions of each stage :

Component Test Random samples of each component batch are tested to check compliance with required specifications.

Bare board Test The Bare boards (unpopulated PCBs) are tested for physical defects such as cracks.

PCB Assembly Components are assembled and soldered on bare boards.

Visual Inspection Each PCB is tested for any visually detectable faults such as misplacement of component.

In-circuit Test The PCB is mounted on a fixture, which is part of the in-circuit tester. The fixture establishes contact with specific points internal to the PCB. The in-circuit test aims to identify inconsistencies in the values of passive components (resistors, capacitors, and inductors) from their nominal values, which are programmed into the tester at an earlier stage.

Dynamic Logic Test This is part of the in-circuit test and involves testing dynamic logic parts in the circuit.

Functional Test Each PCB is tested by an ATE which is driven by a functional test progrm (see above). The PCBs are tested by applying input stimulus and monitoring their response via the edge connector pins (input/output pins and pins connected to specific test points on the PCB).

Harness Backplanes Test If the PCB is part of a given system, then the external wires and connectors connecting the individual PCBs are checked and tested.

Assembly System A number of PCBs are interconnected to form a more complex function.

Environmental Stress Test Mainly involves testing under different temperature conditions to check for defected components.

Soak Test ensures that the product can operate correctly for a length of time defined in the specifications.

Acceptance Test This includes a number of tests, set by the customer, which ensure that the products function adequately.

1.2 A Case of Test and Diagnosis

This thesis investigates a problem, faced by the test engineering group at Ericsson Telecommunications Ltd., related to the testing of a number of their locally manufactured PCBs. The test strategy followed in the company conforms to that described in the previous section (figure 1-1). Since the problem is encountered at the functional test level, the rest of this thesis will concentrate on this level of testing at the firm.

The functional test ATEs are LIC (Line Interface Card) testers since they communicate with the tested PCB via an interface card which is connected to the edge connecters on the PCB (i.e. there is no direct connection to internal points). Each ATE functionally tests the attached PCB by applying both analogue and digital stimuli to the edge connector pins. Due to the complex functionality of most of the PCBs examined by such equipment, a number of tests may be required each of which may involve the application of different types of stimulus and the examination of different pins of the edge connectors.

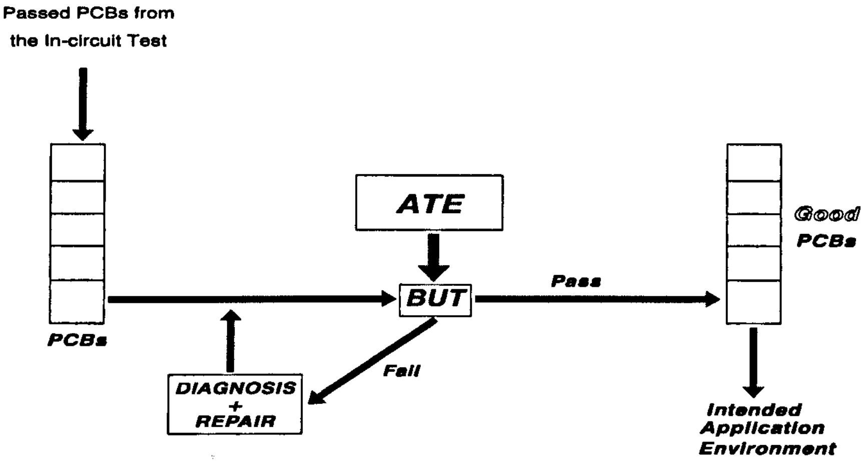

Passed PCBs from the In-circuit Test
ATE
BUT
Pass
Good PCBs
PCBs
Fail
DIAGNOSIS + REPAIR
Intended Application Environment

Figure 1-2: The functional test procedure.

A PCB can *pass* the tests of the ATE, in which case it is considered to be ready for its intended purpose. Otherwise, a PCB *fail*s one or more of the tests, in which case it is considered as faulty and taken off-line. A failure report is produced, for each faulty PCB, which lists information regarding each failed test. The report is intended to guide the diagnostic technicians in locating the fault. The technicians rely mainly on their experience with previous failure cases in locating the fault, i.e. they recognize patterns of failed tests which correspond to particular faults. However, in some cases the need arises for probing the PCB circuit using some electronic measurement equipment such as oscilloscopes, voltmeters, . . . etc. During the probing procedure voltage measurements and waveforms at certain points on the faulty PCB are compared with the corresponding points on a known good PCB. Figure 1-2 illustrates the basic testing procedure followed at this stage.

Since fault diagnosis of this type is a time consuming task it causes a delay in PCB production. The test group at the company has access to relatively large amount of data in the form of failure reports related to the functional testing procedure and it was thought that these reports (and those obtained in the future) could be utilized in automating the diagnosis process and avoiding delays caused by probing. As access for both test and circuit information is limited since the PCB design and generating the test programs are tasks which are performed in different sites in different countries, the possibility of using a system which is based on an artificial intelligence (AI) approach was considered. The system should accept the test failure report for a given faulty board and output the most likely faulty component(s).

Another reason for the need for such a system is to store information which otherwise is lost with the departure of the technical staff from the company.

1.3 Preliminary Introduction to the Thesis

This thesis details the research work carried out in investigating the case described in the previous section. In addition, the thesis describes a number of novel aspects produced as a result of the investigations above.

Data was obtained from the company in a number of stages. Initially, the data concerned one of their most common PCB styles. For this reason a significant proportion of this research work was carried out with these data. The use of other boards was investigated later through this work.

Chapter two is a literature survey into research work carried out in the diagnosis of failures in mixed analogue\digital circuits. The chapter describes the use of techniques which are based on the simulation of circuit behaviour through the use of mathematical circuit theory equations. In addition, the chapter reviews the use of fault dictionaries in diagnosing both digital and analogue circuits. Finally, the chapter considers the work done in applying artificial intelligence (AI) techniques to fault diagnosis of such circuits. The review revealed that most of the work use expert systems in circuit diagnosis. The chapter also considers machine learning techniques due to their possible application for classifying the examples in the functional test data (see section 1.2).

Chapter three describes further aspects of the diagnosis case considered in this thesis by analysing the data and looking at the design of the PCB concerned and its complexity.

The review of chapter two revealed the possibility of using machine learning techniques in classifying the data. Chapter four considers the use of one such technique. The chapter details the necessary modifications to the data in order to apply the technique. Other details such as the results obtained and the limitations of the technique are also described.

Due to the inflexibility of machine learning techniques (caused by their algorithmic nature), the use of a knowledge-based technique was considered. Such techniques could incorporate heuristics used by the diagnostic technicians in identifying failure reports. Chapter five describes the development of a knowledge-based system which is modelled to mimic the behaviour of a diagnostic technician during fault diagnosis. The chapter considers each of the elements of the system describing the aspect which it represents in modelling the expert's behaviour.

The system developed in chapter five could not deal with the ambiguous cases in the data. For this reason, a thorough analysis of the data is carried out and is described in chapter six. The analysis aims to explain the ambiguous cases in the data in terms of the behaviour of the circuit. Further analysis is done on cases in the data which led to the use of heuristics by the technicians. The aim was to generalize these heuristics or find alternative general ways of dealing with the above cases.

The work described in chapter six suggested the need for using connectivity information in improving individual cases in the data. Hence, chapter seven describes how circuit connectivity information is used to enhance the performance of the system by recovering cases which cannot be isolated using matching alone. A number of heuristic techniques, developed as part of this work (which are based on manipulating connectivity information), are described in this chapter.

The system described in chapter five operates with a set of manually developed rules. Such rules are specific and could not cope with slight changes which could be caused by changes in the system environment (e.g., a change in the format of the failure reports). Hence, chapter eight describes the development of two heuristic techniques which enable the system to develop and adjust its internal rules. In chapter nine, these techniques are compared with one of the standard techniques used in concept learning.

The work described in chapter ten aims to investigate additional ways of improving the performance of the system. The investigations in this chapter consider additional details in the failure reports which were not considered earlier in the system described in chapter five.

The work described in chapter eleven aims to investigate the ability of the system in coping with changes in the design of the boards and to examine its performance with changes in the information content of failure reports, which could be caused by different ATEs. For this reason two additional boards of varying complexity are used. The results obtained are compared and analysed in terms of their circuit design.

Chapter twelve summarises the work performed in this thesis highlighting original aspects. A number of suggestions are provided to be pursued in the future.

CHAPTER TWO
A REVIEW OF LITERATURE OF FAILURE DIAGNOSIS TECHNIQUES FOR MIXED ANALOGUE\DIGITAL CIRCUITS

2.1 Introduction

Fault diagnosis is required so that a defective element in a circuit can be identified with a view to repair, or to provide feedback data for improving the yield of the manufacturing process. The diagnosis of a fault in an electronic product follows from testing it and hence depends on the test strategy employed (see section 1.1). In the case of structural testing, diagnosis aims to determine a fault[1] caused by a particular defect in the circuit. For example, determining whether a particular connection is either stuck-at-1 or stuck-at-0 [MAN92] for a digital circuit or, more generally, whether a transistor is stuck open or there is a short. In the case of functional testing, diagnosis aims to identify a functional failure in a component (or a group of components) [WIL86]. As mentioned in the previous chapter, this thesis considers the use of functional test data in the failure diagnosis process.

The main approaches employed in the area of fault diagnosis during the last decade are either based on mathematical circuit theory, based on fault dictionaries, or involve the use of an artificial intelligence technique. This chapter reviews research work in the area of fault diagnosis of electronic circuits, emphasising the above techniques. Based on this review, the chapter will suggest techniques which should be applied to failure diagnosis based on the utilization of functional test data (such as the case mentioned in section 1.2).

2.2 Mathematical Techniques

This section describes a number of papers which perform diagnosis through the simulation of circuit behaviour by using mathematical equations which are based on the circuits to be diagnosed. However, the complexity of such equations grows with the increase in circuit size and complexity. For this reason, such techniques have only been applied to circuits with a limited number of nodes and their use has only extended up to the early part of the last decade.

Biernacki [BIE81] deals with multiple-fault detection for linear analogue circuits. The method proposed is based on measurements of voltage using current excitations and has been developed for the location of a number of faults. It utilizes certain algebraic invariants of

[1] Here, a fault is the electrical effect of a defect in the circuit.

faulty elements. Computationally, it depends on checking the consistency or inconsistency of suitable sets of linear equations. The equations themselves are formulated via adjoint circuit simulation.

In [WU82] a simulation-after-test algorithm for the analogue fault diagnosis problem is proposed in which a bound on the maximum number of simultaneous failures is used to minimize the number of test points required. The resultant algorithm is applicable to both linear and nonlinear systems with multiple hard or soft faults and can be used to isolate failures up to an arbitrarily "replaceable chip or subsystem". In the algorithm, the components (individual chips, discrete components, or subsystems) are divided into two groups at each step of the test algorithm. It is assumed at each step that one group is composed of good components and the known characteristics of these components are used together with the test data to determine whether or not the remaining components are good. The process is repeated at the next step of the test algorithm with a different subdivision of components. The number of components which may be tested at any one step is dependent on the number of test points available. The test is formulated by assuming a *component connection model* for the circuit or system under test. For the linear case the component equation is modelled in the frequency domain. The system was implemented on the VAX 11/780. The work described in [RAP83] develops the multifrequency fault diagnosis problem in the context of a tableau based on the use of the component connection model, which was also used in [WU82]. This method avoids the computation of the composite system transfer function matrix. The resulting fault diagnosis equations have a regular structure with fixed polynomial order (often quadratic) which is exploited in the solution process. The development includes a test for diagnosability as well as a set of bounds on the number of test frequencies necessary to conduct a diagnosis. An iterative scheme to solve the fault diagnosis equations is proposed and is shown to be locally convergent. The component connection model and the test algorithm used in [WU82] were also applied on the implementation of an analogue ATPG (Automatic Test Program Generation) for both a linear and a nonlinear case in [WEY85] and [WEY84] respectively.

2.3 Fault Dictionaries

A *fault dictionary* in its simplest form is a data-base in which a number of faults are associated with information which assist in their diagnosis. The dictionary is used as a look-up-table during fault diagnosis. The information stored in the dictionary varies depending on the type of the circuit (i.e., analogue or digital), as will be revealed in the next two sections. Fault dictionaries have been used widely in the diagnosis of digital circuits, however their use for analogue and mixed circuits has been limited. The next sections are a review of work in this area.

2.3.1 Diagnosing Digital Circuits

The fault dictionary technique is one of the early techniques used in fault diagnosis of digital circuits. A common method of diagnosis using a fault dictionary [BEN82, WIL86, BUC89] consists of two stages (*pre-test* and *post-test*).

In the pre-test (off line) stage the dictionary is built by simulating the effect of some pre-selected test patterns (chosen by the test engineer). For each test pattern the fault-free response and the *fault cover* (faults detected by a particular pattern) are stored. In the post-test (on line) stage the circuit is supplied with the same test patterns. A decision tree is generated [BEN82, WIL86], on-line or a priori, which is used in locating the fault. [BUC89] gives some technical detail about the use of such a dictionary in an industrial testing environment.

[RAT86] describes a fault dictionary which is mainly used to assist fault location with a guided probe technique [TUR90]. The fault dictionary is created based on the results of fault simulation, which models the effects of various fault types on circuit outputs. A fault dictionary generation program uses the simulator's predictions about the faulty circuit's primary output states for each test pattern applied. It uses this information to compile fault signatures -lists of primary output/pattern combinations which uniquely describe the simulated faults. When a real circuit fails during test, a lookup program compares the failed outputs with each fault signature; *close matches indicate the likeliest fault source*. The guided probe and fault dictionary diagnostics are integrated with the system such that the fault dictionary is referenced by the probe algorithm each time the next probing point is selected.

[TAN88] presents a system for generating system level fault dictionaries. It utilises a hardware structure for checking the soundness of logic circuits and finding the fault if it exists. This structure, which has the ability to trace from different points in the circuit, is used by the fault dictionary generating system in generating fault dictionaries. The dictionary consists of a list of replaceable units for each logic module ordered according to failure probability. It also contains the name and assembly location of the unit.

The fault dictionaries used in all of the work above are generated prior to the test of the circuit. The work in [KAT89] describes a fault diagnosis approach which is based on post-test fault dictionary generation. The fault dictionary is used in conjunction with an electronic beam tester in a two stage fault diagnosis process for VLSI circuits. It is used only to store faults/symptoms which cannot be detected by thorough examination of go/nogo test symptoms (and hence a smaller dictionary is produced). Primitive gate level circuit and stuck-at type fault models are required.

2.3.2 Diagnosing Analogue Circuits

Research work in the area of the diagnosis of analogue circuits has been restricted in comparison with the work done in diagnosing digital circuits. This is due to the difficulty in testing/diagnosing such circuits, which is caused by the various ways in which an analogue circuit can fail [QUT88]. In addition to faults caused by short and open circuits [QUT88,QUT289], many others could occur due to variation in the values of analogue component parameters (e.g., values of resistors and gain of transistors) form their nominal values chosen at the design stage [MCK89]. The main difficulty in the diagnosis is caused by variations in component parameters.

A common approach to testing analogue circuits is to compile a fault dictionary [MAS89] which is generated in a manner which resembles that used in the digital approach [BEN82] and [WIL86], however, the information stored in the dictionary are mainly based on simulated voltage measurements (as will be revealed below). Most of the work performed in this area follows a similar approach to the *fault location* procedure. This procedure was first adapted in [HOC79] and was later modified by others.

As in the digital approach, the diagnosis consists of pre-test and post-test stages. In the former a low-level circuit simulator (SYSCAPII [HOC79], SPICE [MAS89, LIN85]) is used to simulate the circuit to be tested, both under fault-free DC conditions and with a number of singly-chosen faults. The fault dictionary is simply a table of single faults, input vectors (chosen by the test engineer so as to make faults distinguishable at the output), and their corresponding output faults. The post-test stage involves applying input simuli to the circuit under test (similar to that in simulation). The method utilises comparison on a least square basis in locating the faults. Fault isolation is achieved by grouping faults, with outputs which are not sufficiently different to permit fault isolation, into *ambiguity sets* and manipulating these sets using various algorithms.

Work described in [LIN85, QUT90, VIS84] involve modifying the above approach to increase its efficiency. [LIN85] modifies the fault detection procedure such that a simple dictionary 'lookup' operation is enough to identify the fault. This is done by using a few heuristics for reducing the number of the test nodes in the ambiguity sets. The paper describes other improvements which involve both the pre-test and post test stages which aim to make the technique suitable for larger circuits. Another approach for reducing the number of test nodes is described in [QUT90]. It involves calculating a ***sensitivity factor***, which is a measure of how sensitive a given node is to faults, for each node in the CUT. The nodes selected are those with highest sensitivity. If the Fault Coverage obtained is not acceptable then the nodes with the next highest are picked.

An impractical assumption made in [HOC79] is that good components have their exact nominal behaviour. [VIS84] presents a robust approach which allows tolerance variations in the good components.

Although all the work described is based on the development of the same approach, [HOC79] states "Faults selected by the engineer can encompass opens, shorts, low gain device conditions, leakage resistances, and other dc type failure modes". But the rest indicate the suitability of their approaches to hard faults only. [HOC79] considers some historical information in building the dictionary by considering a number of statistical reports, obtained from different industrial sources in the USA. However, the use of the data is system independent.

The work in [PRA90] presents a fast algorithm for generating fault dictionaries for linear analog circuits. It is based on the adjoint network concept. However, due to the computational complexity of the algorithm it is not suitable for VLSI and other large analogue circuits.

Most of the work carried out in the area of fault diagnosis of analogue circuitry using fault dictionaries concentrates on testing using DC signals. Attempts at generating AC fault dictionaries have been limited to small circuits [PAH82, SCH79] and diagnosis efficiency decreases as circuits increase in size. As in the case of [PRA90], this is due to the use of mathematical equations in generating the fault dictionaries. The use of DC diagnosis for analogue circuits is common because of the increase in manufacturing mixed Analogue/Digital circuits and the use of DC in testing digital circuits. Hence the same tester could be used in testing both parts instead of two different ATEs (or a Multi-signal ATE, which is more expensive).

2.4 Artificial Intelligence Techniques

The use of AI techniques in the failure diagnosis of electronic circuits is potentially advantageous for two main reasons. These are :

1. The advance in the design and production of circuits has lead to an increase in their size and complexity. This increase has made it impractical to tackle such circuits with exahstive algorithmic diagnosis methods. However, such circuits could be tackled by using heuristics. The field of AI is the source of many techniques for exploiting heuristic approaches to problem solving.
2. With experience, humans can diagnose complex circuits that are intractable using simulation techniques. AI techniques attempt to capture this human diagnostic behaviour.

The rest of this section will consider those fields of AI which could be applied to the diagnosis problem which is the subject of this thesis (see section 1.2). The section will also consider the existing research work in these fields.

2.4.1 Theoretical Background

2.4.1.1 Expert Systems

An area of AI, relevant to the problem addressed in this thesis, is expert systems. Expert systems are said to be *"advanced"* knowledge-based systems that can solve difficult problems [AMB87]. Expert systems attempt to emulate the methods used by a human who is an expert in a particular field. Up-to-date knowledge about the field is usually obtained from experts in that field and are stored in the part of the system known as the knowledge-base.

The majority of such systems in the area of fault diagnosis fall into two main classes based on the way knowledge is utilized. These are ***Rule-Based Systems*** and ***Model-Based Systems***. This section will briefly introduce the two system types and will examine knowledge representation and manipulation.

2.4.1.1.1 Rule-Based Systems

The diagnosis approach in rule-based systems is empirical, based on the supplied relationships between faults and symptoms. The most popular approach to representing the domain knowledge (both facts and heuristics) for such an "expert" system is by production rules, which have the format: **"IF *<premise>* THEN *<conclusion>*"**[2]. Rules used to deduce faults from sysmptoms have the form :

$$\text{IF symptom}_1 \text{ AND symptom}_2 \text{ AND} \ldots \text{ THEN fault}_1$$

[2] Production rules can be used to represent knowledge in a model-based system.

Where $symptom_1$ and $symptom_2$ are some symptoms of a given fault, $fault_1$. In some cases, conclusions do not point directly to faults but to some intermediate descriptions of faults [KRI86], e.g. a functional block within which the faulty component lies, for example :

IF $symptom_1$ AND $symptom_2$ AND . . . THEN fault within $module_1$

Conclusions could also point to a possible repair action [HAV89], e.g.,

IF $symptom_1$ AND $symptom_2$ AND . . . THEN replace diode

The reasoning underlying rule-based systems is known as shallow reasoning. This is because shallow reasoning resembles the method technicians (rather than electronic engineers) use in relating failure symptoms to the actual cause of the fault. Technicians do not use comprehensive electronic or circuit knowledge (such as test engineers) in their diagnosis, but follow rules (mainly based on experience) which associate symptoms with faults.

When developing such a system as many circuit failures as possible have to be represented beforehand. And new rules have to be added to the system as unforeseen faults occur. The process is likened to building up a fault dictionary [EVA91].

The main source of the diagnostic rules is the diagnostic technician and the circuit designer. For this reason, a number of technicians and designers are usually interviewed (by the system designer) in order to construct the rules for the system [JAC90]. In some cases, a limited amount of simulation may be required in order to identify the circuit behaviour under certain conditions [KEN91].

2.4.1.1.2 Model-Based Systems

Diagnosis, in a model based system is theoretical and based on the structural design of the device. The system tracks normal behaviour and infers faults from violations in this behaviour.

The reasoning mechanism that model based systems utilize is known as deep reasoning. This is the sort of reasoning an engineer might perform. In general, a circuit is represented using a frame oriented representation, with individual components being instances of generic object types [RIC91]. The connections between the objects provide the network links across which circuit information is traced during the diagnostic process. Simplistically, a fault is identified based on discrepancies between observed and predicted values at the input or output of a device component. The reasoning process is facilitated by the use of a hierarchy, first tracing faults at the highest possible level and subsequently isolating components within the faulty sub-structure.

Model based systems exist which are based on structural information of electronic devices at logical gate level [DAVE84], basic transistor level [DEK87], and more complex functional level [TAY90]. Section 2.4.2.1.2 will describe the main structural levels used in building model-based systems for failure diagnosis of electronic circuits.

2.4.1.1.3 Hybrid Systems

Rule-based diagnosis will efficiently diagnose familiar symptoms, but the required knowledge acquisition is time consuming. Model-based diagnostics simplify knowledge collection by deriving diagnosis from the design data incorporated into the model. The model can diagnose unfamiliar symptoms, but only considers faults it can model. By combining the two approaches, the limitations of the two, mentioned above, can be overcome since, according to [HAV89, PFL89], the two methods complement each other and hence a powerful system is obtained.

2.4.1.2 Machine Learning

A significant limitation of the AI techniques considered so far is the need to acquire large amounts of detailed knowledge from experts or by simulation. Machine learning is an alternative to the manual acquisition of this knowledge.

An important part of the work done in machine learning is the learning of general relations from sets of examples (induction). Learning from examples involve classifying objects without being given explicit rules. A major part of the work in this area has been carried out by [WIN75], [MIT77], [MIT78], [QUI86], and [KOD85].

The work in [WIN75] generalizes a number of examples of configurations of blocks to "concepts" of arch, tent, and house. The basic approach followed starts with a structural description of one known instance of the concept and call that description the concept definition. By examination of descriptions of other known instances of the concept, the definition is generalized to include them. The definition is restricted to exclude objects which are not instances of the concept in question but are very similar. Such objects are termed **near misses**.

Both [MIT77] and [MIT78] describe another approach to concept learning called **version spaces**. The goal is the same as [WIN75]: to produce a description that is consistent with all positive examples, but no negative examples in the training set. But while [WIN75] did this by evolving a single concept description, version spaces work by maintaining a set of possible descriptions and evolving that set as new examples and near misses are presented.

As training examples are processed, the notion of where the target concept might lie is refined to form the current hypothesis of the system, which is represented as a subset of the concept space called the version space. The version space is the largest collection of descriptions that is consistent with all the training examples seen so far. It consists of two subsets. One subset, called G, contains the most general descriptions consistent with the training examples seen so far; the other subset, called S, contains the most specific descriptions consistent with the training examples. The version space is the set of all descriptions that lie between some element of G and some element of S.

The algorithm for narrowing the version space is called the ***candidate elimination algorithm***. In general, each time a positive training example (inside the concept space) is received, the S set is adjusted as little as possible to be more general so that it covers the new training example. Negative training examples (outside the concept space) serve to make the G set more specific. With each negative example, G is adjusted to include the most general set of descriptions in the version space that do not cover the example. If the S and G sets converge, the range of hypotheses will narrow to a single description. The algorithm is described in greater detail in [MIT77 and MIT78].

Another approach for learning from examples is the induction of *decision trees* of discrimination function in terms of the attributes of the objects [LUG93]. An example is the algorithm described in [QUI86], which uses an iterative method to build up decision trees. The program starts by choosing a random subset of the training examples. The algorithm builds a decision tree that classifies all the objects in the subset. The tree is tested on the training examples outside the chosen subset. If all the examples are identified correctly, the algorithm halts. Otherwise, it adds a number of training examples to the subset and the process repeats. Each node in the tree is constructed by choosing a single attribute (from the attributes of the examples in the data) and measuring its information yield. This is essentially, the usefulness of that attribute for discriminating between members of the subset. The concept used in measuring the information yield was developed by Shannon in [SHA48]. Shannon defines the amount of information in a given attribute as a function of the probability of occurrence of each possible attribute. The chosen attribute is the one with the largest yield.

As mentioned above, the discrimination made by [QUI86] was based on choosing a single attribute from all the attributes of the objects in the training set. For examples, choosing the attribute colour only from the set {colour, height, width}. The approach developed in [MIC81], is based on inducing decision rules based on variable-valued logic developed by Michalski in [MIC75]. Variable-valued logic is used for representing decision problems involving many-valued variables (variables that can take on some range of values). This approach is used in [MIC81] for developing rules for diagnosing soyabean diseases. This approach can provide a more accurate generalization of rules, since it can accommodate for different attributes of objects in the training set and allow these attributes to take a range of values. Hence, a possible boolean **recognition function** for the same example above would be:

$$(\text{colour} = \text{blue}) \text{ or } ((\text{colour} = \text{red}) \text{ and } (\text{width} > 3))$$

Due to the flexibility with which decision rules could be represented by Michalski's approach relative to the rest of the approches encountered so far, the rest of this section will be dedicated to the theoretical aspects involved in this approach.

The approach followed by [MIC81], operates on groups of example objects by making generalizations by similarity detection. As a result, an expression is produced for each group which describes the examples in that group only.

Recognizing groups of elements from a large group by considering different attributes of these elements is performed using recognition functions, which are boolean expressions in terms of the different attributes of the group which define an element or a group of elements within the large group. Developing a recognition function for two different groups is based upon an algorithm [MIC81]. The algorithm is described below. The description is followed by a worked example. In the description a number of definitions are used.

Definitions

The following terms will be used throughout these definitions :

If we consider a group (set) of elements $E = \{e_1, e_2, \ldots.., e_n\}$, where each element can be specified by different values of the attributes $X = \{X_1, X_2, \ldots\ldots, X_l\}$, where n and l are integers, then

1) $G(e_i / e_j)$

This represents the most general recognition function which recognizes the element e_i and rejects the element e_j. This function is obtained by comparing the values of the attributes in e_i and e_j. If the variable X_i has the same value in e_i and e_j, then X_i does not occur in $G(e_i / e_j)$.

This function will be in the form of *disjunctions (logical sum)* :

$X_i \vee X_j \vee \ldots\ldots \vee X_k where X_i, X_j, X_k \in X$

2) *$G(e_i / \{e_j\})$*

This represents the most general recognition function which recognizes the element e_i and rejects the set of elements $\{e_j\}$.

For each e_k in $\{e_j\}$, G(e_i/e_k) is found. The conjunction of all the **G**s obtained is the most general recognition function which recognizes the element e_i and rejects the set of elements $\{ej\}$.

This function will be in the form of the *conjunction of disjunctions (product of sums)*:

$(X_i \vee X_j \vee \ldots\ldots \vee X_k) \wedge$

$(X_i \vee X_k \vee \ldots.) \wedge$

.

.

.

The above function can be multiplied out to form disjunctions of conjuctions (sum of products):

$(X_i \wedge X_j \wedge \ldots\ldots \wedge X_k) \vee$

$(X_j \wedge \ldots\ldots\ldots.) \vee$

.

.

.

Each term in this disjunction is a recognition function. The question thus arises as to which of these is the most suitable or best function. In choosing among alternative recognition functions, Michalski [MIC81,KOD85] employs the notion of ***sparseness***.

Sparseness

sparseness describes the generality of a function which is describing a given set of elements, and is defined as the difference between the total number of different elements the recognition function can potentially recognize and the different number of elements which have been actually observed (i.e, the elements which we actually have in the set).

3) $G_{opt}(e_i / \{e_j\})$

This represents the "best" recognition function which recognizes the element e_i and rejects the set of elements $\{e_j\}$ (see 2).

According to Michalski, the best term is the one with the *minimum* sparseness. The choice of the best term here resembles that made in [QUI86], where the measure of the information yield of each attribute is considered in the choice of the "best" attribute for making a decision at a given node of the tree. Michalski indicates that the reason why the term with minimum sparseness is chosen to represent $G_{opt}(e_i/\{e_j\})$ is that since sparseness estimates a *degree of generalization* in the recognition function, then the less the sparseness the more specific is the information described. Whereas, using maximum sparseness a more general is the describtion. The concept of sparseness is illustrated in figure 2-1, where + represents the positive examples in the data space and - represents the negative examples.

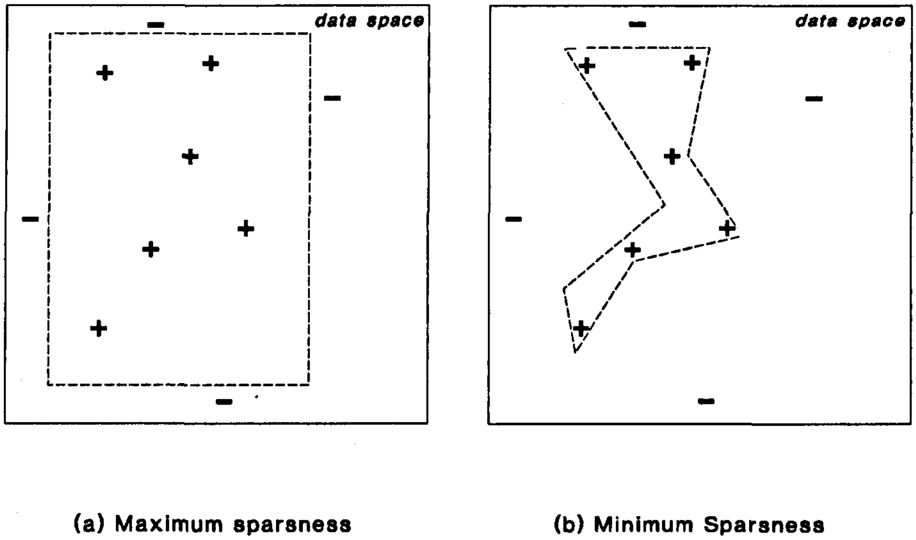


Figure 2-1: Illustration of maximum and minimum sparseness.

The following example illustrates the use of the above concepts in identifying a recognition function which recognizes the elements of group$_1$ rejecting the elements of group$_2$. Table 2-1 includes the details of the elements of each group.

Group	*Example*	T_0	T_1	T_2	T_3
1	e_1	*1*	-	-	-
	e_2	-	*1*	-	*1*
2	e_3	-	-	*1*	-
	e_4	-	-	-	*1*

Table 2-1: Details of the elements for the example

The examples of the table have the test domain $[T_0 .. T_3]$, any of which could fail. Failure of any of the tests for a particular example is indicated with a 1 in the corresponding test slot.

$$G(e_1/e_3) = T_0 + \overline{T}_2$$

$$G(e_1/e_4) = T_0 + \overline{T}_3$$

The recognition function which recognizes e_1 but rejects the whole of group$_2$, $G(e_1/\{e_3,e_4\})$ is obtained by logically ANDing the above two expressions to obtain the following;

$$G(e_1/\{e_3,e_4\}) = T_0 + T_0\overline{T}_3 + T_0\overline{T}_2 + \overline{T}_2\overline{T}_3$$

To find the sparseness of each term we use the definition mentioned previously. Considering the first term, T_0, the total number of different elements it can potentially recognize is (1.2.2.2 = 8), while the number of elements it can actually recognize in group$_1$ is 1 (e_1 only) and hence the sparseness for this term is 7. The procedure is used to calculate the sparseness for each term in table 2-2 below.

$G(e_1/\{e_3,e_4\})$	Sparseness
T_0	7
$T_0\overline{T}_3$	3
$T_0\overline{T}_2$	3
$\overline{T}_2\overline{T}_3$	3

Table 2-2: Sparseness values for each term in $G(e_1/\{e_3,e_4\})$.

As mentioned above, $G_{opt}(e_1/\{e_3,e_4\})$ is the term with the minimum sparseness. This means all the terms with the sparseness of 3 are possible candidates for $G_{opt}(e_1/\{e_3,e_4\})$.

Now, a recognition function, R, can be formed. At this stage $R = G_{opt}(e_1/\{e_3,e_4\})$. Testing the recognition of R for the elements of $group_1$, only e_1 can be recognized (This is the case with all the candidate terms of G_{opt}, see above). Since the recognition is not complete we repeat the same procedure above for finding G_{opt} using e_2, to be added to (logically ORed with) R. In a manner similar to that above, it can be shown that:

$$G(e_2/\{e_3,e_4\}) \qquad = T_1 + T_1\overline{T}_3 + \overline{T}_3 + T_1\overline{T}_2 + \overline{T}_2T_3 + T_2\overline{T}_3 + T_1T_3$$

and that the respective sparseness values are as in table 2-3 below:

$G(e_1/\{e_3,e_4\})$	Sparseness
T_1	8
$T_1\overline{T}_3$	4
$\overline{T}_3$	8
$T_1\overline{T}_2$	4
$\overline{T}_2T_3$	3
$T_2\overline{T}_3$	3
T_1T_3	3

Table 2-3: Sparseness values for each term in $G(e_2/\{e_3,e_4\})$.

As is clear from the sparseness values of table 2-3, any of the terms $\overline{T}_2T_3$, $T_2\overline{T}_3$, and T_1T_3 are possible candidates for $G_{opt}(e_2/\{e_3,e_4\})$, and hence any of them can be added to R and hence;

$$R = T_0\overline{T}_3 + \overline{T}_2T_3$$

or $$R = T_0\overline{T}_2 + T_2\overline{T}_3$$

.

.

.

Any of the expressions of R can recognize both elements of $group_1$ and none of $group_2$, and hence constructing the recognition function, R, is complete. In cases where recognition is required among a number of groups, such as the fault groups in the data (see later), a recognition function such as R must constructed for each group against the rest, and hence the procedure is repeated a number of times and more complicated expressions are produced due to the relatively larger number of tests. The practical implementation of the method using the actual data will be the subject of chapter four.

2.4.2 Existing Reaserch Work in Application of AI to the Diagnosis of Electronic circuits

Much of the published work, in the field of applying AI to fault diagnosis of electronic circuits, have concentrated to the development of expert systems. The next section will consider the work done for each class of expert system, i.e., rule-based, model-based, and hybrid (see section 2.4.1.1).

2.4.2.1 Expert Systems in the Literature

2.4.2.1.1 Existing Rule-Based Systems

Papers [APF85,WIL85,WAW89] describe Expert Systems used in fault diagnosis of electronic circuitry. The system described in [APF85] replaces a skilled technician in analyzing an in-circuit test failure data; deciding if additional measurements are required; and then, using a rule-based system, creating a simple message. A knowledge database exists for each board which is diagnosed by the system. It is divided into more than one domain (for possible device types and nodal failures). Associated with each of the domains is a set of diagnostic rules which are ordered in a hierarchical order to improve performance. MIND is an expert system for VLSI test system diagnosis [WIL85]. It utilizes a data base to store information about different tests (such which test is fired, the outcome of the tests) and an efficient hierarchical rule base which consists of rules with IF..THEN..ELSE format arranged in a tree structure which are general at the top and specific at the bottom (symptom-action for specific parts of the test system). The knowlege-base of the system described in [WAW89] consists of diagnostic knowledge represented by: facts, rules, and *frames*. Facts and rules are represented in predicate calculus [AMB87]. Frames are efficient constructs for representing electronic devices, they can be thought of as hierarchical data templates for describing and recognizing objects.

[KRO89] describes an expert system (CIRCOR) for repair oriented diagnosis of NMOS and CMOS digital circuits. The system capacity is limited to handling relatively small circuits described at transistor level. CIRCORs knowledge base consist of rules for recognizing faulty circuit portions and the correct action. The rule components are described in terms of connectivity information and circuit components (faulty circuit description->correct circuit description). This system is classified as a rule-based system since its rules are similar to (*symptom->fault*) rule structure and are not detailed enough to be classified as a model-based system (see next section).

[ELL90] is a general expert fault finder which performs fault isolation on any system which could be modelled on lowest replaceable units (LRUs). The system utilizes a data base which consists basically of an array of LRU records, which contain information such as name and other statistical information. The inference procedure uses *Fuzzy Logic* [LEE79] for *premise* matching certainty and combining premise certainties in determining which rule to fire. The rules and LRUs are manually added using a Knowledge Base editor and only statistical results and certainties are modified by the system.

Matching is also used in [ODR85] which stores a pre-simulated probability distributions of introduced artificial defects in a fault dictionary like database in an order which depends on the sensitivity of IC parameters. Matching of probability distributions of the parameters is used in the fault finding stage.

The main learning procedure in the AI systems above is by the modification of the rule base by the test engineer. However, the system in [ELL90] associates four parameters with each rule. These parameters are the expert's confidence in the conclusion, the number of times the rule was tried, the number of times it was successful and the output of the fuzzy function (see above).

Algorithms exist for learning rules, which are based on developing logical recognition functions which increase in complexity (get more specific) as more examples are introduced to the system. This algorithm is explained in [MIC83] (which is based on the work described in [MIC81]) and is improved in [RIC89], where it is used in a fault diagnosis problem in a chemical process.

Other systems exist which are used to aid circuit test in other areas [GUY87,LEA88]. These are only mentioned for reference.

2.4.2.1.2 Existing Model-Based Systems

The system described in [TAY90] is aimed at troubleshooting of analogue and mixed signal hybrids and PCBs. It guides the technician to the desired probe location on the physical layout of the hybrid and contains the following information: test plan that defines the specification of a correctly manufactured circuit, the schematics representing the circuit's connectivity information, the physical layout depicting component locations, and the knowledge of the circuit's normal working behaviour which is presented as functional block diagrams at several hierarchical levels.

The system above operates on knowing the task of the specific hybrid or PCB under test using behavioural information of functional components of the main circuit. The approaches described in [MCK89] and [BEN87], however, contain behavioural information of primary components of the circuit. [MCK89] introduces a technique for diagnosing analogue circuits by producing a behavioural model which accommodates the imprecise nature of analogue circuits. A model of the circuit is formed from the constraints imposed by behaviour of the components and the interconnections, which are modelled as sets of rules or relationships between the parameters at their interconnections. The values of parameters within the circuit are deduced by propagating the effects of measurements through this model. Faults are implied from the detection of inconsistencies, and located by suspending constraints within the model.

This approach was initiated in [DAV84, DEK87, DEK84]. [DAV84] mainly dealt with relatively small combinational circuitry, while [DEK87] dealt with simple transistor circuitry. In [ROGC89, ROGD89], the approach was modified to be applied to larger combinational and sequential circuitry.

2.4.2.1.3 Existing Hybrid Systems

The rule-base in [HAV89] is arranged in a 'tree like' structure for each unit (rule). Each unit contains a symptom (or symptoms which differ in complexity). Associated to symptoms of a unit are *action*(s) which could be repair procedures, additional tests to carry, questions to the user, and heuristic search rules which could be modified according to the answers to the previous questions. Different approaches of developing this model is described in the paper.

[PFL89] describes an expert system which controls the action of an Automatic Test System and learns from its historical actions. The rule-base contains diagnosis rules based on diagnostic experience. Physical knowledge of the unit under test (UUT) is incorporated into a causal UUT model.

In both of the systems above, the most difficult task is to form the rule-base which contains information based on interviewing the expert (engineer or technician). Since knowledge acquisition based on interviewing the expert is not efficient (although sometimes is the only way) and requires updating when the rules or the system is changed/updated.

2.5 Conclusions

Many fault diagnosis systems implemented in the previous decade and early stages of this decade solved the diagnosis problem in electronic circuits (especially analogue) by simulating circuit behaviour by using mathematical equations, which are based on circuit theory [BIE81, RAP83]. The reason for this could have been the relatively smaller size of circuitry which were manufactured then (which could be represented in equation forms). The VLSI 'revolution' has increased the complexity of the design which could be manufactured on an IC. This reversed the previous situation in that it was quite cumbersome to represent a circuit with typically more than 100,000 components in an equation form! However, it is common to have a data base with large amount of stored data and efficient manipulation procedures (which has witnessed a significant reduction in expense in the last few years.

In addition to fault dictionaries and AI-based systems (discussed in this chapter), other systems which store and manipulate test data are data management systems. These store large amounts of data produced from multiple testers [MAN86, WIN86, BUS86] and perform different analysis tasks on them which, according to [MAN86], decreased unnecessary repair operation in the manufacturing cycle. The storage and management of design data is also performed frequently as in [CHE88, ECK88]. [TUR90] and [MAN86] point to the importance of collecting test data and the possibility of using them to assist in the diagnosis process, however, they elaborate no more.

The most common approach for using a fault dictionary in testing analogue, digital, and mixed circuits is simulation-before-test [BEN82, WIL86, HOC79, LIN85]. And in most cases the data stored consists of simulated faults on the circuits (Stuck-at faults in digital and nodal voltages in analogue).

As indicated by McKeon in [MCK91], fault dictionaries based on prior experience of faults on the same board are more appropriate than those obtainable by simulation for analogue circuits. McKeon further points that, circuit boards of the same design will be more susceptible to some faults than to others, since they have the same design deficiencies, are stressed in a similar way and use the same types of components. The expectation is that, after a period of time, faults which occur are likely to have occurred before. At each failure, the fault and its symptoms are stored in the dictionary. However, no serious attempts have been made in implementing the above. This could be due to the difficulty of obtaining test data from an industrial source by the researchers.

Fault diagnosis is one area in which AI-based systems have proved successful, hence, many systems exist which are used in diagnosis in different areas (chemical, mechanical, ...etc.).

The AI-based systems reviewed use one or both of the model-based or rule-based system approaches. Both approaches store and manipulate circuit specific information (models and rules). In most of the systems, the type of information stored is related to the circuit topology. This is true for both AI-system types. Model-Based systems store information describing the correct behaviour of the circuit under diagnosis and rule-based systems have rules related to the diagnosis of the circuit (e.g., if component1 is faulty check component2). Any additional databases stored data which the rules operated on.

The updating mechanism in the systems is mainly performed by adding new rules *manually* (since they depend on circuit topology, they are usually implemented by the test engineer) to the system using an editor. Any change of circuitry or modifications/additions to the rules are performed manually. However, some systems have the ability to record the historical success of the rules and take it into consideration when the rule is to be fired [ELL90]. Also, systems exist which deduce information from the already existing knowledge by either calculation or logical deduction.

The review did not reveal any systems which mainly perform their diagnosis by exploiting test failures produced by ATEs. Most of the systems depend on electronic knowledge about specific circuits and/or technical diagnostic rules stored in their knowledge-base.

Investigation of AI techniques revealed that the use of a machine learning technique may be employed for investigating the failure reports in the data. However, so far such techniques have only been applied to small problems [KOD85]. One of the aims of this work is to investigate extending the application of this technique to the cases in the data and finding its limitations.

CHAPTER THREE
A DETAILED EXAMINATION OF A TEST AND DIAGNOSIS CASE

3.1 Introduction

This chapter considers the test and diagnosis case, mentioned in chapter one, more closely. Specifically, the case of one of the most common boards in the company is investigated. At this stage, all of the data received from the company were associated with this board.

The investigation considers the PCB design with emphasis on aspects related to the information in the failure reports in the data. In addition, the data are analysed.

This chapter aims to set the scene for the next chapters since the PCB concerned will be used in the research work associated with the majority of the following chapters.

3.2 General Design of the PCB

The board contains eight modules called ***individuals***. Individuals are different rows on the PCB with identical task and circuitry. The bottom row contains the digital control circuitry. The overall design of the PCB is shown in figure 3-1.

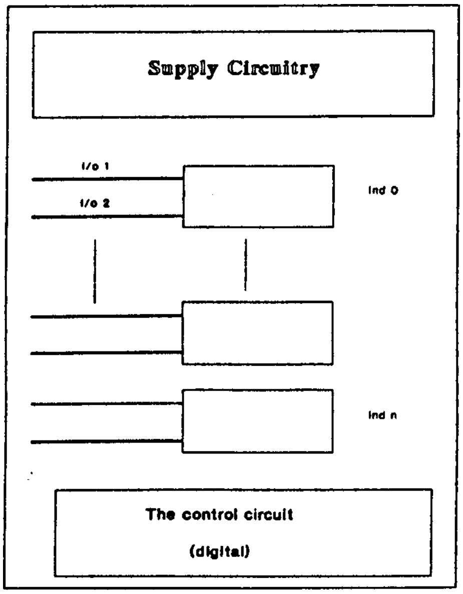


Figure 3-1: The overall design of the PCB.

The figure shows the different individuals (ind), with each individual containing two i/o lines which are the means of contact with the outside world. Most of the circuitry is *analogue*, however, the control circuitry which is located at the bottom of the PCB is mainly *digital* since it performs the task of synchronizing the operation of the different individuals.

In general, the components of the board vary from the complexity of very large scale integration (VLSI) chips, with highly complicated design and multitasking functions, to capacitors and resistors. Each component on the board is identified by a component position number (CPN)[1]. Components of the different individuals are such that components in the first individual have round numbers (i.e. are multiples of 10 such as 40, 150, 490, . . . etc.) while corresponding components in the other individuals are identified by the component number in the first individual, added to it the individual number. For example, the components corresponding to 490 in individual numbers 1 and 4 are 491 and 494 respectively. This characteristic of the CPN assisted in the identification of components of the same type in different individuals.

3.3 The Data

A failure report consists of one or more failed tests. The failure of a given test is usually associated with a number of parameters on the report slip. These parameters are as follows :

Test Number Tests are divided into groups according to their functionality, for example *processor tests*, *power tests*, . . etc. This parameter indicates the group number of the failed test.

Test Name Identifies the test within the groups described above. For example, power tests might include *diode test*, *rectifier test*, . . etc.

Test Condition Indicates the conditions under which the test is applied, e.g. measuring the voltage by applying a load of say 50 ohms.

Measurements Values of parameters measured during the test, such as voltage, current, or gain.

The following is an example of a failure report which consists of two failed tests :

Test 1:	**Test Number:**	*19*
	Test Name:	*Analogue-Digital Signal Noise*
	Test Condition:	*Load = 1500 ohm*
	Measurements:	*Gain = 20 db*
Test 2:	**Test Number:**	*22*
	Test Name:	*OVP Circuit - Thyristor*
	Test Condition:	*Line A set*
	Measurements:	*Current = 20 ma*

[1] Such numbers exist for all the boards manufactured in the company as will be seen later.

The tests and their parameters in the above example are chosen for the sake of illustration only.

The tests in the data could belong to any one of 20 groups, each identified by a test number. Each group could include up to 25 test names applied under different conditions (up to 15 per test name).

Each report contains a number of failed tests for some particular board. A single slip might describe from 1 up to 50 or more failed tests. Typically, a report describes in the region of 8 failed tests. The report is intended to guide the diagnostic technicians in locating the fault in the board to the smallest replaceable unit (SRU). Each report in the data is associated with the CPN of the correct SRU diagnosed by the technician at the time of failure.

The number of reports received from the company for this PCB totalled 497, received in three stages.

When the failures of the different individuals are treated separately, a relatively small number of reports are associated with each fault, however, when the failures of similar components in different individuals are treated together then the number of reports with each fault increases enough to make analysis possible. The bar chart in figure 3-2 illustrates the number of reports associated with the *most common* failed components in the data. The failures point to components of similar type across all individuals where CPN ends with 0. CPNs with * indicate a faulty component which is not a member of an individual, but is a member of either the supply or the control circuitry (see figure 3-1). Table 3-1 points to the component type corresponding to the CPNs in figure 3-2. Slics and Slacs are complex mixed analogue/digital ICs.

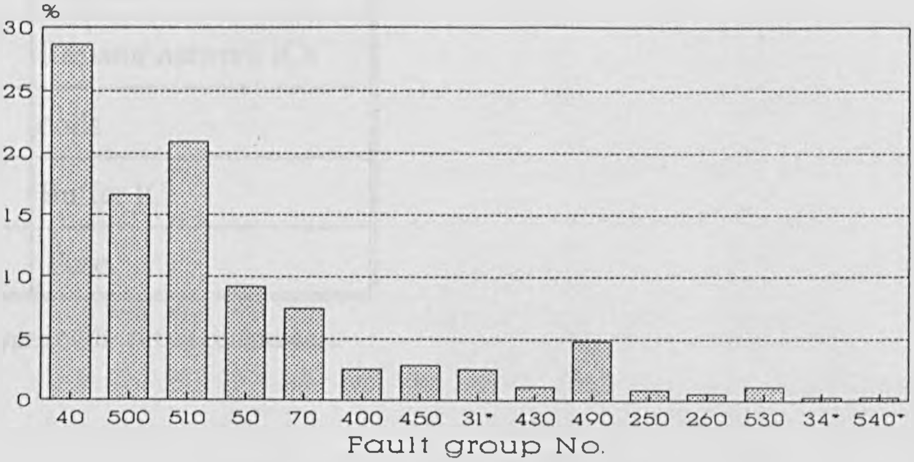


Figure 3-2: A chart of the faults in the data.

It can be seen from the bar charts, that components 40, 500, 510, 50, and 70 are the most commonly failing components. The common failure of 40 and 50 are due to the large number of tasks that each of these components perform in the circuit. It can also be noted, by analysing the data, that in many cases tests belonging to certain components fail due to the failures of 40 or 50 (examples of these will be given in chapter six). By understanding the operation of the circuit, it could be noted that these components control the operation of some others in the circuit.

CPN	*Component Type*
40	Slics
500	Ring-Relays
510	Test-Relays
50	Slacs
70	Bridge Rectifier ICs
400	Capacitors
450	Capacitors
31	Processor
430	Capacitors
490	Thyristors
250	Resistor network ICs
260	Resistor network ICs
530	coils
34	Buffer IC
450	Fuse

Table 3-1: Types of the circuit components.

3.3.1 Ambiguities in the Data

By analysing the reports in the data some ambiguous cases could be noted, some of these cases repeat for a number of times in the data. Some of these are mentioned here briefly and will be the subject of a more detailed analysis later.

1. Some reports are identical in the tests that they contain, however, they are diagnosed by the fault finder to be due to completely different components.
2. In some reports, a single test (or a group of tests) fail(s) in 7 of the 8 individuals. However, the component which is diagnosed to be the faulty one is in the 8th individual.
3. A relatively large number of components diagnosed to be the reason for the failure of some reports with small number of tests. This is thought to be due to fault finder's error, i.e. changing many components by trial and error till replacing the faulty one.
4. In some cases the report seemed to be incomplete. This is thought to be due to the fault finder.
5. The failure of some tests, which are known to test a certain component, are diagnosed to be due to a completely different component.
6. A test (or a group of tests) fail(s) in a component of a certain individual, however, the failure is diagnosed to be due to the same component, but in a different individual.

3.4 Investigation of the Problem Domain

The main characteristics of the diagnostic problem faced by the firm are :

1. The manufactured PCBs have complex mixed signal (analogue and digital) circuitry with high degree of interaction among the majority of its components, and in particular the complex IC chips. This accounts for the complexity of the test failure patterns which characterizes the reports in the data.
2. Un-availability of information regarding the operation of the circuit and the task of its components.

3. Un-availability of technical information regarding diagnostic repair. Hence, diagnostic technicians rely mainly on experience in recognizing patterns of failed tests and associating them with the corresponding faulty components.

6. A total of 500 failure reports exist, in addition to those which are continuously generated by faulty PCBs.

Due to the complexity of the PCBs and the data (described in 1), the use of an AI-based technique is considered.

(2) and (3) make the use of a model-based or a rule-based system impractical, due to the lack of circuit information and diagnostic rules (other than simple heuristics adapted by the diagnostic technicians).

Analysis of the problem pointed to two AI techniques which could be used to investigate the bases of a failure diagnosis system which could utilize the reports in the data and those produced by faulty boards in the future. The first uses a machine learning technique in analysing the reports in the data, while the second uses a technique similar to that of the diagnostic technicians. The rest of this thesis will consider each technique in a greater detail.

CHAPTER FOUR
THE AUTOMATICALLY GENERATED DECISION TREE METHOD

4.1 Introduction

In chapter two (section 2.4.1.2), three learning techniques, typical of those used in machine learning, were described. In this chapter, the use of Michalski's technique [MIC81] is considered in constructing a decision tree which is based on the functional test data (section 1.2). The rest of this section will argue the choice of this technique.

Decision tree construction is a popular technique for building systems that can learn to classify example data, in our case symptoms classified according to faults. The algorithm described in [QUI86] utilizes a typical sample size of approximately 400 examples to deal with large numbers of examples (several thousands). In our case, the training data totalled approximately 500 example reports, hence we could afford to use a more computationally expensive technique for constructing recognition functions at the nodes of a decision tree.

Learning by using version spaces ([MIT77] and [MIT78]) is not practical for the construction of recognition functions at the nodes of a decision tree. This is because a decision at each node will require a complete version space (i.e. both an S and G sets). For relatively complex problems the size of the resulting tree could be unmanageable. In addition, learning with version spaces does not involve searching to identify the best description of a given concept. The description of a concept is determined by the order in which examples are presented.

In general, recognition functions are not unique. In order to deal with the failure cases in the data, it is desirable to experiment with various functions to improve the performance of the system. In the case of Michalski's technique [MIC81] many descriptions of the required concept are searched. These descriptions are compared with each other in terms of sparseness.

From all above, the use of Michalski's technique was considered for classifying the failure reports in the data. Although, as mentioned above, this technique is computationally expensive, it offers flexibility in decision making at each tree node by using recognition functions (section 2.4.1.2). This flexibility arises from the ability to experiment with the generality of the recognition functions. This chapter will consider implementation details of extending this technique to the classification of the test data.

4.2 Generating a Tree for Fault Finding

This section describes the practical implementation of Mickalski's technique. The outcome of the technique is a tree-like construct which is searched during fault diagnosis.

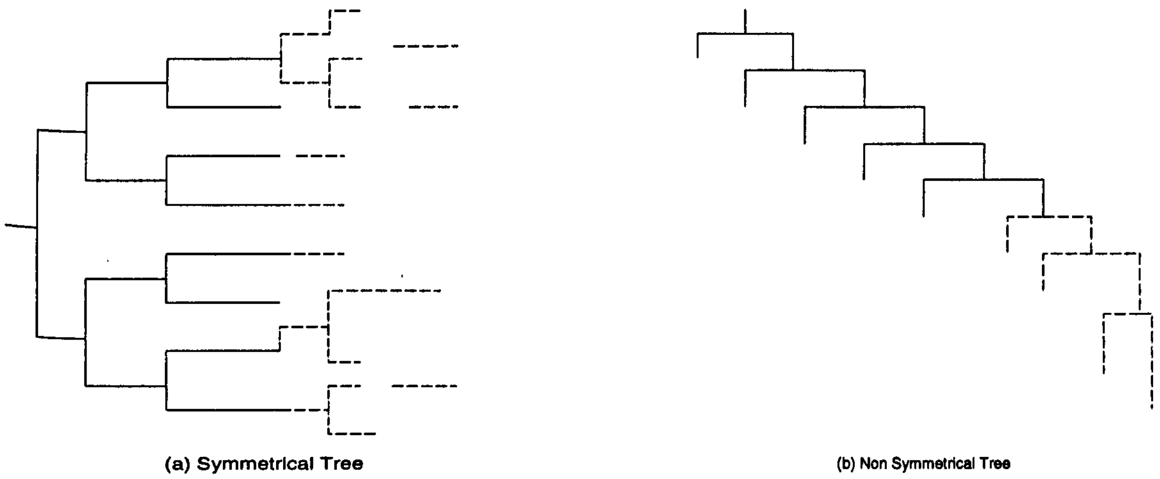
As mentioned in chapter two (section 2.4.1.2), the technique operates on groups of example objects by making generalizations by similarity detection. As a result, an expression is produced for each group which describes the examples in that group only.

Due to the relatively large size of the data, a special arrangement was required for grouping the data. From inspection of the data, it was clear that some faults were more easily identifiable than others. These faults tended to produce one of a small number of clearly distinguishable failure reports. A binary tree of fault groups was thus constructed as follows. The set of all possible faults was divided into two groups such that one contains the most reliably identifiable faults and the other group contains all the others. 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. This 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 and so this is where the most easily identified groups should be placed.

As a result another tree is produced which consists of expressions describing each of the groups. The next section will discuss the aspects related to generating the different trees in greater detail.

4.2.1 Organization of the Data

The data are arranged such that the reports which point to the failure of the same fault are in the same group.



(a) Symmetrical Tree

(b) Non Symmetrical Tree

Figure 4-1 :The two types of trees used.

Since the basic concept of this method is to identify different groups with respect to each other, then these groups must be arranged in a form which enables the algorithm to operate on them. The method used for presenting the different groups to the algorithm is to arrange the groups into the leaves of a ***binary tree*** structure, termed here ***data tree***.

The groups of data are arranged into the leaves of the data tree in an order which depends on their ***'reliability'***, with the most reliable fault groups being those which have the largest number of identical reports and can be distinguished from reports of other faults, and hence can be more confidently identified. The classification of the faults according to their reliability is performed manually after analysis of the data.

Two different types of trees were used for experiment; the symmetrical tree, figure 4-1a, and the non-symmetrical tree, figure 4-1b. The tree which provides the best performance will be chosen, as will be seen later.

4.2.2 Generating a Decision Tree

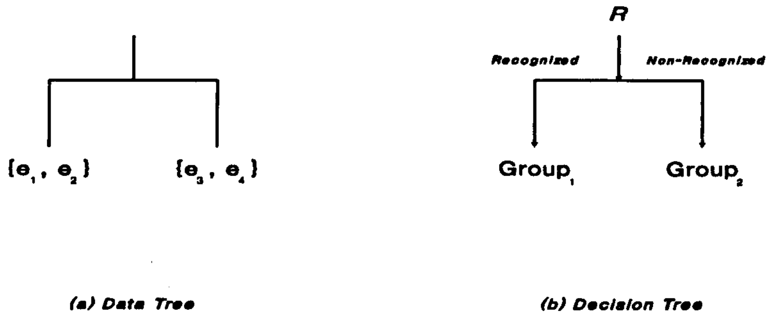
The implementation of this part of the system was based on the theoretical aspects mentioned in section 2.4.1.2 (related to the technique used by Michalski), hence this section will make frequent reference to that section.

Before the generation of the decision tree, the data has to be separated into different groups. The algorithm contains some functions which are based on the definitions of section 2.4.1.2. A recognition function is used to recognize an element in one group against an element in another, $G(e_i/e_j)$. By repeating this procedure for all the elements of the latter group, a recognition function is obtained which identifies the single element of the first group against the whole of the second group, $G(e_i/\{e_j\})$. As mentioned in section 2.4.1.2, this function is in the form of a disjunction of conjunctions and usually contains a large number of terms, each disjunction term containing as many terms as the total number of tests seen in all the data (more than 240 tests) since the function will be in terms of the seen and unseen tests. The best $G(e_i/\{e_j\})$ is obtained by putting the function in the form of a conjunction of disjunctions which requires multiplication of all the terms. The complete multiplication cannot be performed on computers due to the amount of terms involved. For this reason smaller number of multiplications, chosen in a random order, are performed to obtain a limited number of terms (10-100), the *best* of which is chosen, by simple blind searching the above terms, to be $G_{opt}(e_i/\{e_j\})$. Initially, Michalski's concept of the best function (i.e., the function with minimum sparseness) was used. So, in order to find the best term, sparseness must be found for all the terms and the term with the smallest sparseness is chosen for $G_{opt}(e_i/\{e_j\})$. Blind search

was used since no reasonable heuristic (or concept) could be found in order to direct the search for the best term to represent $G_{opt}(e_i/\{e_j\})$.
By repeating the above process for all the elements of the first group a recognition function is obtained which identifies one group but not the other.

So, given two groups the above algorithm can find the 'best' recognition function which identifies one but not the other. This algorithm is applied to each group in the data tree against the rest of the tree and generates a recognition function corresponding to it. This function is placed in the position corresponding to the node separating the group from the rest in another parallel tree structure, called the ***decision tree***. Hence, the decision tree is an exact replica of the data tree in shape except that the nodes of this tree contain the recognition functions which are generated by the algorithm. Each function identifies a group in the leaf immediately below against the group representing the next leaf (each leaf in turn could have additional binary branches).

The data and decision trees, for the example of table 2-1, are illustrated in figure 4-2 (a) and (b) respectively. The decision tree generated contains the recognition function, R, at the node of intersection between the two groups. A report presented during fault diagnosis will be used to search the tree. If the report is recognized by R then the failure is of type $group_1$, otherwise the failure is of type $group_2$.



(a) Data Tree

(b) Decision Tree

Figure 4-2: A simple example of generating the trees.

The discrimination, of examples, achieved with a decision tree constructed in the manner above is not specific in that when an example is not recognized by the recognition function of one group, then it is assumed that the example is a member of the opposite group. However, this is not always correct since an example could not be a member of any of the groups involved in building the decision tree.

A more specific discrimination could be achieved by generating a decision tree with two boolean functions at each of its leaves. The first function identifies the fault group at a given leaf in the data tree against the groups in the rest of the tree, and another function which identifies the rest of the tree but not the group at the leaf. The second boolean function can be obtained by simply exchanging the order of the two groups presented to the function which generates a recognition function between two groups, as mentioned above. If an example is identified by the first function then it belongs to the group at the leaf, whereas if it is identified by the second function then it belongs to one of the groups in the rest of the tree and hence the search is continued down the tree. However, if the example is not recognized by any of the functions then it is assumed to be spurious and is not identified. This avoids any unnecessary search down the tree.

In the rest of this chapter the tree with a single boolean function will be referred to as a ***single boolean function tree(SBFT)***, and the tree with two boolean functions will be referred to as a ***twin boolean function tree(TBFT)***.

The procedure of generating a decision tree is relatively long and takes 15-30 minutes on a SUN 3/260 (M68030 based machine). This procedure is usually referred to as the ***learning*** procedure, since the system is analysing the data and trying to make rules based on them. In the practical use of the system, the procedure will be run each time a considerable amount of data is collected, say each two weeks.

4.2.3 The Fault Diagnosis Procedure

This procedure is usually followed after constructing a decision tree. Given a failure report, the tree nodes are searched for a recognition function which best recognizes the report, and that will in turn point to the fault group which the report is most likely to belong to. Searching the nodes of the decision tree is directed by the recognition function at each node and is depth-first [RIC91] in nature as a consequence of the structure of the non-symmetric tree (section 4.3.1). This procedure is fast and usually takes seconds.

4.3 Performance Evaluation

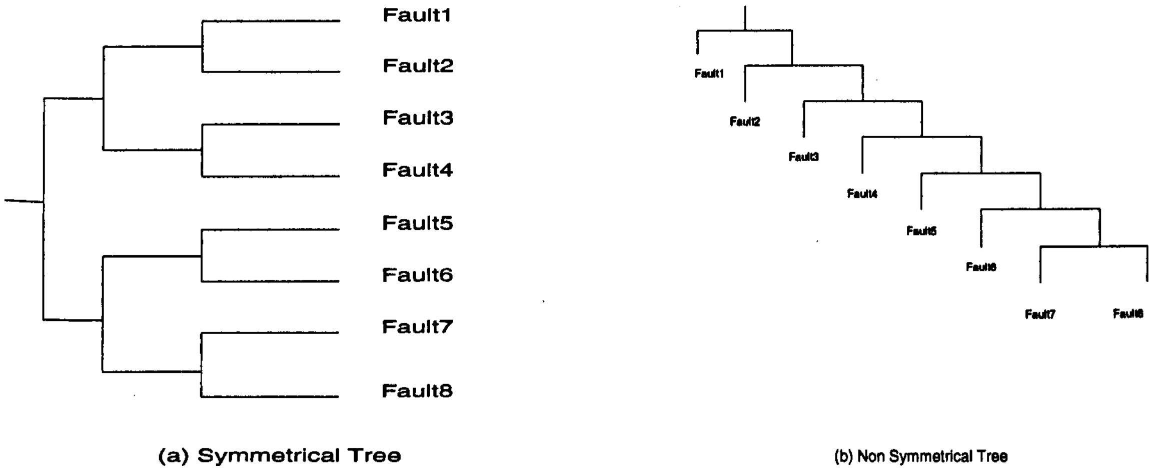
This section describes the work done in choosing the *best* tree characteristics, constructing different trees using the actual data , and analysing their performance.

4.3.1 Biasing the Trees

This section describes the procedure of choosing the different attributes for the final tree to be used in diagnosis. By the tree attributes we mean the type of the tree (symmetrical or non-symmetrical) and the required sparseness (minimum or maximum).

Due to the complexity of the real data, which are characterised by the relatively large number of reports and the different types of ambiguities, some of which were mentioned in chapter three, some dummy data were manually prepared and put in a form similar to that of the real data. The dummy data, which were made to consist of eight fault groups *{fault1, fault2, . . . fault8}*, are illustrated in greater detail in appendix A.

These data are presented to the program after being arranged in the form of both symmetrical and non-symmetrical trees, as shown in figures 4-3 (a) and (b), and the corresponding decision trees are obtained for each case.



(a) Symmetrical Tree

(b) Non Symmetrical Tree

Figure 4-3 :The distribution of the faults on trees.

Each tree was tested by presenting it with a report of a known fault, one of the above, and noting the result. This was repeated nine times for each report. The experiments were repeated with both minimum and maximum sparseness. The results of these experiments are shown in tables 4-1 and 4-2.

It is clear from the tables that the *non-symmetrical tree with maximum sparseness* provided the best set of results, since most of the results obtained by each of the nine trials {**T1, T2,....., T9**} are consistent with the correct fault number in **TF**. The tree identified five of the eight faults with no dispute in all of the trials, while most of the other faults are identified in the majority of the trials. This performance is clearly better than all the other trees generated. The result of this experiment will provide us with the initial setting of the data tree which will contain the real data, and which will be described in the next section.

The results indicate a better diagnosis performance when maximum sparseness is used in building the decision tree. Although this is contradictory to the theoretical facts of section 2.4.1.2, however, it is expected due to the relatively large size of the *data space*. This means that the data (reports) encountered so far are relatively small compared to the number of failures which could occur.

TF	***T1***	***T2***	***T3***	***T4***	***T5***	***T6***	***T7***	***T8***	***T9***	***Sparseness***
8	8	8	8	8	8	8	8	8	8	***Max***
7	7	7	7	7	7	6	7	7	5	
6	6	6	7	6	6	6	6	6	7	
5	5	5	5	5	5	5	5	5	5	
4	4	1	4	4	4	4	1	1	1	
3	3	3	7	3	3	3	3	6	3	
2	1	1	1	2	1	2	2	1	1	
1	8	1	1	1	1	1	1	1	6	
8	8	8	8	8	8	8	8	8	8	***Min***
7	7	7	7	7	7	7	7	7	7	
6	7	7	7	8	6	7	8	8	6	
5	5	5	5	5	5	5	5	5	5	
4	4	4	4	4	4	4	4	4	4	
3	7	6	7	8	7	8	8	8	8	
2	2	2	2	2	2	2	2	2	2	
1	7	6	7	8	7	6	8	8	8	

Table 4-1: Results of tests with the dummy data using symmetrical tree

TF represents the true fault number

TF	*T1*	*T2*	*T3*	*T4*	*T5*	*T6*	*T7*	*T8*	*T9*	*Sparseness*
8	8	8	8	8	8	8	8	8	8	***Max***
7	7	7	7	7	7	7	7	7	7	
6	6	8	6	7	8	6	6	6	6	
5	5	5	5	5	5	5	5	5	5	
4	4	4	4	4	4	4	4	4	4	
3	8	3	8	3	3	6	3	3	7	
2	2	2	2	2	2	2	2	2	2	
1	1	1	8	1	1	6	1	1	1	
8	8	8	8	8	8	8	8	8	8	***Min***
7	7	7	7	7	7	7	7	7	7	
6	8	8	8	8	7	8	7	8	7	
5	5	5	5	5	5	5	5	5	5	
4	4	4	4	4	4	4	4	4	4	
3	6	8	8	3	7	8	8	6	7	
2	2	2	2	2	2	2	2	2	2	
1	6	8	8	8	7	8	8	6	7	

Table 4-2: Results of tests with the dummy data using non-symmetrical tree

The use of minimum sparseness chooses the most restrictive recognition function of the example data. This means any of the less frequent reports, which have not seen before, will be rejected since the function is *very restricted* and hence does not include examples which do not closely resemble the data used in building the decision tree. Using maximum sparseness these less frequent cases will be recognized, since the recognition function represents the sample in the most general way. The use of maximum and minimum sparseness was illustrated in figure 2-1.

The maximum sparseness obtained with the dummy data will also apply with the real data since the former were made to resemble the actual data especially in the proportionality of the ratio $\frac{number\ of\ tests}{number\ of\ possible\ faults}$ to that of the real data. Hence a similar data space will be expected.

4.3.2 Building Trees Based on the Data

This section considers the different trees generated and their performance during diagnosis.

In this section the technique is applied to the real data. However, before presenting these data to the program they were put in a form which is similar to that of the dummy data (see section 4.3.1). Different reports were divided into groups according to the faults which caused the failure of their tests. The faults were arranged on the leaves of a non-symmetrical tree according to their reliability (the more reliable groups are arranged in the top of the tree), since this type of tree produced the best set of results during the biasing procedure which was mentioned in section 4.3.1.

The test data applied to all of the trees were the same, so that the performance of all the trees in identifying the data can be compared. The test data were chosen to be of different levels of complexity in order to monitor the performance of different trees and compare them. Tests were applied to each tree to find the limit at which it fails to identify faults. At the start of the testing procedure, reports were identical to the data used in constructing the tree. This was to test whether the tree could recognize itself (i.e. its own data). A failure in any of these tests indicates a programming error since the tree should recognize the examples used in its learning procedure. In the second stage of the testing procedure the reports were slightly different from the data (i.e. one or two tests missing or added from/to the top, middle, or bottom of the failure list). The purpose of these tests was to identify whether the tree can identify any slight changes in the data and how much change it can identify, if any, since this is what actually happens during the testing procedure in the company. Failed tests which cause the failure of a component are similar but could include more or less failed tests. If the above tests are passed, then the next step is to test the tree with data which require a very close decision between one or more of the faults. The length of the test data applied was 1-25 failed tests/report.

One of the ambiguities contained in the data is the existence of reports which indicate the failure of the same tests, but which are diagnosed as being due to different components. These were mentioned in 3.3.1. The first problem encountered when constructing trees with the real data is the failure in the building procedure when the algorithm is trying to find a recognition function which recognizes one group but not the other, with both groups having identical reports. Since the building procedure cannot proceed without discriminating between all the groups, these ambiguities had to be dealt with.

The way in which these ambiguities are isolated is by creating a new group which contains all the reports duplicated in more than one fault and assigning a group identity which differentiates it from the rest of the groups in the data. The procedure followed in assigning group identities is to include all the fault position numbers in the identity in an order depending on the frequency of the report in the different faults (e.g., 45-66-77 for a report failed most frequently in 45). By performing this for all the duplicate reports in different fault groups, the tree building procedure terminated successfully.

In creating test data for the trees by deleting or adding additional tests to the failure list, it is necessary to look back at the data and check that none of the reports used in testing is a report for another fault. For example, by deleting a single failed test in a list of failures which indicate a fault in component 53 the new test example could be one of the examples in the data which are grouped under faults of 47.

The initial trees generated using the real data were characterised by the large size of the recognition functions which in turn resulted in relatively large tree sizes. An alternative to sparseness for dealing with this problem has been devised. This will be introduced in the next section before considering in detail the trees built with the data.

4.3.2.1 Compactness : An Alternative for Sparseness

Due to the large amount of data and the large amount of individual tests, the boolean functions obtained were large. This in turn led to an increase in the size of the decision trees obtained. For this reason, an alternative concept is developed here for choosing the best term for $G_{opt}(e_i/\{e_j\})$. This concept, which is termed ***compactness***, is used instead of sparseness, which was suggested by the original approach, and aims to reduce the size of the decision tree generated. Such a concept has not been used in any of the work mentioned in the literature (see chapter two).

Definition

> ***Compactness is defined as the difference between the number of elements (in this case reports), which a given recognition function describes, and the number of conjunction (product) terms in the function.***

The best term should identify as many reports as possible in the data and it should also have a small number of terms (i.e. the function with small size). This means a *maximum* compactness should be used for a small and efficient tree. The performance of the trees implemented with maximum compactness will be described later in this chapter.

To implement the concept of compactness in the algorithm, each term in $G_{opt}(e_i/\{e_j\})$ had to be used in searching all the data for the number of reports it recognizes.

Examples of table 2-2 are used to illustrate the concept of compactness. The compactness of the first entry, T_0, is calculated by finding the number of examples recognized in $group_1$, 1, and the size of the term, 1. Hence the compactness value is 1-1 = 0. In the same manner the value is calculated for all the other terms, as in table 4-3 below:

$G(e_1/\{e_3,e_4\})$	Compactness
T_0	0
$T_0\overline{T}_3$	-1
$T_0\overline{T}_2$	-1
$T_2\overline{T}_3$	-1

Table 4-3: Compactness values for each term in $G(e_1/\{e_3,e_4\})$.

The term with maximum compactness is T_0. This choice is not in agreement with the choice made using sparseness, but, due to the simplicity of the example all of the terms recognize the same number of examples in the data space and the result is not incorrect. The need for the use of compactness is clearer using the real data.

4.3.2.2 Trees Based Upon All the Data

In these sets of tests the technique was used to construct decision trees using all the available data. The trees generated generally differed in the concept used to produce the best recognition functions at their nodes i.e. sparseness (both maximum and minimum) and compactness and also in the use of SBFT and TBFT.

The reaction of the different trees to the test data was nearly the same in that all trees managed to identify any report which was identical to the report in the data used to build the tree (this is expected, since any wrong identification in this case points to an error in implementing the algorithm). However, when the test data presented were slightly different from the original data then in most cases the identification was incorrect. In general, the main difference among the trees was in their size. Table 4-4 shows the different options used in building the trees and the sizes obtained. The time taken to build a tree was about 4 hours on a SUN 3/260.

SBST/TBST	***Sparseness***	***Compactness***	***Size(K Byte)***
SBST	Min	---	530
SBST	Max	---	400
TBST	Min	---	930
SBST	---	Max	380
TBST	---	Max	600

Table 4-4: Comparison of sizes of different trees

The smallest tree was obtained when maximum compactness was used with SBST in building the tree. Since the performance of all the trees is similar (as mentioned above), the TBST tree is ignored since it does not improve the performance and it nearly doubles the size of the tree from the case where SBST is used.

4.3.2.3 Trees Based Upon the Most Common Faults in the Data

As mentioned before, the trees generated in section 4.3.2.2 were based on analysing all the data. Most of the reports identified using these trees were either exactly similar to the reports in the data or very slightly different from them (which occurred very rarely). For this reason the amount of the data presented to the tree building algorithm was gradually reduced and the performance of the trees to the same tests was monitored. The performance of the tree was tested by picking a number of the most common faults and implementing the tree with them. If the performance showed an improvement, it was possible to find the limit at which the performance deteriorated.

This poor performance in identifying slight changes in known reports could be due to the large size of the data and to the large number of faults existing for which only one or two examples (reports) exist. The reports of uncommon faults are included in the boolean expression generated in the recognition function equally with the reports of the common faults, with no indication of the frequency of their occurrence. For this reason the next stage was to choose some of the faults which most commonly appear as the cause of the failure reports in the data.

When referring to the performance of the trees below, no comments will be made about testing reports which exactly match the reports in the body of the tree, since these were identified. The important improvement in the performance is required in identifying reports which differ from the exact reports but contain the key tests (main tests which when failed indicate a failure of a specific device).

The ***eight*** most commonly occurring faults were chosen, which are the faults in components 31, 40, 41, 42, 45, 47, 514 and 517. The decision tree with these data was built in about three hours, occupying about 33Kbytes. The tree could identify slight variations in the data for relatively long tests (10-20 failed tests), but, when slight changes were made to the data with relatively short tests (less than 10 failed test), the tree could not recognize the fault. The use of this tree is not practical since most of the reports used in the data belong to the failure of the same component but in different individuals, components 40-47 are ICs of the same type (Slic) and the rest are test-relays. Although the above components are the eight most common ones in the circuit, the devices represent only two types of components in the circuit and hence do not cover enough failures of the board.

A better modification to the data was to combine the same faults occurring in different individuals, treat them as one fault, e.g. slic faults, slac faults,...etc., and let the program form a tree on this basis. This is reasonable since similar components in different individuals perform the same task and are subject to similar conditions of operation, as the circuit of each individual performs the same task in its operating environment (telephone exchange). In fact, similar components in different individuals could provide additional examples of the possible failed tests which could lead to a failure in a given component. The ***ten*** most common components in the circuit, ignoring their individuality, are:

500	***510***	***70***	***490***	***40***	***50***	***400***	***450***	***31***	***530***

most ***least***

The above procedure was performed in two steps:

In the first stage, the faults were grouped as mentioned above and the ten most common faults were chosen, but, the individual numbers in the data were not removed. This produced a tree of 210Kbytes in about four hours. When slight changes were made to relatively long reports (more than 10 tests long) the fault could be recognized. However, if the number of failed tests which led to the fault were less than 10 then the tree looked for the fault which contained most of the failed tests and chose it as the fault.

Next the individuals were removed from the data itself and the learning program was run on the same data as before. The size of the tree was reduced to about 73Kbytes and the time taken for the program to build the tree was about 30 minutes. The tree could now recognize most of the reports which contained slight variations to the reports in the body of the tree. In cases where the test data were similar to the actual data in more than one fault, the tree could give very accurate answers. There were some cases from the above examples in which wrong identifications were made.

The above procedure was repeated by taking the most common 14 faults and the most common five faults in the data. In the former case the tree was efficient and could make accurate decisions. The same could be said about the latter case, however five faults are not enough to represent most of the faults which could occur in the whole circuit.

It is clear that the most common faults chosen above are not consistent with the analysis of chapter three which was carried out on the data. This is because only a portion of the data was provided at this stage and the analysis above was carried out on the total amount of data provided for this board.

4.4 Conclusion

In this chapter the test and diagnosis case, mentioned in chapter three, is investigated by employing one of the standard machine learning approaches used for concept acquisition. In order to use this approach, the existing data must be grouped in such a way that a decision tree can be produced. The leaves of the tree represent individual faults, the higher nodes represent groups of faults with the root node representing all possible faults. The learning algorithm associates each node of the tree with a discrimination function which is used to determine which subnode group of faults is indicated by a given failure report.

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 and so this is where the most easily identified groups should be placed. Discrimination or recognition functions are added to each node of the tree as follows: Given two groups of faults, an algorithm [KOD85] 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, it was found that the more general functions, that is 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.

The decision trees are relatively straightforward to build and provide very fast identification of new reports. However, they have the following disadvantages ;

1. difficulty in handling ambiguous data, such as identical failure reports for different faults,

2. the identification was found to be unreliable. Only a small number of reports were correctly identified and almost all of these were exact matches of examples in the training data.

3. the tree will become out of date and will require re-building periodically. This occurs when the training data is no longer representative of the real data.

When a given report is being identified it is compared to all boolean functions in the decision tree and the fault group associated with the function which recognizes the tests of the report is chosen. The results of applying the tree approach to the data indicated that it is capable of identifying reports which closely resemble those used in building the tree (training data space), however it does not offer any flexibility in handling reports which do not closely resemble the reports from the training data space. This flexibility is necessary in fault identification since, in many cases, reports indicating the failure of a given component are significantly dissimilar, although they include the failure of some tests which give the characteristics of the failure of that component. Furthermore, the data included some ambiguities which are characteristic of the failure of some devices. These are usually recovered by diagnostic technicians in the company by using some heuristic techniques based on experience. It would appear that the training set (some 500 reports) is a relatively small sample from a very large space of possible reports. All of the above suggested the need for a more heuristic approach to be included in the diagnostic process. This might resemble the process followed by the human experts (diagnostic technicians).

In addition to the above, the learning process is required to be progressive, such that the information in the system is kept up to date as much as possible.

The identification of reports which are different from the training data was not successful when all the data were used, due to the large number of reports and faults occupying a large search space. However, when the most common data were chosen, the performance was improved, especially after ignoring individuality. As the number of data decreased the performance increased, which is as expected.

In order for the method to proceed, the duplicate reports in different fault groups were manually manipulated. This is an additional disadvantage in the method since it requires further manual intervention in the procedure.

A human expert uses heuristics and some knowledge about the data which could be achieved by observation. It is better to use a flexible method which uses such heuristics in finding the fault.

CHAPTER FIVE
THE DEVELOPMENT OF A FAULT DICTIONARY BASED SYSTEM

5.1 Introduction

As it is clear from the investigations carried out in the previous chapter, the automatic tree generation method is capable of identifying reports which closely resemble the ones used in building the tree (data space), however, it does not offer any flexibility in handling reports which are slightly different from the ones in the data space. As initial analysis of the data revealed, this flexibility is necessary in fault identification since in most cases reports indicating the failure of a given component are not exactly similar, but contain some common test failures which indicate a malfunction in some of the functions the device performs (and the test program tests for). Furthermore, the data included some ambiguities which are characteristics of the failure of some devices. All of the above required some heuristic approach to be included in the diagnosis process. Such an approach is usually followed by the diagnostic technicians who develop some knowledge, some of which may be in the form of heuristics, of the diagnosis process.

With experience, diagnostic technicians learn to associate certain failure reports with specific faults by being able to recognize tests in the report which are related to the failure of certain components. This is usually possible because of previous experiences involving similar reports. For this reason, recognition becomes clearer in the case of components which fail frequently with similar sysmptoms, i.e. similar failed tests in the failure report.

When a new failure report is encountered, a technician will compare the failed tests in the report with test sequences previously encountered in failure reports associated with different faults, as he/she remembers. It could be said that the technician is *matching* the tests in the report with failure reports in his/her *memory*, which could be said to resemble a database (or knowledge-base) of reports. If the tests in the new report closely match those in one of the previously seen reports then the fault associated with this report will be considered by the technician to be a suspect fault for the new report. This process will become more complex when the new report matches reports of more than one of the faults encountered before. In this case technicians may use other information about the reports which is available (i.e. information tracked and remembered) by him/her.

The diagnosis approach followed by the diagnostic technicians is efficient since it enables them to recognize a large proportion of the faults especially those which occur frequently. In addition, the approach is flexible since it is non-algorithmic and hence heuristics could be used, where available and used by the technicians themselves, in improving the recognition process.

This chapter describes the development of a system which is modelled on the conduct of a diagnostic technician during the failure diagnosis process. The system performs a diagnosis by matching a failure report with dynamically stored information about previously seen reports.

5.2 System Implementation and Main Elements

Investigations carried out as part of this work revealed that one way of implementing a system which mimics the behaviour of a diagnostic technician, as described above in section 5.1, is by making it consist of a number of basic blocks representing the different aspects involved in the diagnosis process performed by the technician which include dynamic storage of information in his/her memory, matching, and using heuristics.

As described in section 5.1, the main requirement of such a system is a store of information, extracted from encountered failure reports, which can be updated dynamically. This section of the system, which corresponds to the memory of the technician where experience is accumulated, is termed the *knowledge-base.*

In order to compare new reports with ones already in the knowledge-base, a matching strategy is required which can perform the comparison efficiently. The matching strategy is embedded in the section of the system termed the *matcher.*

For manipulating and buffering the outcome of the knowledge-base search performed by the matcher, a number of rules are required. These are stored in another section of the system which is termed the *rule-base.*

In addition to the above, a number of heuristics are required for enhancing the diagnosis process. These were obtained from the diagnostic technicians, who develop these with experience. These heuristics usually assist in recognizing certain cases of failure reports.

In summary, the main elements of the system are the following :

i. The knowledge-base.

ii. The matcher. (knowledge-base search)

iii. The rule-base.

iv. Heuristic rules.

The rest of this section will consider each element in a greater detail.

5.2.1 The Knowledge-Base (KB)

As mentioned in the previous section, the knowledge-base is the main store of data about the faults encountered by the system. Investigations carried out as part of this work revealed the need for storing two aspects of the information in failure reports. For this reason, the knowledge-base is implemented in two separate parts (***Test-Pattern KB*** and ***Test-Significance KB***). The next two sections are dedicated to describing each of these parts.

5.2.1.1 Test-Pattern KB

The conclusion made as a result of both analysis of the data and interviewing the diagnostic technicians is that the main factor in identifying frequently occurring faults is the recognition of a test or a sequence of tests which usually fail when the fault occurs. In most cases more than one such test or sequence of tests exist for a given fault. A sequence of test failures occuring in a single failure report is termed here a ***test-pattern***.

Due to the significance of recognizing test-patterns in the diagnosis process, this section of the knowledge-base is dedicated for the storage of the different test-patterns encountered for each fault met by the system, and hence is termed the test-pattern knowledge-base.

The test-pattern knowledge-base contains a list of ***Fault Entries***. A fault entry exists for each fault presented to the system up to the present time, it contains the information required by the system about each fault. Each fault entry contains two parts : a ***Fault Number***, which indicates a fault in a specific component (and represents the CPN of the failed component), and a ***Test Patterns*** part, which contains a list of test-patterns previously encountered for this fault.

Each single ***Test Pattern*** in turn consists of the following parts:

i. ***Frequency Count***: is the number of times that a specific test-pattern causes the failure of a given component.This field is added (and later modified) by the system to avoid storing duplicate copies of the same test-pattern.

ii. ***Tests***: contains all the tests which form the test-pattern.

This representation of ***Test Pattern*** enables the system to copy the diagnostic technician when faced with a test-pattern which is encountered in more than one fault. When faced with such a case, the technician compares the cumulative frequency count of the different test-patterns in determining the most suspected fault.

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 information corresponding to these fields is extracted directly from the failure report since each test in the report has identical fields associated with it. For information on the format of the tests in a failure report refer to section 3.3 of chapter three.

As mentioned above, each fault has a specific fault entry which contains all the test-patterns which caused the failure of that fault. Figure 5-1 exhibits the structure of a fault entry in a tree form.

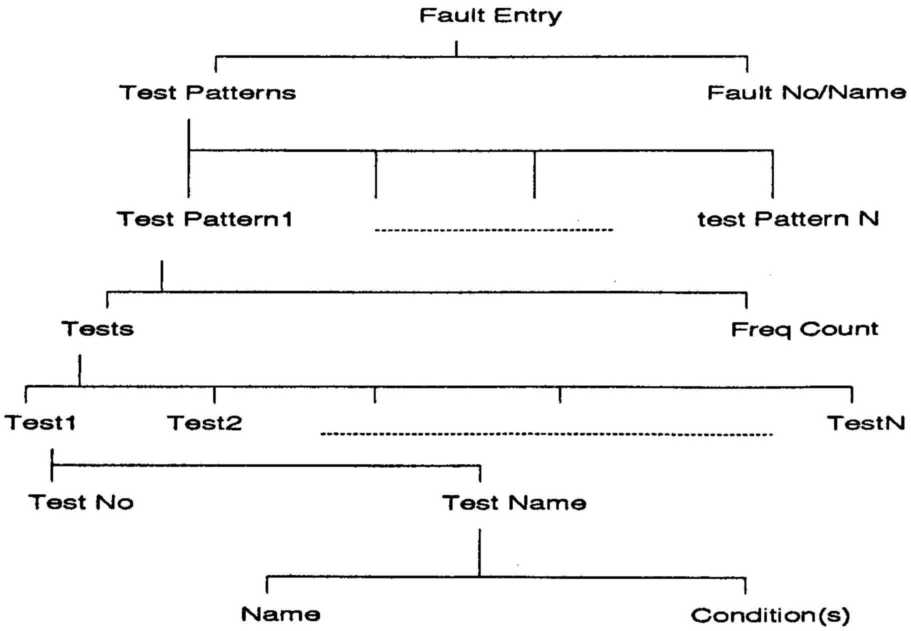


Figure 5-1: The structure of a fault entry

5.2.1.2 Test-Significance KB

In addition to associating the different test-patterns with the appropriate faults, a diagnostic technician will also develop, with experience, the knowledge of the significant tests with respect to certain devices in the PCB circuitry. This is developed by noting the failure of certain tests during the failure of certain devices each time one of these devices fail. Such information about the individual tests in the report will provide an additional measure when comparing a new report with ones stored in the knowledge-base.

The test-significance knowledge-base is developed in order to enable the system to perform the above. This section of the knowledge-base is a table like construct which stores the frequency of each test, encountered so far, in the failure reports associated with each faulty component encountered. Each table entry is allocated for a specific test and stores the frequency information as a histogram of the failure frequency for the corresponding test in each fault.

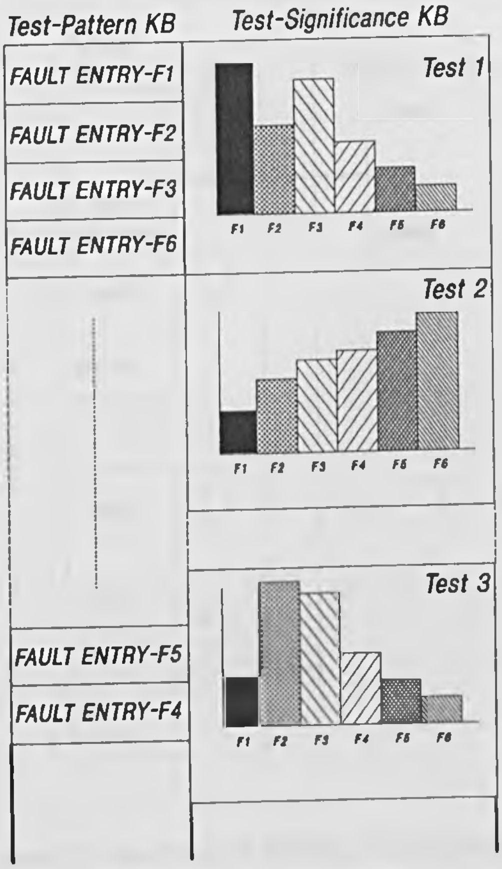


Figure 5-2: The main sections of the knowledge-base.

Both sections of the knowledge-base are illustrated in figure 5-2. The knowledge-base in the figure is constructed in terms of a number of example faults, F1. . .F6, and a number of tests, test 1. . .test 3.

5.2.2 The Matcher (the knowledge-base search)

When the system is presented with a failure report, a number of heuristics are used to detect cases of multiple module failure. These scan the reports and involve examining the failed tests and the modules of failure to look for a pattern of failure which resembles one of the standard patterns, these are the patterns noted in groups of identical failure reports obtained by analysing the data and are made into heuristics. Section 5.2.4 will discuss these heuristics in a greater detail. When the pattern in the report is identified, then the corresponding heuristic action is applied. This basically points to a suspect module and selects the tests corresponding to its failure in an optimal form (mainly by removing duplicates). The group of extracted tests is termed a ***match-pattern***.

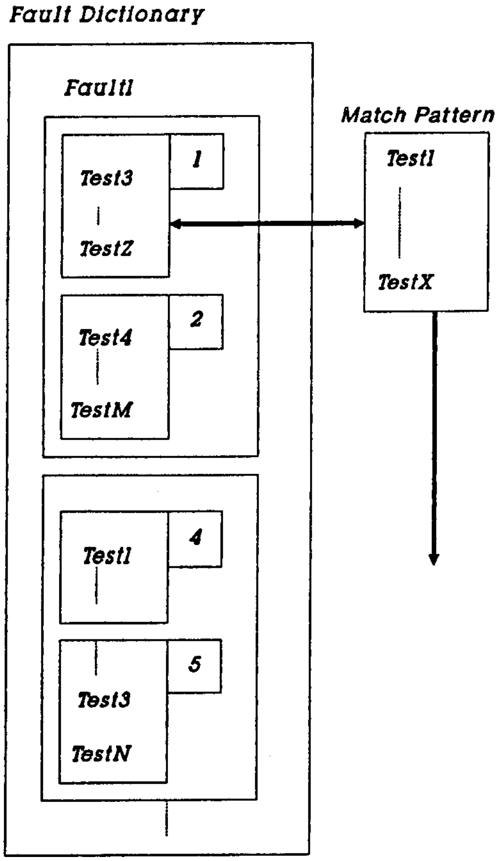


Figure 5-3: Searching the KB for matching patterns.

Next, the match-pattern is presented to the matcher which compares the sequence of tests to each test pattern in a fault entry. The procedure is repeated for each fault entry in the test-pattern knowledge-base, as illustrated in figure 5-3. When the report is compared to each test pattern, the following are marked :

1. The number of common tests, S. (Non negative integer)
2. The number of common tests relative to the total number of tests, P. (Integer in the range 0-9, 9 indicates that all tests are common)
3. The ***Significance*** of the common tests relative to the significance of all the tests, G. (Integer in the range 0-9)

The Significance is obtained from the information in the test-significance knowledge-base. Since the latter contains information about the number of times a particular test failed for a given fault. The significance of a given *Test* in a particular *Fault*, **Significance(*Test*, *Fault*)**, represents *the number of times in which the failure of test **Test** was caused by the failure of component **Fault** relative to the number of times it was caused by the failure of all components.* For example, if a test *t1* fails once in component *c1* and twice in component *c2* and not otherwise, then :

$$significance(t1, c1) = 1/3$$

The **overall significance (*G*)**, mentioned above, of a test-pattern in a given Fault is defined as *the significance of the tests which are common between the tests of the report and that of the fault entry relative to the significance of all the tests in the pattern and the report.*

Note that, 2 and 3 are chosen to be numbers between 0 and 9 only to ease the process of comparing the corresponding scores of different patterns.

4. The frequency count of a pattern, *F*. (Positive integer)

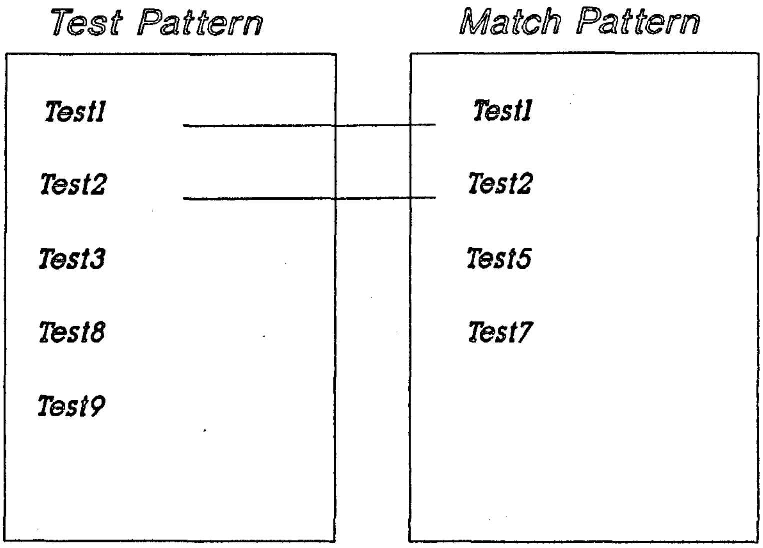


Figure 5-4: Matching example.

in order to illustrate the procedure of evaluating matching scores for a match-pattern and a given test-pattern within a fault entry, consider the example in figure 5-4. The scores for the example are as follows :

1. The number of common tests is 2, (*Test1* and *Test2*).

Hence $S = 2$

2. The number of common tests (2) relative to the total number of different tests (7) is 2/7. In order for this value to be an integer score in the range 0-9, then the ratio is multiplied by 9 and the nearest integer is used as P.

   Hence $$P = \frac{2}{7} \times 9 \approx 3$$

3. Given that the significances of tests 1-3 is 2 and that for each of the others is 4 then

   $$G = \frac{4}{22} \times 9 \approx 2$$

   Note that the multiplication by 9 is for the same reason as that mentioned in 2.

4. The frequency count is that of the test pattern, say 3.

The ***matching scores*** (S, P, G, F) are taken into account when choosing the best matches from all the suspect faults produced as a result of searching the knowledge-base.

5.2.3 The Development of a Rule-Base for Evaluating Match Quality

Searching the knowledge-base, see section 5.2.2, results in a number of possible suspects faults each associated with its matching scores. In order to find the most suspected component a method is required of evaluating these scores and comparing them with each other.

This section describes the attempts which lead to the development of a rule-base for comparing the scores of the different suspects and choosing the most suspected fault.

5.2.3.1 Initial Investigations

Ordering the matches is possible by presenting all the matches to a *sorter* which will re-order the matches in descending order of the quality of the match described by a given criterion. The criterion, presented to the sorter, should cover all possible combinations of two matches which could be presented and an indication of the best match by either confirming or rejecting the order in which the suspects are positioned in the suspect list presented to the sorter prior to the sorting procedure.

Initially, a simple criterion is used and is as follows :

> *Given two patterns (pattern$_A$ pattern$_B$); then pattern$_A$ (the best match) is the one which is exactly matched or the pattern with the largest scores, otherwise the two patterns must be reversed in order.*

If neither of the two patterns are exact, then they are sorted according to their score total. Tables 5-1 (a) and (b) are examples to illustrate the use of the above criterion in discriminating a best match. Pattern$_A$, in table 5-1a, is an exact match (P = 9) hence the order of the two patterns is not altered in this case.

	S	*P*	*G*	*F*
Pattern$_A$	2	*9*	*5*	*6*
Pattern$_B$	2	*6*	*7*	*8*

(a)

	S	*P*	*G*	*F*
Pattern$_A$	*4*	*7*	*5*	*8*
Pattern$_B$	*4*	*6*	*9*	*6*

(b)

Table 5-1: Illustration of initial criterion.

While in the example of table 5-1b, neither of the two patterns is an exact match, and hence their score totals are compared which leads to reversing the order of the two patterns since the scores of pattern$_B$ total higher than those of pattern$_A$ (25 > 24).

The above criterion is crude and it does not offer sufficient control over the choice of the match, since it was noted by basic inspection of some matching results. In addition, there are cases where comparing the total does not produce a correct match. As an example, consider the two test patterns in Table 5-2 and their corresponding matching scores:

	S	*P*	*G*	*F*
Pattern$_A$	*1*	*1*	*1*	*6*
Pattern$_B$	2	*1*	*1*	*1*

Table 5-2: A restriction in initial criterion.

The total for pattern$_A$ is 9 while the total for pattern$_B$ is 5. According to the previously described criterion; pattern$_A$ is a better choice. However, as it was noted with different matching results, the difference in *S* is more significant than a difference in *F* since the match is not an exact one any way. More tests in the intersection is a better indication to the match as long as the increase in the other scores is not 'very' large. The previous example is one of many cases where the initial criterion failed in making a correct decision.

Furthermore, there were cases where the amount of difference between corresponding scores in different patterns was significant. e.g. provided the first three scores in the first pattern are larger than the corresponding ones in the other pattern, then the second pattern will only be chosen if its fourth score, *F*, was larger by about 25. This example is another case where the initial criterion failed to make a correct decision, this lead to the development of an alternative criterion which offers more control over the choice of the best match. The next section describes this criterion in detail.

5.2.3.2 Generating a Rule-Table

The comparison between the scores of the two matches (pattern$_A$ pattern$_B$) could be performed using the predicates >, <, and =. Hence an alternative matching criterion could be developed, to that of the previous section, which involve the use of these predicates in expressions (or rules) which describe the required relationships between the matches. Examples of such rules are the following:

IF $[S_A>S_B \; \& \; P_A>P_B \; \& \; G_A>G_B \; \& \; F_A>F_B]$ *THEN* *OK*

and *IF* $[S_A<S_B \; \& \; P_A<P_B \; \& \; G_A<G_B \; \& \; F_A<F_B]$ *THEN* *~OK*

. . . .

OK and ~OK are sorting predicates which either indicate that the current order is *OK*, hence no change is required, or that the order is not OK, hence the matches will acquire a new order with respect to each other which is the reverse of their previous order.

By developing a matching criterion which is made up of rules, such as those above, more flexibility is introduced into the decision making process. This is because separate rules could be used for individual cases. This procedure could recover clearly distinguishable cases as well as those which require more careful examination, such as the case in table 5-2 mentioned earlier. For this reason such a criterion offers more control over the matching outcome than the one provided by the initial criteria of section 5.2.3.1, which treated all cases at a similar level of ambiguity.

	S	*P*	*G*	*F*
Pattern$_A$	*1*	*3*	*5*	*5*
Pattern$_B$	*1*	*3*	*3*	*7*

(a)

	S	*P*	*G*	*F*
Pattern$_A$	*1*	*3*	*5*	*5*
Pattern$_B$	*1*	*3*	*3*	*15*

(b)

Table 5-3: An example of a complex rule.

The rules provided at the start of this section are examples of those aimed for obvious cases, however, in order to understand the construction of those developed for more difficult cases consider the matching pairs of the example in tables 5-3 (a and b).

Both examples represent the same relationship among the corresponding scores of the patterns A and B except in the amount that F_B exceeds F_A. The amount F_B-F_A is larger in the case of example (b). By considering the different examples of the above cases in the data it was clear that in the cases where F_B-F_A is relatively small (< approx.5), as in (a), then pattern$_A$ is a better match, whereas if the difference is larger, as in (b), then pattern$_B$ is a better match. An example of a rule to recover examples of this case is as shown below :

IF $[S_A=S_B \,\&\, P_A=P_B \,\&\, G_A>G_B \,\&\, F_A<F_B]$ ***THEN***

IF $[(F_B\text{-}F_A) < 5]$ ***THEN OK***

The above rule is in the form

$$\underbrace{\textbf{\textit{IF}} \quad [CONDITION_{LEVEL1}] \quad \textbf{\textit{THEN}} \quad \underbrace{\textit{IF} \quad [CONDITION_{LEVEL2}] \quad \textit{THEN} \quad [ACTION]}_{\textit{Secondary rule}}}_{\textit{Main rule}}$$

The above form could be considered as a *main rule* with a *secondary rule* within it and is a general form for *all* the matching rules. The rules mentioned earlier in this section are special case of the main rule form where *CONDITION$_{LEVEL2}$* does not exist.

If we consider the total number of possible relationships among scores of two suspect patterns, then only a finite number exists. Since 4 such scores exist and 3 possibilities exist for comparing corresponding scores (=, <, >), therefore 3^4 possibilities exist for CONDITION$_{LEVEL1}$. This provided the possibility of storing all the main rules in a look-up-table, which offers the advantage of fast access. *A single main rule is represented as one entry in the look-up-table, with its $CONDITION_{LEVEL1}$ being the table indices and the secondary rules being the actual content of the entry.* Such a table with its rule contents is termed as the ***Rule-Table***.

Choosing the appropriate secondary rules for each rule-table entry is performed *manually* by considering the examples of each main rule in the data and their corresponding matching scores when processed by the system, and then developing the appropriate secondary rule which will lead to the recovery of the majority of the examples. This process is referred to as *manual tuning* of the rule-table. A relatively small number of secondary rules resulted from the tuning process since a number of different main rules shared the same secondary rule.

Table 5-4 illustrates a rule-table for a matching system which generates two scores S_1 and S_2. It describes all the possible relationships which could exist between the two scores of any two successive faults and an indication to the system of what should be done (to the *order* of the two successive faults), i.e, the secondary rule. The table is usually constructed by the system designer in terms of a limited number of secondary rules (OK, OKif,...etc.) selected and implemented by the designer himself. The names are chosen for simplicity and illustration purposes.

		f_{S2}		
		>	<	=
f_{S1}	>	***OK***	***OKif***	***OK***
	<	***~OKif***	***~OK***	***~OK***
	=	***OK***	***~OK***	***Both***

Given Faults F_i and F_{i+1} in the order ($F_i F_{i+1}$) in the Suspect List (i.e, F_i preceding F_{i+1}, then :

f_{S1} **Represents the result of comparing S_1 in F_i with S_1 in F_{i+1}.**

f_{S2} **Represents the result of comparing S_2 in F_i with S_2 in F_{i+1}.**

Table 5-4: A possible configuration for a manually tuned rule-table.

For example, if a system using the above rule-table encounters two adjacent faults F_1 and F_2 where $S_{1F1} > S_{1F2}$ and $S_{2F1} > S_{2F2}$ (the first table entry) then when the corresponding table location is referenced the system will realize that no change is required in the order of the two faults and hence the secondary rule *OK* indicates no change to the order of the two faults.

~OK indicates that the order of the two faults described by the relationship of their scores is *not OK* and hence change in their order must be performed. *OKif* indicates no change provided an additional condition is satisfied (see $\text{CONDITION}_{\text{LEVEL2}}$ mentioned earlier) e.g, S_{2F1}-S_{2F2} > 5. *~OKif* is the opposite of *OKif.*

In the actual rule-table constructed for the system, see appendix B, four scores are used and hence a multidimensional look-up-table is used with secondary rules which operate on the scores in a manner similar to the above. As can be deduced from the table, more rules exist and these are more complex in the number of different scores examined by $\text{CONDITION}_{\text{LEVEL2}}$ in their secondary rules.

5.2.4 Heuristic Rules

5.2.4.1 Observations Based on the Data

This section briefly considers some of the heuristics and observations, made by the diagnostic technicians in the company from their experience with the board failures, in aiding them to locate faults. These are in the form of notes obtained by interviewing a number of technicians and studying their notebooks.

Some of these heuristics are as follows:

1. According to the technicians, the likelihood of various failures occurring is as follows (most likely first):
   500s, 40s, 31, Leg not through for a particular component....etc.

2. If a single test (or any number of tests) fails for all of the individuals with the exception of one then the faulty component is most likely to be in the only non-failed individual. The faulty component is usually isolated by using technicians experience with associating the previous failures of the test (tests) with a component (group of components).

3. When a failure slip shows different failed tests in different individuals, then the tests failed in the *last* individual in the slip are used in identifying the component type (usually as in 2.) within this individual.

4. Failures of tests from group (Test 0) usually indicates the failure of the Processor (31) and in some cases the failure of Fuse 540.

5. Failures of tests from group (Test 6) usually indicates the failure of one of the following, listed in the order of decending likelihood: Bridge rectifiers (70s), Thyristors (490s), Resistor networks (250s and 260s), Relays (500s and 510s), and Slics (40s).
6. Distinct patterns (a pattern is a sequence of failed tests) fail regularly due to a fault in specific components. Some of the examples of these in the data are:

   i. A pattern of 8 tests in 400s.

   ii. A pattern of 14 tests in 70s.

   iii. A pattern of 20 tests in 40s.

   .

   .

Due to the common failures of the component groups 40 and 50, the following two points are specific to these;

7. The following tests usually indicate the failure of 40s:

Longitude noise D->A
IMP.UNB.(Longit) D->A
False Loopimpdata
No Loopimpdata
Curr. Feed active Slic
No offhook
No onhook

The above tests usually fail alone, repeated a few times (2-3) under different conditions.

8. The following tests usually indicate the failure of 50s:

False Bdata
False Rdata

. .

. .

Overall Loss A->D
ToT.Dist.(Noise) A->D
Linearity (Noise) A->D

The above tests are an indication of the failure of the corresponding component whether the report indicates a multiple individual failure or a single one.

5.2.4.2 Heuristics Used During Diagnosis

Some of the heuristics used by the diagnostic technicians involve identifying a single test or group of tests which indicate the failure of a certain device or group of devices. Examples of this type of heuristic are 4,5,6,7, and 8 in section 5.2.4.1.

The identification of faults based on the comparison of tests in new reports and previously encountered ones is the basic principal behind matching. For this reason, this type of heuristics are incorporated in the matching procedure. In practice, the system can identify such cases after being presented with 1-2 example reports of the case.

The heuristics of 2 and 3 deal with cases of multiple individual failure and are incorporated to the system (in Lisp code) since they cannot be recovered by matching alone. These heuristics are useful in recognizing multiple individual cases and choosing the tests of the appropriate individual as a match-pattern to be utilized in matching, see section 5.2.2

The heuristic in 1 need not be incorporated into the system since it represents the most common faults according to the diagnostic technicians over a certain period of time. Such information can become out of date with changes in the manufacturing procedure. The system has access to such information since it stores statistical information about the individual tests and test-patterns for each fault and updates any changes dynamically.

5.3 System Learning

When the correct fault is entered by the diagnostic technician, the knowledge-base is searched for that fault. If it was not found then a new fault entry will be generated with the corresponding fault number and the tests in the report as the test pattern which will be given a frequency count of 1 since it is the first pattern seen for the given fault. However, if the fault was found then the test patterns of the fault are scanned. If the pattern in the report was found then only the frequency count is increased however, if the pattern is not found, then it is added to the list of test patterns belonging to the fault with a frequency count of 1.

Each test in the report with the correct fault are used to update the table in the test significance knowledge-base in a manner similar to above.

5.4 A Walk Through the System

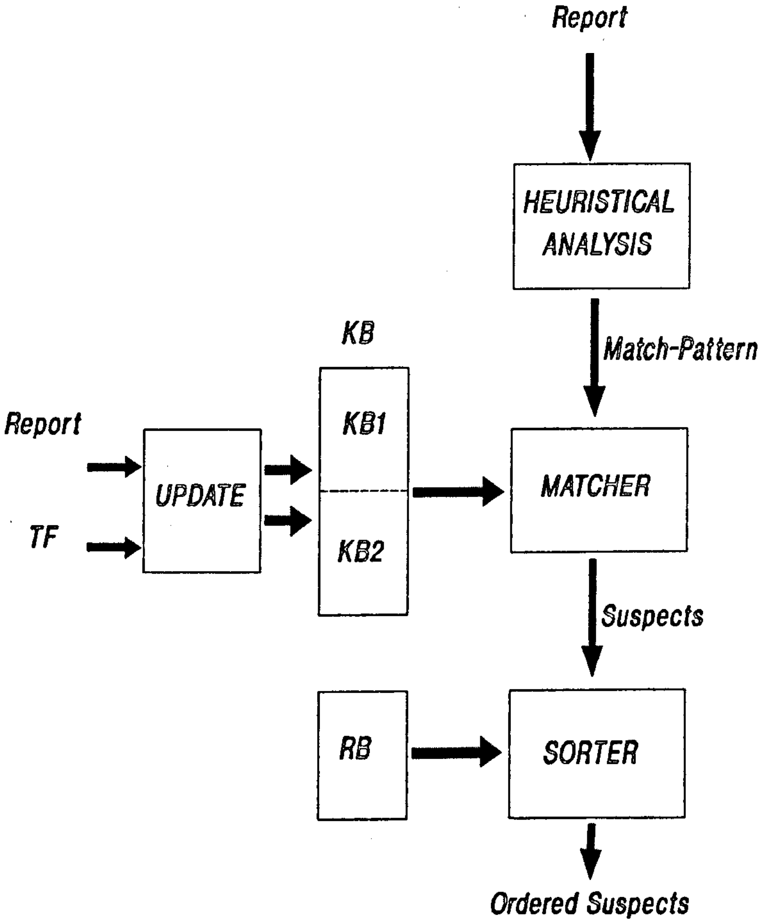


Figure 5-5: The system in operation.

To perform a diagnosis, the system must be presented with a failure report. The first operation to be performed on the report is to recognize whether the failure is in a single individual or in more than one. This is performed using some of the heuristics incorporated into the system, see section 5.2.4.2. In either of the two cases an optimal number of tests and a suspect module is indicated. The tests are the *match-pattern*, which is in turn presented to the matcher. By searching both parts of the knowledge-base, a number of *suspects* are produced. These suspects are later sorted according to their matching scores, see sections 5.2.2 and 5.2.3, and are the final outcome of the system. This procedure is illustrated in figure 5-5.

For the system to be kept up to date and in order to improve its performance, it must be updated by the correct cause of failure, TF, at a later stage. This procedure, which is described in section 5.3, involves modifying both fault entry and test pattern knowledge-bases with TF associated with the failed tests in the report.

During conditioning the system in the company some more modifications were incorporated to ease the task of updating for the technicians by storing diagnosed reports in a queue like structure. In practice, more than one report is diagnosed successively and update is only possible after a period of time which could involve tracing the fault in the circuit, if system diagnosis was not correct.

5.5 System Evaluation Environment

A considerable amount of time and effort was put into examining the performance of the system by adjusting the different attributes of the main system elements, such as choosing an efficient method of information storage in the KB, tuning the rule-base, manipulating the heuristics. All of this required an environment in which the system can be tested efficiently which requires feeding the system with reports which are as random as the ones met in its actual diagnosis environment (given the quality and the quantity we currently have access to). In addition, the environment should provide relatively easy access to the different parameters of the system during testing and allow controlling the testing procedure with a number of options. An evaluation environment implemented around the system is illustrated in figure 5-6 and will be the subject of the rest of this section.

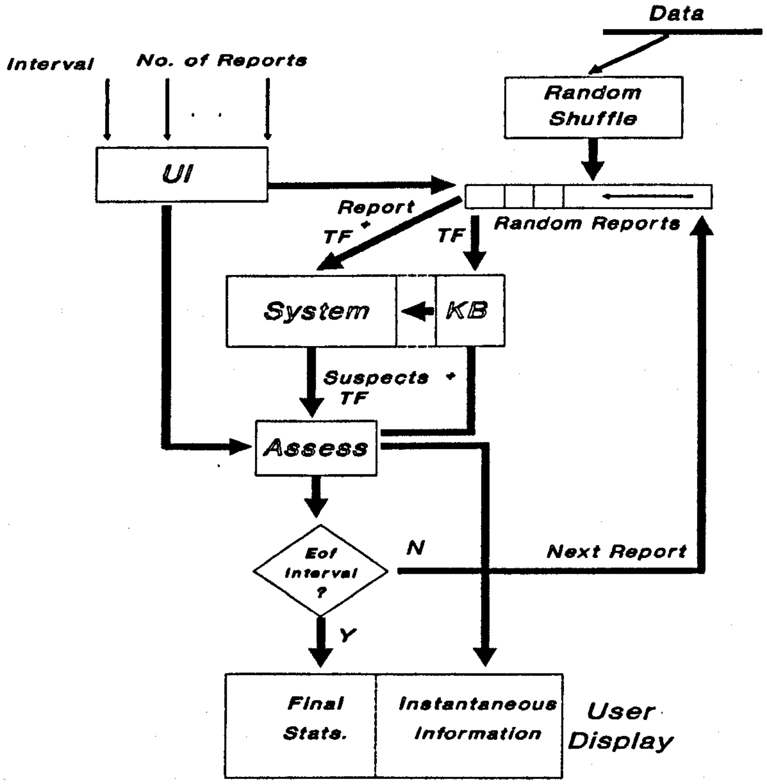


Figure 5-6: The debugging procedure.

All the reports we had access to are stored in a data-base which could easily be accessed by the system. The reports are stored such that each failure report is accompanied by a note of the actual faulty component (True Fault, ***TF***) which caused the failure, this is in order to enable comparison with the system's own prediction. Each time the system is tested the user has the option of *shuffling* the reports so that they could be presented to the system in a different random order in different test occasions. The shuffled reports are placed in a buffer ready for use.

The user interface of the system (***UI***) enables the user to specify a number of options which include the following

1. The number of reports used to build the knowledge-base prior to diagnosis and the number of reports to be assessed.
2. The length of the interval (in reports) at which a total assessment of the diagnosed reports should be made.
3. Display a number of system parameters at regular intervals during the diagnosis procedure such as related portions of the knowledge-base, match-pattern, un-processed report, ...

When the system evaluation commences, the first report of the randomly shuffled reports are presented to the system. The system produces a number of suspects ordered in a decreasing order of likelihood of being the faulty component. Each suspect fault is accompanied with its matching scores.

Next, the suspects are presented to the assessment routine which tries to locate the true faulty component in the list of suspects. If it is located then the routine will record the position in the suspect list at which it was identified, otherwise the report will be counted as an incorrectly matched unless the suspect list is empty in which case this will be indicated.

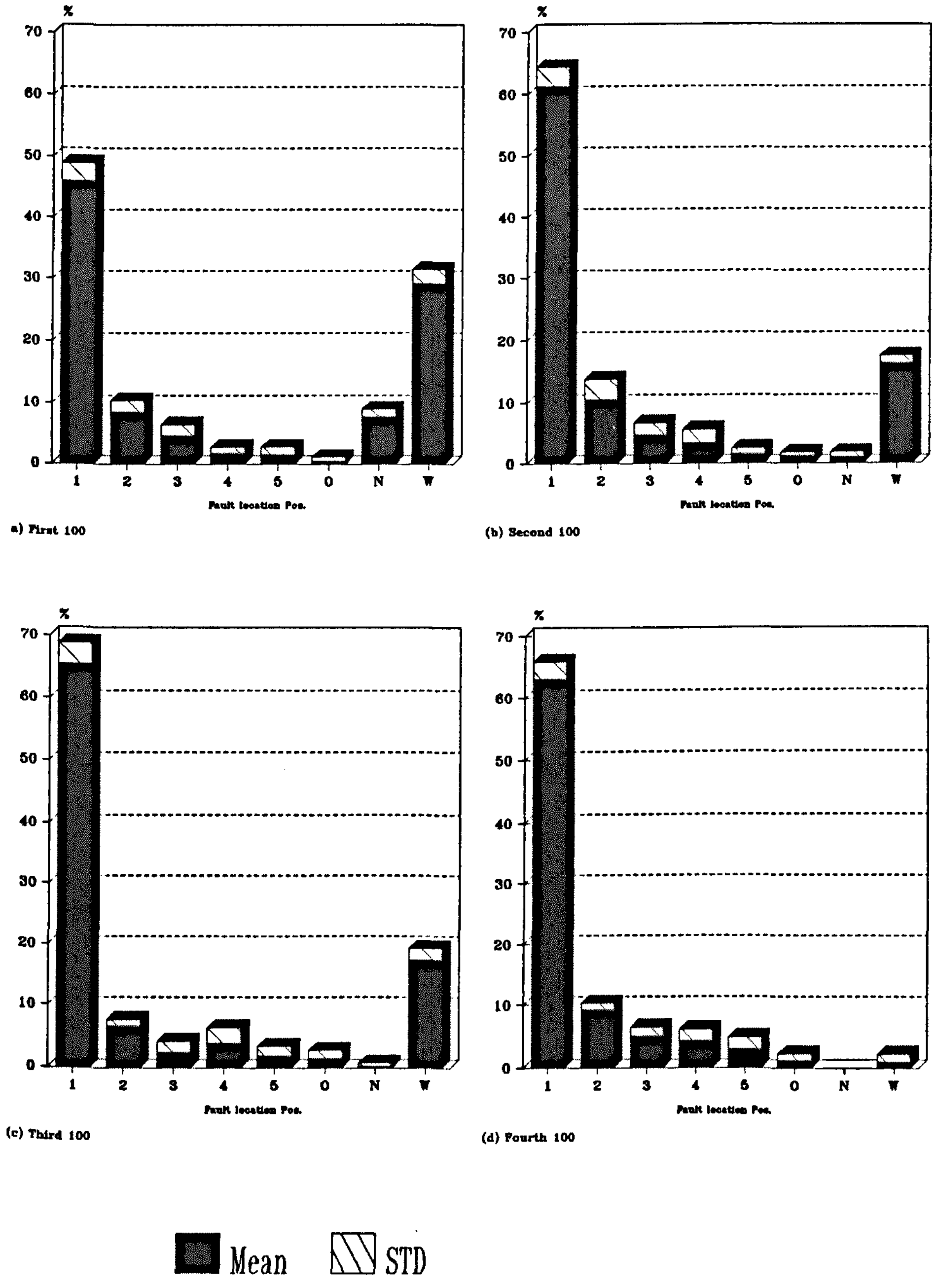
The user is provided with two types of information, the first is instantaneous and is performed after each assessment of a report and the second is performed at regular intervals specified by the user. The instantaneous information include the suspects and their corresponding scores, TF, and any other optional information set by the user using UI. The periodic assessment displays statistical information such as the number of reports identified as the first best match (1st position in suspect list), second best match (2nd position....),...etc. Also, some information about the frequency of different tests failing due to a fault in different components in a bar chart form.

During system assessment process, the updating procedures for the different parts of the system are performed automatically, since this is an important part of maintaining such a system.

The evaluation environment described in this section played a major role in assisting to improve the system's performance such as recognizing the significance of the different matching scores and hence the development and tuning of the rule-table. In addition, it provides an efficient way of monitoring the system by locating specific cases where the system fails in its diagnosis and hence provides a way of *debugging*.

Due to the efficiency of this environment in indicating the system performance, it will continue to be used in most of the investigations mentioned in the rest of this thesis.

5.6 System Performance



a) First 100

(b) Second 100

(c) Third 100

(d) Fourth 100

Figure 5-7: Mean fault identification charts for the system.

The performance of the system, using matching alone in its diagnosis, was assessed by using the most common PCB style manufactured in the company and are illustrated using ***Performance Charts*** of figure 5-7. The charts are developed by presenting different random sequences of 497 reports to the system and assessing the performance after each 100 reports. The performance is assessed by examining the order at which the fault is identified. The orders in the charts are:

x	*Faults identified as the xth most likely fault.* *Where x is one of {1, 2, 3, 4, 5}*
O	*Faults identified beyond 5th position.*
N	*Fault not identified.*
W	*Faults identified incorrectly.*

As the charts indicate, 65% of the reports presented to the system are correctly identified as the best match. If the best three matches are considered, then about 75-80% of the reports are identified correctly.

The above group of reports represent the majority of the failure cases in the data. When analysed, it was clear that the pattern of failed tests in the reports are exact symptoms of the corresponding faulty components (i.e, their exact signature).

The remaining 25-30% of the reports fall into one of the groups described below :

1. Faults identified at positions other than the first best fault. These were found to be mainly due to test patterns common to more than one fault.
2. Unidentified faults; these are usually ones with new test patterns and occur usually at early stages of building the knowledge-base.
3. Incorrectly identified faults; these include errors in the data and odd failures of a certain component.

Initial examination of some of the examples of the above cases, revealed that some classification may be possible according to the pattern of the failure in the report. However, this work will not be considered in this section and will be the subject of the next chapter.

5.7 Conclusions

In this chapter a system was developed modelled on the conduct of a diagnostic technician during the failure diagnosis process. The system performs a diagnosis by matching a failure report with dynamically stored information about previously seen reports in its knowledge-base.

The results obtained by the system are satisfactory since about 65% of the reports presented to the system are correctly identified as the best match. One of the reasons for this performance is that the approach used mimics that of the technicians in analysing a failure report, by using a combination of some heuristic rules with matching and continuously improving with experience. This approach improves on the automatically generated tree approach which is less flexible.

However, some more investigations were necessary in order to identify the cases where improvement is possible and investigate how so. Also, the system includes some heuristic rules which are not confirmed completely and are only based on observations of the technicians. It will be desirable to develop these into a more general ones which could be applied to a greater variety of board styles. Furthermore, it is advantageous to make the system more self-dependent by enabling it to tune its matching rules. All of these are possible options for investigations in the next chapters of the thesis.

In addition, keeping the system up to date involves more complications than the mere addition of new cases and keeping records of test and pattern frequencies. Faults vary with changes in processes such as manufacturing or assembly leading to the disappearance of some faults and possibly the appearance of others. The faults which disappeared might or might not appear again. However, since the information related to such faults remains in the knowledge-base unchanged, they could keep appearing as possible suspects for other faults. For a more complete update procedure such changes must be reflected into the knowledge-base. This point will be carried futher in a later chapter.

CHAPTER SIX
ANALYSIS OF SPECIAL CASES IN THE DATA

6.1 Introduction

The results obtained using the system, described in chapter five, are encouraging since the faults associated with the majority of the reports are identified correctly. As illustrated in section 5.6, about 65-70% of the faults associated with the reports are identified correctly as being the *first* most likely fault while if the three most likely suspect positions are considered (1st, 2nd, 3rd), then about 75-80% are identified correctly.

However, about 20-25% of the faults are not recognized at all (incorrectly matched or no matches are found) with about 5% identified as least likely suspects (such as 4th position, 5th . .etc). This chapter is an attempt to investigate these cases.

The investigation is based on analysing the different ambiguous cases which exist in the data. When analysed, these cases can be classified into groups of report ambiguities. The examples of each group share a common feature in the failure of their tests distinguishing them from other groups and from the majority of the reports in the data. These ambiguities are briefly mentioned when the general problem and the data were first introduced in chapter three, see section 3.3.1, and are also encountered as heuristics followed *blindly*, without knowing the reason, by the technicians in the identification process as in section 5.2.4.

The ambiguities are examined by locating examples of each case in the data. By studying these examples an attempt is made to find ways of recovering some of these cases later in this thesis and finding the reason where recovery is not possible.

Most of the cases mentioned here are ones which involve the failure of more than one individual (module) in the circuit since these constituted the majority of the ambiguous cases, hence most of the ambiguous cases discussed in this section are referred to by the pattern of individual failure which characterises them.

The investigation for each case is described in a separate section, where the different examples in each group are noted for the frequency at which they occur, the individuals failed...etc., and a possible explanation in terms of the circuitry where possible.

This chapter concludes by suggesting some additional work which might lead to improvements in some of the cases and in the systems performance in general.

6.2 Multiple Individual Failure Cases

6.2.1 The Last Individual

Examples of this case include reports in which tests fail in more than one individual, the majority of the examples in the data indicate that the tests of the last individual should be considered in matching with the corresponding individual number to identify the correct faulty component (this could be due to the fact that each test is applied in an ascending order, starting with the lowest numbered individual which is individual 0). This procedure is employed as one of the heuristics of section 5.2.4.

6.2.1.1 An Overview

Examples of this case could fall into one of the following groups :

1. **The *Non-Individual* components**

   These are components that do not repeat in each individual such as the digital control circuitry elements, the fuses,...etc. Since some of these components are connected into all of the individuals and some are connected to many components, their failure leads into the failure of different tests in different individuals (sometimes in all the individuals).

2. **The *Individual* components**

   These are the different components which constitute the circuit of each individual. Table 6-1 summarizes the different examples of this case.

Each entry in the table represents a single failure report by pointing out to the following information in the report in their respective order:

The *reported* faulty individuals.

The *actual* faulty individual.

The faulty component within the individual.

The number of times a similar report has been encountered in the data.

Reported INDs	*Failed IND*	*Component*	*Repetition*
0,1	1	slic	2
0,1	1	test relay	1
6,7	7	slic	1
1{2},3	3	slac	1
0{2},5{4}	0	slac	1
1{2},2	1	thyristor	1
4,0	4	C_{400}	1

Table 6-1 : Examples of Inter-Individual components

The numbers between { } indicate the number of times the corresponding individual failed.

By consideration of table 6-1, the following are noted :

1. The failure of a component in individual 1 causes the failure of some tests in both the faulty individual and the previous individual. The failed tests are not identical, however, in most cases both groups of tests point to a similar component in both individuals. A possible explanation for this case could be that the failure of the component in individual 1 causes the *CUT OFF* of the components of individual 0 from the common line which connects all individuals. This case is illustrated in figure 6-1

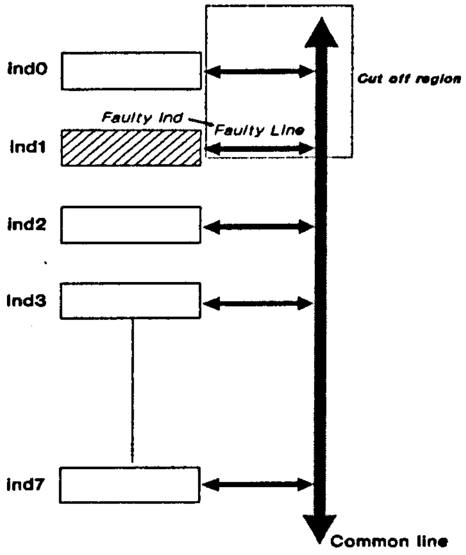


Figure 6-1: Illustration of a cut off case.

2. In many cases the failure of one component causes the failure of another in a different individual without causing the failure of others. Since each individual functions independently from the others, and the only connection between different individuals is the common lines which connect some of their components. This case cannot be explained as a cut off as in 1, however, a possible explanation could be that the tester tries to establish a communication between two individuals in the circuit since in normal operation of telephone circuitry it is possible to have a telephone link between any two individuals. The same tests could be applied while testing the communication of two individuals, the only difference is that the caller circuit is closer which could effect a parameter such as load. The next section will consider these cases more closely. Figure 6-2 illustrates a case in which communication is established by the ATE between individuals 0 and 3.

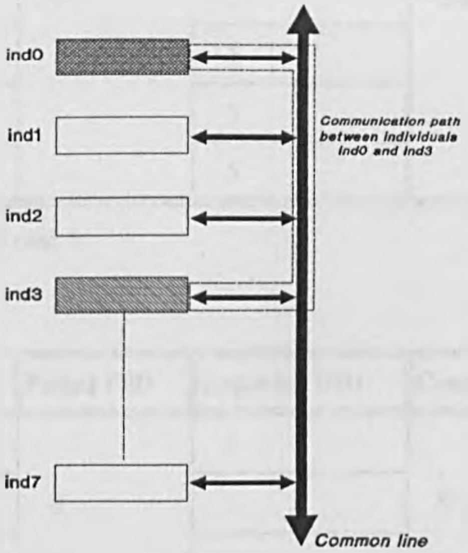


Figure 6-2: Communication of individuals established by the ATE.

6.2.1.2 A More Comprehensive Examination of the Cases

This section considers some of the cases of table 6-1 individually. By analysing the different cases in the table, it can be seen that the different reports indicate a failure in two individuals which are adjacent or apart. The next sections consider the cases of each of these.

Each section considers a number of cases (report). Each tabulated case points to the following information in the report :

The test name and (its group number).

The actual failed individual.

The reported failed individuals.

The faulty component within individual.

6.2.1.2.1 Separate Individuals

The cases considered are the ones which are caused by the failure of two random individuals which could not be explained by CUT OFF.

Case 1

Tests ***(Test No)***	**Failed IND**	**Reported IND**	**Component**
Ring relay *(1)*	0	0	Slac
No offhook *(1)*		0	
Overall loss a-d *(11)*		5	
Overall polrev a-d *(11)*		5	
Overall polrev d-a *(11)*		5	
Overall loss d-a *(11)*		5	

Table 6-2 : Details of the report in case 1

Case 2

Tests ***(Test No)***	**Failed IND**	**Reported IND**	**Component**
No offhook *(1)*	3	1	Slac
No offhook *(1)*		1	
idle ch noise a-d *(14)*		3	

Table 6-3 : Details of the report in case 2

Case 3

Tests ***(Test No)***	**Failed IND**	**Reported IND**	**Component**
overall loss a-d *(11)*	4	4	$C_{slic-i/p}$
tot dist noise a-d *(15)*		0	

Table 6-4 : Details of the report in case 3

Discussion of the Cases

The reports had the following characteristics:

1. The failures in the faulty individual are due to tests which indicate a failure in one of the functions of the Slac.
2. The failed component in the first two cases is the **Slac**, however, the third case failed due to a **capacitor** between two Slic inputs (the purpose of this capacitor could be related to one of the tasks which the Slac carries out via the Slic and hence the failure of the tests).
3. In the first two cases the tests failed in the first individual indicated a fault in a Ring-Relay (or a Slac). However, the failed tests in the last individual (which is the faulty individual in this case) indicate a fault in a Slac.
   In the third case, however, all the failed tests indicate the failure of the Slac.
4. In the third case the higher numbered individual fails before the lower numbered one. This is possible since each test is applied in a top-down manner (i.e, in the order 0-7).

6.2.1.2.2 Adjacent Individuals

Case 4

Tests ***(Test No)***	**Failed IND**	**Reported IND**	**Component**
OVP b-wire thyristor *(6)*	7	6	Slic
OVP a-wire thyristor *(6)*		7	

Table 6-5 : Details of the report in case 4

Case 5

Tests ***(Test No)***	**Failed IND**	**Reported IND**	**Component**
OVP b-wire thyristor *(6)*	1	1	Thyristor
OVP a-wire thyristor *(6)*		1	
OVP b-wire diodeBridge *(6)*		2	

Table 6-6 : Details of the report in case 5

Discussion of the Cases

By examining the cases, it is clear that the tests failed in both cases indicate a failure in the OVP circuitry with the tests in both faulty and non faulty individuals being of the same type.

Examining the values at which these tests failed in each of the two reports indicate that the value in the faulty individual is out of its limits by a large margin, however, in the non faulty individual the value is only slightly out of the range of its limits.

6.2.2 The Odd Individual Cases

Most of the diagnostic technicians in the company who were interviewed indicated that in some cases a report contains a certain test(s) which fail in all of the eight individuals except one. Usually the cause of such failures were found to be the component in the individual which passed the test. Suspecting the individual which is not reported in the failure report is justified since the failure of a single module is more likely than the failure of all of the modules in the PCB circuitry.

Examples of this case, which were encountered in the data, are tabulated in table 6-7. The information in the table is similar to that of table 6-1.

Reported INDs	*Failed IND*	*Component*	*Repetition*
0-6	7	Test Relay	1
1-7	0	slic	2
0,1,3,4,5,6,7	2	slic	1
0,1,2,4,5,6,7	3	slic	1
0,1,2,3,5,6,7	4	slic	1
0,1,2,3,4,5,7	6	slic	1

Table 6-7: Examples of the odd individual case.

Notes

1. It can be seen that examples of this case occur in only two types of components (Test Relays and Slics). Both types of components have a common line which connects all the components of their type in all the individuals.
2. The failed tests in the slics are all of the same type in all of the cases.

3. In some cases in the slics, a single test failure is reported in the failed individual. The test has the same general task as the test(s) failed in the reported individuals, but is different in the condition under which it is applied.

6.2.2.1 Possible explanations

A possible explanation to the failure of the Test Relay in individual 7 could be due to a weak (or an open) connection in the common ***test*** inputs line at the point of contact with individual 7. If this situation occurred then the input could reach only individual 7 and not the others.

A possible explanation for the other cases, which mainly involve the Slic, could be due to sharing a common line among all individuals. Information is sent to one individual at a time, at this time the rest of the individuals must block their input to the common line so that information is only moved to the required individual. The blocking mechanism could be one of the functions of the Slic since it is at the receiving end of the common line. Hence, when the faulty individual is tested no error is reported since all the other individuals can operate their blocking mechanism, however, an error is reported when each of the other individuals are tested since in each case the single faulty individual fails from blocking itself.

6.2.3 The Wrong-Test Cases

A relatively large number of reports in the data contained failed tests (in one individual) which have the task of testing a certain component in the individual, however, the cause of the failure is diagnosed to be due to a different component in the same individual. These cases were not suspected to be a fault finder error (although some of them could be) because of the repetition of some of them in the separate groups of data received. The examples of this case could be further divided into two separate groups. These will be discussed in the following two sections

6.2.3.1 Group 1

The reports in these examples contain a single test or a group of tests which test a single component and no other tests exist. Table 6-8 illustrates the examples found in the data. For each report the following information are indicated in the table in their respective order:

The test name(s) (its/their group number)

The actual failed component.

The faulty component/circuitry to which the test(s) failure point to.

Number of times a similar report is encountered in the data.

The table contains the analysis of the most common components in the data (Slic, Slac, Test Relays, Ring Relays). Further analysis will be carried out with the less common components in the future if a significant number of such cases exist. The Test field in the table summarizes the test(s) and does not contain any details (such as conditions) this is to reduce the size of the table and make it readable.

No	*Test (Test No)*	*Failed Comp*	*Relevant Comp*	*Repetition*
1	OVP-Tests (6)	Ring relay	OVP circuitry	11
2	OVP-Tests (6)	Test relay	OVP circuitry	4
3	Ring relay (1)	Test relay	Ring relay	4
4	Linearity (13)	Slic	Slac	1
5	Longitud noise (14)	Slic	Slac	33
6	Tot distortion noise (15)	Slic	Slac	4
7	Linearity (13), Idle ch noise (14), Tot distortion noise (15)	Slic	Slac	1
8	Idle ch noise (14), Tot distortion noise (15)	Slic	Slac	1
9	OVP-Tests (6)	Slic	OVP circuitry	3
10	Unbalance (17)	Slac	Slic	1
11	Current feed (1)	Slac	Slic	1
12	Current feed (1), Test relay (1)	Slac	Slic and/or Test relay	2
13	OVP-Tests (6)	Slac	OVP circuitry	2

Table 6-8 : The wrong test cases

Possible explanations

[1-3] :

OVP tests are applied through the common test inputs which feeds into the Test relay of each individual. Within a single individual a test signal passes through a path which leads into the slic passing through connections to Ring relays, OVP circuitry.

Since both Ring relay and Test relay are in the OVP test signal path, then a fault in any of them will cause the failure of the OVP test. The nature of the output response to the test input will depend on the type of the failure in any of the components.

[4-8] :

The tests involved in these reports are related to A-D and D-A conversions which are mainly carried out by the Slac.

A speech signal, which is output from the individual, is processed by the Slac and then passed to the Slic which provides the necessary conditions for the signal to be sent to the other end of the communication line. Therefore, any fault in the Slic could cause the failure of the tests which are testing the functions of the Slac.

[9]

One of the tasks of the Slic is controlling the operation of the Ring Relay which in turn lies in the path of the OVP test signal, hence, a fault in the slic (in the part of the slic which controls the relay) will cause the failure of the OVP test. The cause of the failure could be The Slic itself since it is connected directly to the OVP test signal path (see the explanation of [1-3]).

[10-12]

An incoming speech signal passes through the Slic and then to the Slac. Therefore, a failure in the Slac could cause a failure in the tests which test the ability of the Slic in handling the in coming speech (the nature of the failed indicate problems in handling the signal).

[13]

A faulty Slac will effect the signals entering the Slic which will in turn cause the failure of the OVP test (see the explanation of [9]).

Note

The tests corresponding to the failed component do not appear in the above cases. This could be due to tests being **functional**. Since functional tests cannot test all the states and conditions of the components, but select only a few of them.

6.2.3.2 Group 2

The reports in this group contain more than one type of test; tests which are not relevant to the faulty component and others which are relevant to the faulty component. In most cases, the faulty component appeared to be the reason for the failure of the other tests for connectivity reasons. These reports were not analysed as in the previous case because of the relatively large number of tests involved in each report which will produce tables which are difficult to read.

6.2.4 The Wrong-Individual Cases

Some of the reports in the data contain test(s) which indicate the failure of the faulty component however the test failure(s) occurred in a wrong individual. It must be noted that in these cases no other individual failure is reported. The relevant reports are tabulated in table 6-9 which contains similar information about each report as table 6-1.

Reported IND	*Failed IND*	*Component*	*Repetition*
5	0	slic	1
6{20}	0	slic	1
2{4}	1	slic	1
3{23}	5	slic	1
0	6	slic	1
5	6	slic	1
4	5	slac	1
7	0	rectifier	1
1	3	rectifier	1
7	0	Test Relay	2
6	7	Resistor network	1
5	6	Thyristor	1

Table 6-9 : Wrong individual cases in the data

It is difficult to explain this case since any two individuals function independently from each other. However, failures could occur with components which are on a common line across all individuals. But such failures cause the failure of more than one individual since all the individuals are on the same line and a fault in one individual will effect the rest.

An explanation could be given to the above cases. The explanation uses some assumptions which are possible considering the large number of failure types that could occur in analogue circuits. The explanation assumes that the failure in the faulty component is such that the functional test did not manifest the state in which the failure occurred, hence, the component passes the test. For the component in the second individual, a fault could exist which is not significant in the normal operation of the circuit such as a resistor being slightly higher or lower than its nominal value (or any other component). Due to the fault in the component in the previous individual this slight deviation in value could become significant and could cause the failure of the test for the individual. This is possible since the individuals are connected by a common line.

This explanation could seem a very unlikely case, however, considering the very low number of faulty boards and the low number of such cases within these faulty boards then this case is possible.

This case could be explained by the argument in section 6.2.1 which points out to the tester establishing communication between different individuals of one PCB .

6.3 Conclusion

This chapter detailed the investigations which were carried out on groups of failure reports in the data. The reports of each group share certain aspects which characterize their failure.

A number of these groups were noted by the diagnostic technicians and were made into heuristics in order to recognize similar examples and later treat them. Examples of such cases mainly involve reports which indicate the failure of multiple individuals such as the odd individual cases, described in section 6.2.2, and some of the cases of section 6.2.1.

Other cases investigated include groups of reports which do not belong to the ones mentioned above and show inconsistency with the majority of the reports in the data.

Each of the cases above are investigated by analysing their constituent reports. The analysis aimed to explain the reason for the occurrence of such cases in terms of the electronic behaviour of the circuit. Furthermore, the analysis aims to find ways of generalizing the heuristics used by the technicians.

The system developed in chapter five treated most cases of multiple individual failure (all with the exception of the odd individual cases) identically by assuming that the failed individual is the last individual in the report. Although, this treatment of multiple individual failure cases was followed by the technicians in the company and it could be applied successfully to relatively large number of such cases (approx. 40-45%). However, the investigations in this chapter revealed that a number of other cases exist which include reports involving multiple individual failure but requiring to be treated differently. These cases are as follow:

1. Reports in which all the individuals fail. In order to treat this case some knowledge of the circuit must be utilized in order to trace the device (or devices) which are causing the failure of all the individuals. Such action is justified since the failure of one or two components is more likely than the failure of all of the individuals.
2. Reports involving the failure of two arbitrary individuals. Investigations of this chapter revealed that the most likely cause of such failures could be related to the test strategy incorporated in the test program which in turn selects two random individuals for test. Since no detailed information is available about the test strategy, such information could not be incorporated into the system. However, the investigations revealed that the choice of the highest individual (rather than the last individual) in the report is a better choice since it recovers some additional failure cases in this group of reports.

From all the above, this investigation indicates the need for using circuit information in order to recover failure cases which involve multiple individual failure. The use of circuit information could also aid in generalizing some of the heuristics such as the odd individual heuristic such that it could be applied to any circuit. This will assist in developing the system to deal with multiple PCB styles. Furthermore, the use of circuit information could be exploited in investigating the cases where the use of matching alone (as in the system described in chapter five) could not distinguish. This point will be carried on further in the next chapter.

It must be noted that the number of ambiguities which exist in the data are very small, relative to the size of the data and the majority cases identified by the system, and they occur less frequently than the *frequent* majority cases. The figures associated above with faults wrongly identified or indicated as least likely include cases which could not be recovered at all for reasons ambiguous to us (mistake by technician,.. .etc). Furthermore, it is noted that within this relatively small figure a large number of ambiguous groups exist, all of which have small number of examples.

The investigations of this chapter might not represent some of the cases accurately due to lack of sufficient examples for some of the cases mentioned. The number of examples for all of the cases described in this chapter represent a small proportion of the total data and the frequency of occurrence for such cases in the company is much less than the majority of the data.

Although some of these cases contain a minimum of 1-2 examples, interviews with different technicians and test engineers confirmed that these are regular cases which occurred before and during commencing with data collection, however, with a much lower frequency and hence are included in this analysis.

CHAPTER SEVEN
THE USE OF CIRCUIT CONNECTIVITY INFORMATION

7.1 Introduction

Chapter six illustrated the need for the use of circuit knowledge to improve the performance of the system with regard to certain aspects, see section 6.3. Such a need arose especially in considering cases of failure reports in which the failure of a number of modules are reported. In addition, the same section suggested the need for utilizing circuit knowledge for cases which cannot be distinguished by matching alone. Such cases include failure reports which are common to more than one fault. This chapter investigates the use of circuit knowledge in an attempt to isolate the failure cases mentioned above.

Due to the limited information available to us about the circuit and the tests applied to it by the ATEs (e.g., the exact purpose of each test, the way in which its applied,....etc), only partial use of the circuit information is possible. The only information provided about the PCB is its circuit diagram and a brief description of some of its components.

The most appropriate knowledge about the circuit which could be used in the system is ***connectivity*** without regard to the direction of the signal in the connections, since these are not known due to the restrictions imposed on us by the lack of some circuit and test information.

For connectivity information to be used successfully by the system in the diagnosis procedure, additional software modules are required to be added to the system. The main requirement for such modules is to perform the following :

1. Incorporating connectivity information to the system, i.e. introducing and putting this information into a form which could be utilized by the system.
2. Manipulating this information in order to improve system diagnosis.

This chapter describes the work involved in incorporating the above into the original system and investigating system performance in response to these modifications. Hence, the chapter starts by describing the role of a circuit compiler in introducing connectivity information to the various parts of the system (see chapter five). Next, the chapter describes the use of such information by a heuristic technique developed to deal with cases which matching fails to recover (section 7.3). Later, the chapter details another technique, which also uses connectivity information, developed to deal with cases of multiple module failure (section 7.4).

7.2 A Basic Circuit Compiler

The need for a circuit compiler arises for the purpose of presenting circuit information to the system so that it can be utilized by the relevant elements of the system.

In order to make the system usable for other than system designers, the input circuit description to the system must be in a form which could be reasonably understood/developed by the user. Such a form might not be adequate for the use of the system directly and may require additional processing. For this reason a compiler is required in order to transform the circuit description entered by the user to a form which can be easily manipulated by the system.

A circuit compiler is a necessary tool for a system such as the one which is the subject of this research since it increases the independence of the system in that it assists in the presentation and change of information about different types of circuitry to the system. All this will be possible without the need for altering the actual code as is the case in the heuristics used currently in the system, described in chapter five.

Since the restricted information provided about the circuit allows us only to use connectivity with no significance for inputs and outputs, this is one reason for developing an independent compiler since most of the facilities available in most traditional circuit compilers cannot be used in this system and are hence unnecessary.

Figure 7-1, illustrates the general use of the compiler in association with the other parts of the system in order to improve diagnosis. More elaboration will be made of some of the features in this figure later in this section.

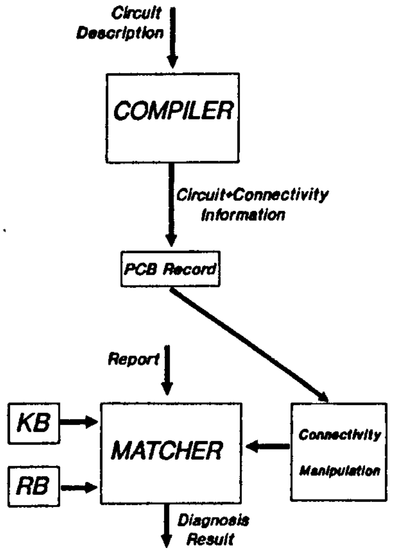


Figure 7-1: The use of the compiler within the system.

Due to the significant role of the compiler in allowing connectivity to be incorporated into the system, this section briefly describes the developed compiler and its compilation mechanism which eases the task of manipulating connectivity.

7.2.1 Basic Circuit Representation Units

The compiler recognizes the following elements, in a circuit description provided by the user:

1. **Component**

   This is the most basic unit in the circuit for which no sub-unit exists.

2. **Module**

   A module represents a group of components which repeat frequently in the main circuit and hence are incorporated into a module structure for the users convenience.

   A module could either be *incremental*, where the component position numbers (see chapter two) of the module are related numerically (as in the circuitry of an individual) or *independent* where no such relation exits.

3. **Main PCB circuit**

   This contains the different components and the module instances which construct the full circuitry of the PCB, described in a global form.

In addition, the compiler recognizes the external connections and the rail connections of the PCB.

7.2.2 Compilation Mechanism

Searching for the connectivity information of a given component/module, made by tracing the connections/connected-components, is a time consuming task since it involves recursive search of the components connected to each other and tracing common connections. This could cause relatively long delay which in turn increases the time required for a diagnosis (which linearly increases with circuit size). This reduces the efficiency of the diagnosis system.

For efficient use of connectivity information, the maximum possible part of this task should be carried out prior to diagnosis in a non time critical stage such as compilation. For the above reason, compilation is carried out in two phases, as described below:

Phase #1

In this phase, the symbolic format of the circuit description, which is provided for the users' convenience, is put in a form which could be manipulated by the appropriate blocks in the system.

This form represents a more realistic representation of the PCB circuit since all the modules in the previous description are expanded into a single circuit representing the PCB. The information presented here is included in a structure called the *PCB Record.*

A PCB record also includes a table like construct which stores the frequency with which certain components cause the failure of others connected to them. A number of such records are created for each component which is not part of the individual circuitry. These are termed *Component-Component Frequency Records (CCFR)s.* The CCFRs are dynamically updated with other sections of the knowledge-base. This updating procedure will be the subject of a later section.

Phase #2

This phase analyses each block in the full PCB circuit, produced in phase #1, for connectivity related characteristics (such as connected components, modules, nets, ...). This information is added to the PCB record to be used, when needed, during diagnosis.

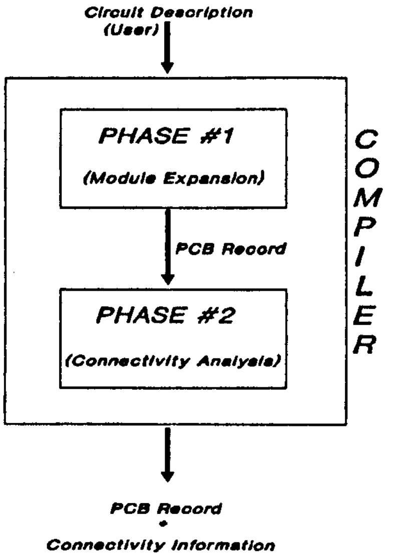


Figure 7-2: The stages of compilation.

The actual mechanism of the two phases described above involve more complications, however, these are not mentioned here since the compiler is a utility for introducing circuit information to the system and is not a subject of research for this thesis. The stages of the compilation process are also illustrated in figure 7-2 which summarizes the above description.

7.3 Filtering Matching Suspects

7.3.1 Faults with Identical Symptoms

Earlier investigation of the failure reports, revealed a number where the malfunction of different components caused the failure of identical test sequences. These were briefly mentioned in chapter three, section 3.3.1. During the random presentation of the data to the automatic fault dictionary based system, described in chapter five, such reports identified themselves by having exactly identical scores.

Since the rules in the rule-base of the system sort suspect faults according to the quality of their matching scores, these rules have no way of distinguishing such reports.

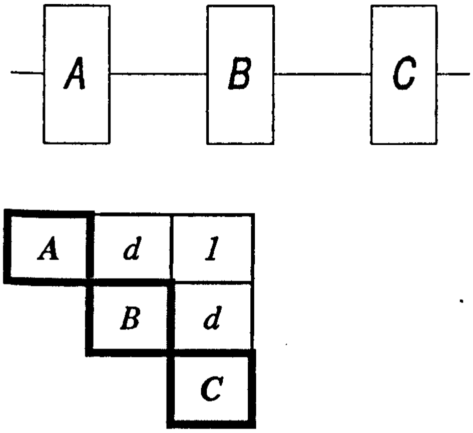
This section investigates the use of connectivity information in differentiating these reports which otherwise are inseparable to the system.

7.3.1.1 The Use of Connectivity Analysis Tables (CAT)

This section describes the development of a technique for recovering the cases of suspects with exact matching scores, described above. The technique is inferred by continuous experimentation and analysis of the different cases which exist in the data. It assumes that matching produces a suspect list which consists of components related to each other in the function they perform such as being part of the same circuitry designed to perform the specific function. This assumption is logical, however, it is not always correct if matching alone is used without considering any electronic information. In general, matching suspects are *mostly* related, but other non-related components could also exist.

The technique utilizes table like constructs, called Connectivity Analysis Tables (CAT). These record the degree of interconnection between each component pair of the indistinguishable suspects and are used to isolate these where matching stops. Each entry in a CAT could contain a number of symbolic representations which indicate the nature of the connection(s) between the two components specifying the entry. Connections between a pair of components are indicated as direct (*d*) or separated by (*n*) components, where n is an integer. In cases where two components are connected by more than one connection of different type, a number of these will exist corresponding to each connection. The information required for the construction of this table is obtained by extracting the relevant information from the corresponding component record in the compiled pcb record. CATs are generated and used at a stage following sorting the matching suspects according to the matching rules in the rule-table (see section 5.2.3.2).

Figure 7-3 shows an illustrative example for using CAT. The suspect components being A, B, and C with the illustrated interconnection in the circuit. CAT indicates that component B has two direct connections to the other two, while each of B and C have a single direct connection.



(a) circuit

A	*d*	*1*
	B	*d*
		C

(b) CAT

Figure 7-3: An illustrative example

Finding the appropriate connections for each CAT entry involves searching the paths associated with each connection of a given component up to a certain ***Level, Search Level,*** which indicates the depth of the search. A given Level is specified by the number of components which the search path passes through during the search. For example, $Level_0$ considers all the components connected directly to each connection of the specified component, while $Level_1$ considers the previous components in addition to components in the search path separated from the specified component by one component. Similarly $Level_2$, $Level_3$ . . etc. can be defined.

Specifying a search level is necessary, since searching the path through each connection of a group of components is a time consuming task and could cause a long delay especially if a large number of components are involved most of which are of high complexity (i.e. large number of i/o), all of these could lead to an increase in the search space and hence a delay in the diagnosis.

However, experimentation revealed that even when the search space is restricted to that specified by level$_1$, most of the examples in the data could be recovered. Figure 7-4 illustrates the concept of levels with the three chips A, B, and C, being a part of a circuit. If level$_0$ is considered, with respect to A, then only the direct connections are searched and hence only B will be picked out by the search, whereas if level$_1$ is considered then both B and C are picked out since the components indirectly connected (separated by 1 block) are considered as well.

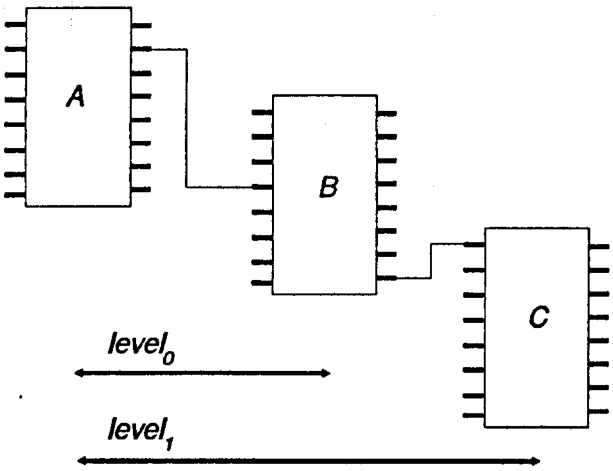


Figure 7-4: An illustrative example of levels.

After constructing CATs for a group of faults some basic concept is required to analyse the tables and distinguish the most likely suspects using the information about the interconnection of each component. *This concept will be explained below, but once defined it can be used as the criteria for ordering the suspects with exact scores, within the suspect list, in a descending order* (the most likely suspects are towards the start of the list).

The concept developed is simple since it is based on common sense in conjunction with continuous examination of the different primary results using connectivity. Basically, the concept could be viewed as a set of rules which favour the component most heavily interconnected with the rest, these are components with the *lowest level connections* compared to the rest, e.g. components with most direct connections are preferred. By applying the above concept to the example of figure 7-3, it can be deduced that ***B*** is a more likely suspect than the other two which share equal likelihoods. This is because B is connected directly to both of the others which have a single direct connection each. In view of the above, the components could be ordered in the suspect list as (B A C) or (B C A).

The following summarize the basic steps of the approach :

Given a *final* suspect list, which contains all the suspect faults associated with their respective scores *after* being processed by the matching rules. The approach searches the list, starting from its *tail*, looking for groups of faults which have identical scores, these are usually formed into groups of adjacent faults at different locations in the suspect list. When a group of exact matches are located then the following is done:

1. The suspects with exact scores are used to form a CAT. If the faults are distinguished clearly then no further action is necessary and the suspects are ordered accordingly, otherwise Step 2 is implemented.

2. The next group of suspects with the next highest scores is used with the group of exacts in 1 to establish a larger CAT. These newly incorporated faults are termed the ***assistants***, since they are only used to isolate the group of exacts and the procedure will not affect them since they have already been ordered using the matching rules (or CAT since they are more towards the tail).

The rationale behind the above is to achieve discrimination (i.e. isolating the most suspected faults from others) by analysing a larger sample of the faults in the suspect list since careful and restricted selection of the suspects, in 1, did not achieve the required discrimination. However, by using a larger sample of suspects there is the risk of including the undesirable poor matches and hence only suspects with high scores are selected. This function is performed for each group of exact matches encountered in the suspect list until the start of the list is reached. The procedure is illustrated in figure 7-5.

Choice of the assistants should be made carefully so that poor matches are not included. The approach followed in choosing the assistants selects the next adjacent match to the group (towards the tail). In the case where a group of exact matches are next to the suspects under consideration then the whole group is used as assistants.

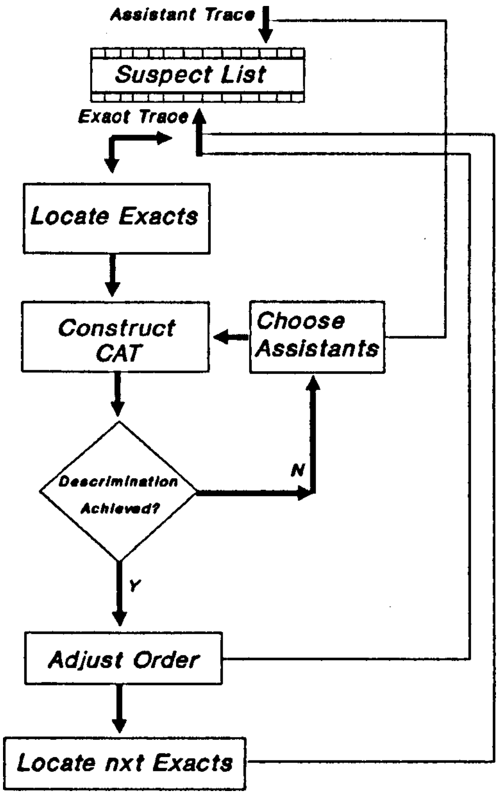


Figure 7-5: The use of CAT.

This approach is cautious and uses as few elements as possible as assistants, however, it is found to be the one which produces the best results since it avoids poor matches as much as possible. The example in figure 7-6 is obtained by constructing a number of CATs for the faults in table 7-1. The table represents each fault with its corresponding matching scores, described in section 5.2.2. The example is extracted from the actual data and shows the level of complexity which the analysis could take. In the first CAT, the three exact matches alone could not provide enough information for discrimination, hence the next component in the suspect list with weaker matching scores is used, as an assistant, to help in the discrimination.

Suspect	*S*	*P*	*G*	*F*
536	*1*	*9*	*9*	*1*
406	*1*	*9*	*9*	*1*
446	*1*	*9*	*9*	*1*
46	*1*	*4*	*2*	*1*

Table 7-1: An example suspect list with corresponding scores.

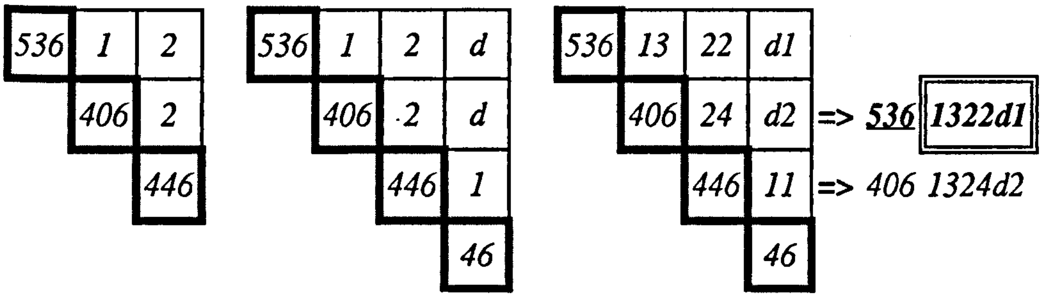


Figure 7-6: Analysis of table 7-1.

Usually, CAT only considers the shortest path between any two components, however, if this does not discriminate between the components then other paths between the components may be considered as well, as indicated in the final CAT of figure 7-6. For example, d1 indicates the existence of one direct connection and a connection through one component between components 536 and 46. The final CAT points to 536 as the most suspected component since it has the shortest interconnections with the rest of the components.

Appendix C shows a number of different matching examples encountered in the data.

7.3.2 Faults with Semi-Identical Symptoms

Section 7.3.1 deals with the cases of exactly identical reports indicating the failure of different faults. This section attempts to investigate cases where reports are similar (have a relatively large number of tests in common), but are not exact. Such reports are reflected in the system by having close matching scores. Although the matching rules can separate these cases, the decision made by the system is not always correct with such close scores.

The same procedure as that used for recovering exact matches is applied to these cases. The results obtained were less encouraging, in that some cases are recovered, however, others were not.

When these cases were investigated it was very difficult to distinguish the two groups since they are all similar in being close matches and no other distinguishing characteristics exist between the two.

The above suggests that *connectivity information should not be used to reverse decisions already made by the matching rules* which give better results for the majority of the cases of close matches than the connectivity heuristics.

7.4 Multiple Module Failure

As concluded from the analysis of the results obtained in chapter five, a significant proportion of the reports which appeared ambiguous to the system were those which indicated the failure of more than one individual. The response of the system to such reports is general in that the fault is associated with one of the components of the last individual.

However, according to the analysis of chapter six, the above seemed to be the case where the number of individuals failed is relatively small (Approx. 2). In the cases where all or most of the individuals fail it is more logical to follow a more *intelligent* procedure in order to locate the faulty component rather than associating the failure with the last individual blindly or holding the components of all the failed individuals accountable.

Such a procedure would be to identify the device(s) to which all the failed individuals are connected and associate the failure with one of these devices. This is logical since the failure of one component at a time is more likely than the failure of most individuals in the circuit.

This procedure could be applied as a check even during the failure of a relatively small number of individuals (or components) since it is logical to check the common components to a number of devices failing a test no matter what the number of failing devices are. Although the circuit associated with the data has identical individuals connected to similar components in the rest of the circuitry, it is still possible to include such a procedure into the system since this could be applied to other circuits due to its generality and will ease the adaptation of the system to deal with multiple board styles.

7.4.1 Fault Identification (Diagnosis)

In order to identify the component(s) which cause the failure of similar tests during testing a number of similar components (or modules), it is necessary first to identify the component(s) associated with the failure of the test(s), since this could ease locating the main cause of the failure.

The initial performance of the system, see section 5.6, indicated that an average of more than 65% of the reports in the data are identified correctly by using matching. This proves that matching is an efficient method of identifying components which are distinguished by the failure of a test or group of tests which test the component or the part of the circuitry associated with it. For this reason matching is only employed, by the method developed, for identifying the component(s) which the failed tests are associated with. This group of components is termed the *Intermediate Components*. Matching rules are used to choose the most suspected of these, which is termed the *Intermediate Component* (*InC*). It is used by the rest of the procedure to identify the main suspect. Results of different experiments indicated that the matching rules usually point to the correct InC since in most of the multiple module failures the failed tests clearly indicate the corresponding InC.

Later, the connectivity routines are used to trace the different signal paths from the connections of the InC in order to locate a component or a group of components which are connected to all InCs in all the failed individuals. This group of final suspects is termed the *Final components*.

Usually a number of final components are produced by the above procedure. These components are obtained by tracing circuit connections only and hence have no additional attributes for distinguishing them. Since we desire to isolate a single component which is more suspect than the rest, an appropriate procedure is developed in order to find the most probable, the *Final Component* (*FC*).

The CCFRs included in the PCB record are developed for this reason. Each such record records the frequency of different FCs causing the failure of InCs connected to them. Hence, the frequencies are used in sorting the FCs according to their likelihood in causing the chosen InC to fail.

This procedure is illustrated in the example of figure 7-7, where component *B* is selected by the matching procedure as the InC. The use of connectivity points to either of $\text{Component}_{\text{square}}$ or $\text{Component}_{\text{circle}}$ as possible suspects for being the FC. Later the statistical information in the PCB record reveals the most likely suspect ($\text{Component}_{\text{square}}$).

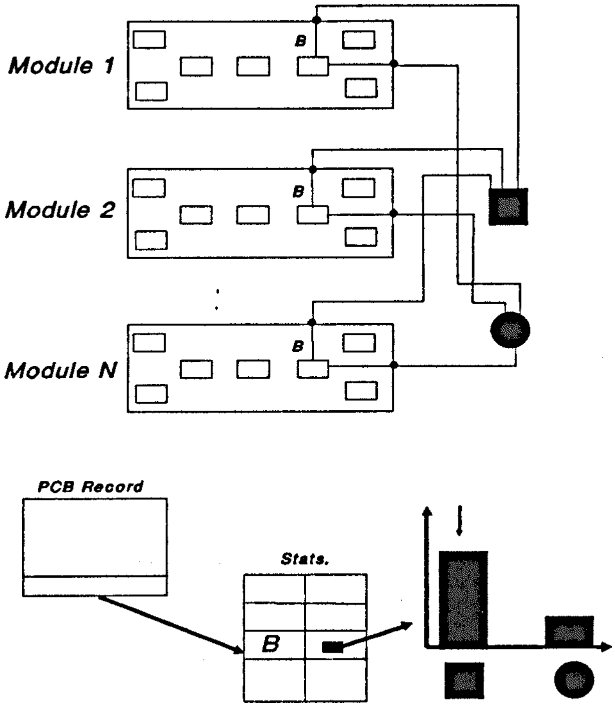


Figure 7-7: An example of a multiple module failure and its recovery

Usually the frequency information is enough to distinguish the correct FC. However, at early stages of the system life only small number of examples are seen hence confident decisions cannot be made using the frequency information. For this reason, in such cases one of the following is used to isolate the components :

1. A CAT could be implemented using all the FCs.
2. Components could be sorted according to their complexity (number of i/o pins) and the most complex is the FC.

Although, the above are simple tasks they were found to be useful in practice and aided in identifying a number of cases.

7.4.2 Updating the System

The updating procedure for the system should be such that, if the system diagnosis is correct, then when a similar report is encountered later then the likelihood of the InC being isolated by matching should be higher and the likelihood of the true fault, FC, causing the failure of the tests of InC should also increase. For this reason the update follows the steps below:

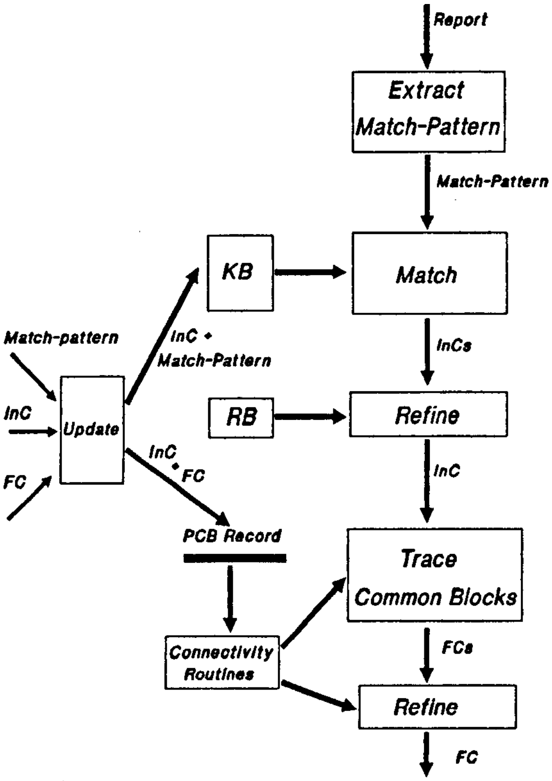


Figure 7-8: Illustration of the use of connectivity when all individuals fail

When the identity of the correct faulty component is introduced to the system, it checks to see whether this is one the FCs. If the result of the check is positive, then the corresponding InC and the match-pattern will be used to update the KB, since a match-pattern identifies InCs (see section 7.4.1). Furthermore, both the FC and corresponding InC will be used in updating the corresponding CCFR in the PCB record. Each CCFR records the number of times the failure of a given FC lead to the failure of tests which identify an InC.

Figure 7-8, above, illustrates both update and diagnosis procedures described above.

7.5 Other Cases

As described in chapter six, the cases of multiple individual failure include cases such as the *Odd Individual* cases, section 6.2.2, and the *Last Individual* cases, section 6.2.1. These could not be recovered with the modifications of section 7.4, since they result from different failure situations and hence require a more specific consideration.

The above cases could be recovered by the corresponding heuristic (obtained from the technicians) used when they are encountered. However, the heuristics incorporated into the system are specific in that they are developed to recognize a given feature in the failure report (the pattern of individual failure) and respond without using any information specific to the circuit of the PCB under diagnosis. Hence if the circuit is altered slightly the diagnosis result will miss the correct fault completely.

Incorporating circuit connectivity information added the possibility of expanding the heuristics employed into a more general form which could cope with changes in the circuit design and hence adds additional flexibility to the system. The use of the system in a multiple board environment will be mentioned in more detail in chapter twelve.

7.6 Analysis of Results

The ambiguities, described in chapter five, occur in small groups which constitute the 25-30% of the data. A number of these groups are such that only few examples are provided and hence a clear view of the case cannot be obtained. For this reason, the improvement achieved by using connectivity information in diagnosis is relatively small (3-4%) compared with the actual size of the data. However, when the improvement achieved is compared with the total number of cases in the data which the connectivity modifications aimed to improve, then a significant improvement is achieved, as will be revealed below.

As mentioned earlier, and as discussions with diagnostic technicians revealed, cases which connectivity aimed to improve occur *less frequently* than the common cases in the data, however, they do occur *regularly*.

Analysis of the system performance, after incorporating the connectivity routines, revealed the following:

1. Filtering the matching output using CAT improved 80-85% of the total cases which showed the need for such an algorithm. While
2. 70-75% of the actual cases which required multiple individual analysis during diagnosis are improved.

The results in (1) do not include cases in which the matching results reveal a number of suspects with exact scores and presented in the correct order (or at least the correct faulty component is located at the start of the suspect list). The ordering, in such cases, is arbitrary and is not performed by the matching rules since all of the suspects have identical scores and hence cannot be distinguished by these rules. The order of suspects with identical scores, with respect to each other, remain unchanged by matching to that when these were presented to the system.

Introducing examples of the above case to CAT algorithm confirmed the order for the majority of the above cases. Examples of such cases in the data are listed in appendix C. These are distinguished from other cases by the letter ***C*** which indicates that CAT confirms the order of the suspect list.

Although the results of applying CAT to identical suspects are treated separately and are not included in (1), the confirmations achieved must be considered as improvements since, as mentioned earlier, the order achieved by matching is arbitrary.

7.7 Conclusions

This chapter investigated the use of connectivity information in improving the performance of the diagnosis performance developed in chapter five.

A circuit compiler was developed for incorporating connectivity information into the system in a form which could be used efficiently by the appropriate blocks of the system. Diagnosis time was reduced by performing part of the connectivity analysis process during the compilation stage.

Connectivity information was used to recover the cases in which multiple individual failure was reported and also to improve the matching outcome in cases where matching rules fail to discriminate.

In order to recover examples of the first case connectivity information was used in association with some dynamically stored frequency information to isolate the faulty component(s). In the latter case a special technique is developed which extracts connectivity information by tracing the interconnections through a number of components with the aid of table like constructs called connectivity analysis tables (or CATs). The search can be specified by a *level* which indicates the depth of the search. Investigations showed that with a moderate search level reasonable identification can be obtained.

The technique described above chooses only suspects from the suspect list which have identical scores and hence cannot be isolated by the matching rules. For this reason *the technique cannot have any negative effect to the suspect list.*

The application of this technique in order to discriminate cases with close matching scores led to a deterioration in the diagnosis results and hence the attempt was abandoned.

CHAPTER EIGHT
THE DEVELOPMENT OF AUTOMATIC SELF-LEARNING TECHNIQUES

8.1 Introduction

This chapter describes investigations aimed towards increasing the independence of the system such that only a minimum amount of help is required from a human expert in assisting it to perform its task. The rule-base is the main part of the system where additional automation is advantageous.

As described in section 5.2.3 of chapter five, the rule-base is a store of rules which represent the relative importance of the individual matching scores, (described in section 5.2.2), and are used to compare one set of scores with another. Each rule, which can discriminate between two suspect faults, is applied in two levels for speed and clarity. The first level ($\text{condition}_{level1}$) is a simple set of tests using only the predicates >, =, and <. This level is implemented as a look-up table for speed. In the second level ($\text{condition}_{level2}$), more subtle comparisons are made.

The rules were obtained by analyzing the system response to different faults and then manually developing secondary conditions within the rule which choose a fault based on its scores. A rule-base such as that described above assumes that the behaviour of the above scores is constant and cannot be altered if any change occurs.

It is thought that when operating in a multiple board environment, a system with an automated rule-base will be more flexible. Such a system can adapt its rules with the changes in the failure report format, which will be different for different ATEs. The effect of these changes could be reflected in a shift in the significance of the scores generated by the matching procedure and/or presenting the scores of different matching algorithms (which could have different number of scores and assess other aspects of the match). In this case the system can adjust its rule-base appropriately for each board to keep up with changes and to make more confident decisions under different situations as more experience is gained with additional examples.

Two techniques of different complexity are developed and are described in detail in this chapter. The first is designed such that a limited amount of information is learnt by the system, which has a rule-base with a similar structure to that of the manual system, described in section 5.2.3.2. The amount of information learnt increases in the second technique and the structure of the rule-base increases in its complexity.

Both techniques concentrate on learning the *secondary rules*, (see section 5.2.3), since they include the most specific descriptions regarding the corresponding main rules and hence are a cause of difficulty for the manual analysis since they require determining relatively accurate score descriptions, between any two matches of the examples of the main rule. Usually these examples are relatively large and in some cases are slightly different.

This chapter builds on the work of section 5.2.3 in chapter five.

8.2 A Statistical Learning Technique

8.2.1 Rule-Table Features

Section 5.2.3.2 of chapter five described the development of the manually constructed rule-table. The rule-table is constructed with a finite number of secondary rules obtained by manually analysing matching results.

The technique described in this section restricts its learning to the finite set of secondary rules incorporated in the manual rule-table only. This is performed by identifying the most appropriate of all the possible secondary rules for the corresponding main rule, specified by the appropriate rule-table entry. It operates by tracking the frequency of each secondary rule performing a correct identification when an example, (a pair of matching score sets that satisfy the main rule condition i.e. $\text{condition}_{\text{level1}}$), is presented to it.

In order to perform the above, all of the different secondary rules are assembled in a *Secondary Rule vector*, $S\hat{R} = \{SR_1, SR_2, SR_3, .., SR_N\}$, where N is the number of all the different secondary rules present in the manually constructed rule-table.

The rule-table is organized such that each entry points to a *Count* vector of the form, $\hat{C} = \{C_1, C_2, C_3, \ldots., C_N\}$. Each C_i is a statistical count of the frequency of success of the corresponding SR_i in $S\hat{R}$.

In order to illustrate the general form of such a rule-table we refer to the example of table 5-4. In this case $S\hat{R}$ = {OK, ~OK, OKif, ~OKif} which includes all the different secondary rules in the table. A possible rule table could be as illustrated in table 8-1, which is for a two score matching system (S_1 and S_2). The statistical count for the secondary rules is represented in the form of a bar chart for each entry. The table represents an intermediate stage in the life of the rule-table where a number of examples has been seen and a clear secondary rule can be distinguished (indicated by the highest bar).

The rule-table formed for the actual system has a similar form, however, the secondary rule vector is larger and all the 81 locations have such statistical information.

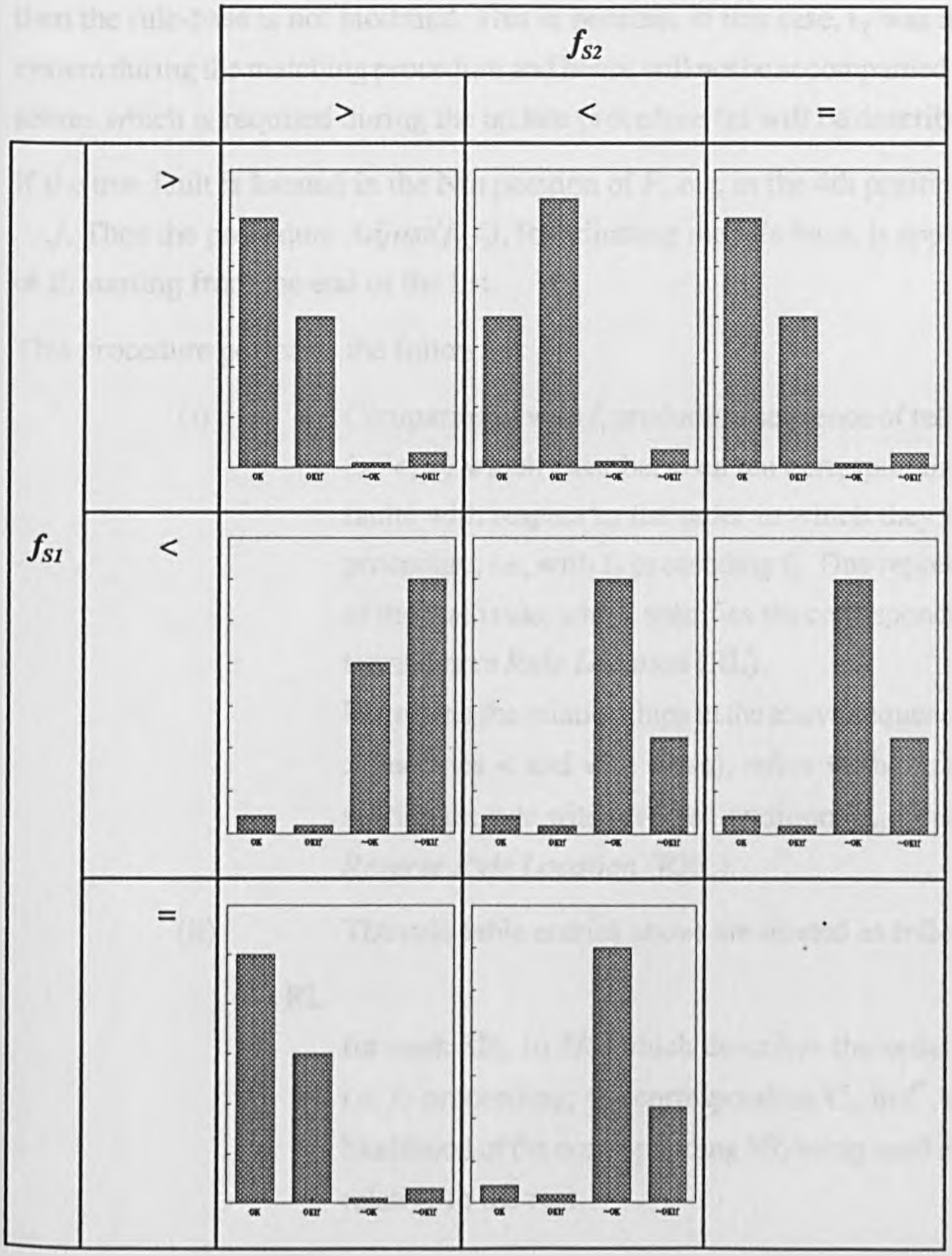


Table 8-1: A graphical illustration of a rule-table generated using the statistical technique

8.2.2 Rule Update

The suspect list produced by the system for a given failure report is stored in order to be used later during update, at which stage the system is informed by the user of the identity of the true faulty component number f_T. During update the stored suspect list is scanned in order to assess the diagnosis made by the system. If f_T is included in the suspect list, then the rule-base is modified otherwise no modification is performed. However, if f_T is not in the suspect list then the rule-base is not modified. This is because, in this case, f_T was not identified by the system during the matching procedure and hence will not be accompanied by a set of matching scores which is required during the update procedure (as will be described below).

If the true fault is located in the Nth position of $\boldsymbol{F}$. e.g, in the 4th position $F=\{f_1,f_2,f_3,f_T,f_5 \ldots\}$, Then the procedure ***Adjust($f_T f_i$)***, for adjusting the rule-base, is applied to each element of F, starting from the end of the list.

This procedure performs the following:

(i) Comparing f_T with f_i produces a sequence of relationship predicates (>, <, =) which exist between the corresponding scores of the two faults with respect to the order in which they are presented to the procedure, i.e, with f_T proceeding f_i. This represents $\text{Condition}_{\text{Level1}}$ of the main rule, which specifies the corresponding rule-table entry, termed here *Rule Location* (RL).
Reversing the relationships in the above sequence of predicates (i.e., > becomes < and vice versa), refers to the rule-table entry representing the rule with reverse $\text{Condition}_{\text{Level1}}$. This entry is termed the *Reverse Rule Location* (RRL).

(ii) The rule-table entries above are treated as follows :

RL

for each SR_i, in $S\hat{R}$, which describes the order of $(f_T f_i)$ correctly, i.e. f_T proceeding; the corresponding C_i, in $\hat{C}$, is adjusted such the likelihood of the corresponding SR_i being used in the future is higher relative to the rest.

RRL

for each SR_i, in $S\hat{R}$, which describes the reverse order of $(f_T f_i)$ correctly, i.e. f_i proceeding; the corresponding C_i, in $\hat{C}$, is adjusted such the likelihood of the corresponding SR_i being used is higher.

In order to illustrate the above procedure consider the Suspect list S = {F1, F2, F3}, which is generated by a matching system with two scores; S1 and S2. The faults in S are ordered at an earlier stage during diagnosis. Table 8-2 outlines the scores corresponding to each fault.

Fault	S_1	S_2
F_1	3	4
F_2	2	3
F_3	10	8

Table 8-2: A tabulation of the scores corresponding to the faults in the example.

Now, supposing that the user feedback to the system indicated that the true fault, f_T, is F_3. The update starts with the last fault in S, and compares each fault with F_3.

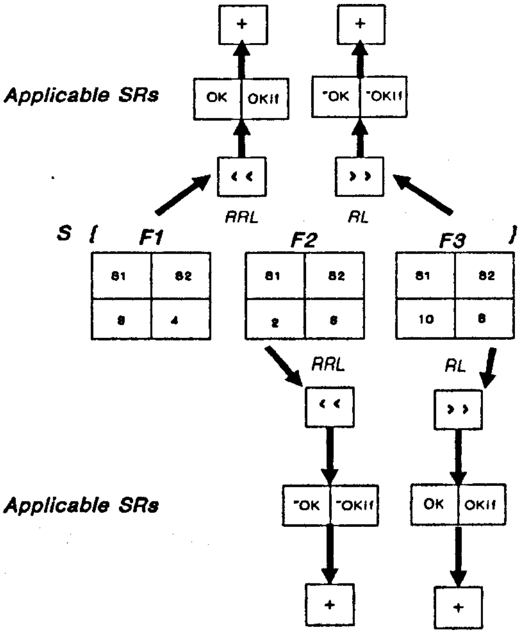


Figure 8-1: Statistical rule update

As described in section 8.2.2, comparing (F_3 F_2) refer to the table location (>>) since both scores of F_3 (which precedes F_2) are greater than the corresponding scores of F_2. Since, it is known that the order of faults is correct (OK), therefore, the secondary rules (OK and OKif) will be increased in frequency (since they allow F_3 to precede F_2). The opposite location in the rule-table (<<) , is the location which will be referred to if F_2 precedes F_3, which is not the correct order (it is not OK), hence the secondary rules which indicate a change should be incremented (~OK and ~OKif). In a similar manner the rule regarding the order (F_3 F_1) is updated. This explanation is summarised in figure 8-1 above.

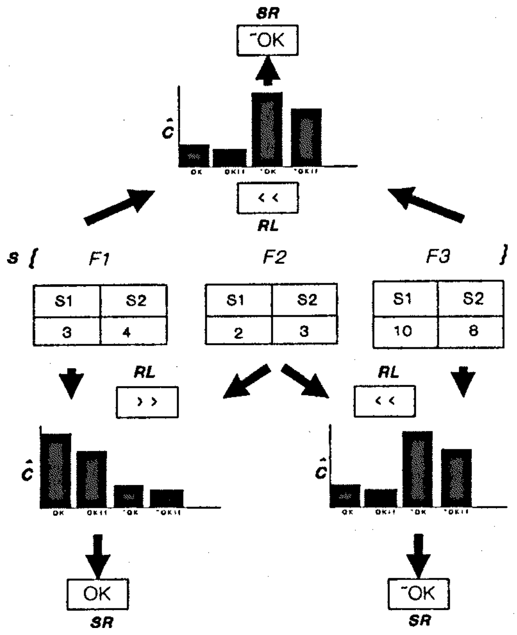
It is important to note that every time the OKif is incremented, then OK should be updated since OK is a more relaxed form of OKif, (which has more conditions). For this reason the increase during the update differs according to the complexity of all the actions which could lead to the correct action in a given case.

As described above, updating a single fault involves updating all the suspect faults in the suspect list associated with the fault. In addition, updating each suspect in turn involves updating both RL and RRL. This indicates that updating a single fault involves a multiple number of update procedures of the rule-base and hence subjects the rule-base to a number of training examples (depending on the size of the suspect list). For this reason, the rate at which a rule-base is constructed, and hence the rate at which the system learns, will be relatively fast.

8.2.3 Rule Execution (Diagnosis)

When the matching algorithm provides the system rule-base (RB) with the list of suspect faults (*S*), these are sorted according to the rule-table which specifies the order of each pair of suspects with respect to each other.

Sorting of the suspects is illustrated in figure 8-2 with the same example used as for the update procedure, (table 8-2). Assume that the faults produced by the matching process produced the list $\{F_1, F_2, F_3\}$ (*not* ordered according to their likelihood of causing the failure). In order to sort the list above, the scores of each pair of faults are examined and the appropriate change is made to their order with respect to each other according to the secondary rule with the highest frequency in the appropriate rule-table entry. For example, comparing F_1 and F_2 indicates that both scores of F_1 are larger than their corresponding scores in F_2, which points to the table entry at >>. Assuming that $\hat{C}$ (which the entry points to) has the status indicated, in a bar chart form, in the figure, then the secondary rule with highest frequency is chosen. This turns out to be OK. A similar procedure is performed when comparing F_1 with F_3 and F_2 with F_3 as illustrated in the example in figure 8-2. The resulting final order of the faults is $\{F_3, F_1, F_2\}$ i.e., F_3 is the most likely fault and F_2 is the least likely.



Correct Order is { F3 F1 F2 }

Figure 8-2: Statistical rule execution.

8.3 Statistical-Ranges Learning Techniques

8.3.1 Introduction

The statistical technique, described in the previous section, attempts to learn a restricted amount of information, such as identifying the appropriate secondary rule from a number of manually developed ones. Some of these secondary rules are not general (since they examine specific scores rather than all scores) because they were developed originally to treat specific cases during the development of the manual rule-table. Such secondary rules decrease the generality of the rule-base and this can be observed particularly when some of the attributes in the testing procedure are altered (e.g. the test data format which could be affected by a change in the ATE program). This could impose a different behaviour on the matching scores and hence could alter the specific bounds examined within the secondary rules.

In spite of the limitations in the statistical technique, an encouraging performance is obtained as it will be seen later. This led the way to expanding the learning procedure so that more information can be monitored and learnt.

In the light of the above, a technique was developed which attempts to learn the information incorporated within secondary rules in addition to statistical information similar to that of the previous technique. The rest of this section will be dedicated to describing and testing this technique

8.3.2 The Development of the Technique

As mentioned in section 5.2.3.2, a *secondary rule* has the following form:

$$\textbf{IF}\quad [\mathit{CONDITION}_{LEVEL2}]\ \ \textbf{THEN}\ \ [\mathit{ACTION}]$$

Action describes the state of the current order between the two suspect faults, under consideration by the rule, given that $\mathit{CONDITION}_{LEVEL2}$ (in addition to $\mathit{CONDITION}_{LEVEL1}$ which must be already satisfied) is satisfied by them. As mentioned in the same section, an action is usually a predicate such as OK, indicating that the current order in the suspect list need not be changed, whereas ~OK indicates the reverse. $\mathit{CONDITION}_{LEVEL2}$ usually examines the amount in which one or more of corresponding scores vary in their values. For example, a possible secondary rule for the two score matching system of table 8-2 would be:

$$\textbf{IF}\quad [S_1\text{-}\hat{S}_1 > 5]\ \ \textbf{THEN}\ \ [\textbf{OK}]$$

Where S_1 and $\hat{S}_1$ represent the value of the score S1 in the two suspect faults presented to the rule. A secondary rule could use more than one score, as in the following example:

$$\textbf{IF}\quad [S_1\text{-}\hat{S}_1 > 5\ \&\ S_2\text{-}\hat{S}_2 > 4]\ \ \textbf{THEN}\ \ [\textbf{OK}]$$

Hence, for a system with *N* number of scores, $\{S_1, S_2, \ldots, S_N\}$, a general secondary rule form would be:

$$\mathit{IF}\quad [S_1\text{-}\hat{S}_1 \# K_1\ \&\ S_2\text{-}\hat{S}_2 \# K_2\ \&\ \ldots\ \&\ S_N\text{-}\hat{S}_N \# K_N]\quad \mathit{THEN}\quad [\mathit{ACTION}]$$

where

S_i ith score of the first suspect

$\hat{S}_i$ ith score of the second suspect

K_i Constant integers

# is one of the predicates (>, <, =)

i 1. . . N

In the above form $CONDITION_{LEVEL2}$ examines only one of the bounds for any of the *score differences*, S_i-$\hat{S}_i$. A more specific representation for the checks aimed by each of the differences would be if both bounds are examined (upper and lower). Hence a more appropriate representation would be:

$IF\ \ [k_1 \geq S_1\text{-}\hat{S}_1 \leq K_1\ \&\ k_2 \geq S_2\text{-}\hat{S}_2 \leq K_2\ \ \&\ \ldots\ \&\ k_N \geq S_N\text{-}\hat{S}_N \leq K_N]\ \ \ THEN\ \ [ACTION]$

Where k_i and K_i are constant integers representing the lower and upper bounds respectively.

In the previous learning technique, the statistical technique, a number of manually designed secondary rules are presented to the system, which is allowed to learn which of these rules are appropriate for each rule-table entry through building statistical information regarding the secondary rules.

The technique described in this section attempts to develop the full secondary rule for each rule-table entry from the example scores presented to that entry. This is achieved by recognizing the upper bound (UB) and the lower bound (LB) of each score difference (see above), which are represented by Ki and ki respectively in the above general secondary rule representation.

The most general form of a secondary rule is when no condition is attached to it, i.e. when the secondary rule is represented by the action only. The action could either provide a positive response to the order of the suspects (OK) or a negative response (~OK). Therefore, the technique is initialised by appointing two secondary rules for each rule-table location. Both of these rules consist of a single action only since no UB/LB has been developed yet. However, one of the rules has an action which responds positively to the orders of suspects (OK) while the other responds negatively (~OK). This is because initially the likelihood of any of the actions being applied, when no examples are seen yet, is the same.

As more examples of each main rule are presented, the technique monitors the following for each secondary rule in the corresponding rule-table entry :

1. The statistical frequency of producing the correct outcome when the secondary rule is used relative to the total number of examples presented to the main rule.
2. The UB and LB for each score difference, by analysing the presented score values.

By performing 1 and 2 for a number of examples, it is hoped that a dominant secondary rule will appear and will have the correct set of attributes, UBs and LBs, for its condition. Hence a complete rule in the form presented above is formed.

(1) is performed in a manner similar to that of the statistical technique, however, in order to develop UB and LB a special heuristical algorithm is developed which tracks the ***range*s** of each score difference in a secondary rule. The algorithm tries to deal with changing ranges, i.e. when new examples indicate a different range, to which the rule-base may be subjected especially when dealing with multiple PCB styles. This algorithm will be the subject of the next section.

8.3.3 The Construction of Ranges

Given a set of example numbers, this algorithm attempts to identify a ***range*** which these numbers lie within. A range is identified by an upper bound (UB) and a lower bound (LB). The rest of this section is devoted to explaining this algorithm.

Earlier analysis of the data revealed the existence of some reports derived from a given fault, which are however, diagnosed to be due to different faults. These reports present noise to the system and hence also introduce noise to the matching rules. For this reason this algorithm should be able to deal with a limited number of noisy score examples. In addition the algorithm should be able to deal with forming a new range if new examples show that the current range is out of date. Dealing with the above situations increases the complexity of the algorithm as will be described later.

As examples are presented to each rule-table entry, after satisfying $\text{Condition}_{\text{level1}}$, a number of possible ranges are constructed based on all the score differences encountered, as will be explained later. A large number of these ranges are only temporarily explored until a clear range is identified which is based on the majority of the examples. Ranges are constructed and manipulated in a buffer, termed the *Ranges Manipulation Buffer* (RMB)

As mentioned above, the algorithm generates a large number of trial ranges in its search for the optimal range. For this reason the size of the RMB multiplies with each incoming score difference. However, with experimentation it was found that after being presented with a limited number of score differences a range can be deduced relatively accurately. Hence it is unnecessary to allow the RMB to grow large (greater than 100) and this would slow the procedure due to the large amount of memory used. For this reason there is a limit beyond which the RMB cannot grow. This limit is specified as a period in terms of the number of ranges in the RMB and is termed the Range Building Period (RBP).

Each range in the RMB is presented in a standard form which includes the following information about the range :

[UB LB PF]

Where

LB Lower bound of the range.

UB Upper bound of the range.

PF *Periodic Frequency* of the range. This represents the number of scores seen within this range within a given RBP

The rest of this section describes the major steps involved in building and manipulating ranges in order to find an optimal range for a given score difference.

8.3.3.1 Processing Scores (Divide and Expand)

This section describes the way in which each incoming score difference is processed in order to produce a range or a number of ranges.

The new score difference, s, is compared to each range in RMB. Comparing s with a given range $[LB_i\ UB_i\ F_i]$ involves performing one of the following operations depending on satisfying their corresponding conditions. Each operation involves the addition of extra ranges into the current RMB.

1. Expand

i. $s > UB_i \Rightarrow [LB_i\ \ s\ \ F_i{+}1]$

ii. $s < LB_i \Rightarrow [s\ \ UB_i\ \ F_i{+}1]$

2. Divide

$$LB_i \leq s \leq UB_i \Rightarrow [LB_i\ \ s\ \ 1]\ \&\ [s\ \ UB_i\ \ 1]\ \&\ [LB_i\ \ UB_i\ \ F_i{+}1]$$

The expand operation includes a score difference which is not included in the current range hence a new range is formed which includes s and the elements included in the original range. The new range is added to all of the already existing ranges in the RMB. Also since the new range already includes all the examples of the original range so its PF will be one more than that of the original range. This operation increases the size of the current ranges and no division is performed here since this is done when score differences triggering the divide operation are encountered. The expand operation is illustrated in figure 8-3.

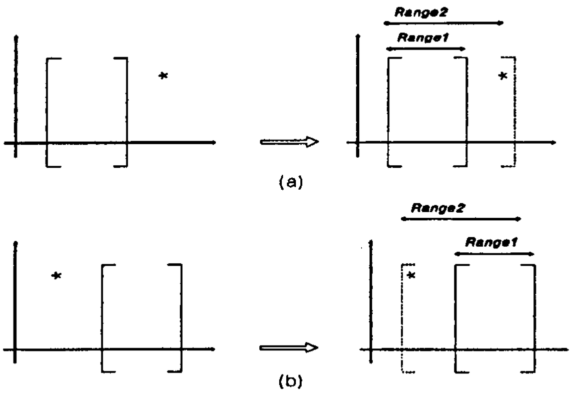


Figure 8-3: The Expand operation.

The divide operation, however, is performed in order to explore new ranges which could be embedded within large ones created during the expand operation. Since both new ranges added into RMB are new ones which do not necessarily cover any of the examples of the original list, both have a PF of 1. However, the original range is increased in its PF since the new score adds to its examples. The divide operation is illustrated in figure 8-4.

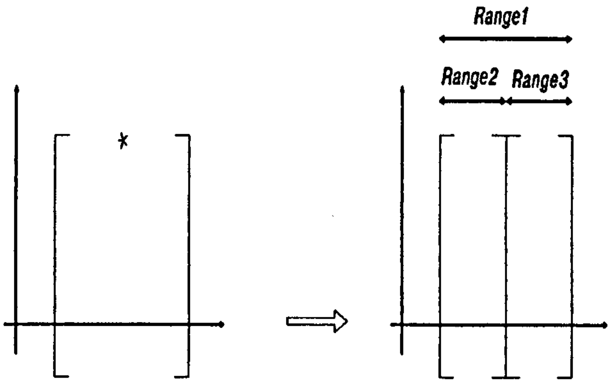
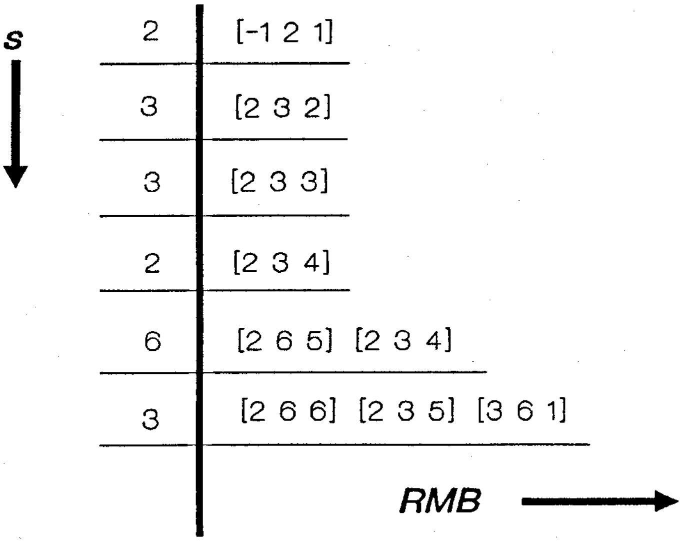


Figure 8-4: The Divide operation.

After each expand/divide operation, the RMB is re-organized such that the ranges with the highest PF are towards the beginning of the RMB. This process assists in choosing the appropriate range during diagnosis when suspects are isolated as will be explained later.

Figure 8-5 illustrates the construction of ranges using divide and expand operations with a number of numerical examples as scores. After introducing each score difference, s, the final state of the RMB is displayed.



s	RMB
2	[-1 2 1]
3	[2 3 2]
3	[2 3 3]
2	[2 3 4]
6	[2 6 5] [2 3 4]
3	[2 6 6] [2 3 5] [3 6 1]

Figure 8-5: An illustrative example of Expand/Divide.

When the first s, 2, is introduced to the system the range [-1 2 1] is formed. The range has no LB, hence -1, and UB=2 and PF=1 since this is the first example seen.

Since the second example, 3, is greater than UB and since no LB exists therefore the range is reconstructed with the only two examples and PF is increased to 2 since this is the second example for the range. Since the range [2 3 2] cannot be divided any more and since the next two examples are already included in the current range, only PF is increased.

The next example, 6, is greater than the UB of the current range, 3, hence expand is performed and [2 6 5] is added to the RMB. Since all the examples covered by [2 3 4] are already included in [2 6 5], PF=4+1 to include the new example. Later, the range with the highest PF is moved to the start of the RMB.

When comparing the final example, 3, with the range [2 6 5], a divide operation is performed since 3 lies between the two bounds of the range. Hence [2 3 1] and [3 6 1] are added to the RMB. In addition, PF is increased for the original range. However, for the range [2 3 4] only PF is increased since the new example is already included in it.

8.3.3.2 Buffering Noise

As mentioned in section 8.3.3, the effect of noisy data is reflected by producing some noisy scores which could mislead the process of building ranges if measures are not taken to reduce their effect. As experimental results indicate these are reflected by introducing false (incorrect) ranges which are especially effective when the actual range is a fraction of the false range and is included in it, in which case the true range can never be recovered.

Investigations revealed that in order to reduce the effect of noise, the incoming score differences should be examined carefully during and after processing them into RMB. This led to the development of a number of steps which are incorporated into the algorithm such that each new score difference is treated as a suspect unless proved otherwise, within a given RBP. This is performed by checking for the consistency of the score among the other scores presented to the algorithm within the RBP. This is performed by placing each new score difference into a buffer structure, termed the Suspect Buffer (SB), at the same time it is processed to RMB. With each incoming next score difference, the following cases are monitored in the SB:

1. Score differences occurring more than once.
2. A number of score differences forming a chain, i.e. a continuous sequence of numbers (e.g., 4, 5, and 6).

In the case when any of the above are located during the regular monitoring of the SB, the corresponding score differences are removed from the SB and they are not treated as suspects for the current RBP.

At the end of a RBP, all the ranges which include score differences still in the SB are removed from the RMB. The score differences are also removed from the SB, however, some score differences from some of the ranges with high PF are added into the SB so that they can be monitored during the next RBP. This will explained in greater detail in the next section.

The procedure is deduced by analysing the effect of noise introduced to the algorithm and is found to produce a significant improvement.

8.3.3.3 Flushing Redundancies

The idea behind using expand/divide, described in the section 8.3.3.1, is to explore as many ranges as possible which could be suggested by the presented score differences before identifying the most likely one. For this reason after applying expand/divide on all the ranges in the RMB, a large number of ranges are produced. Most of these are redundant since only one of them is the best range. The rest could be ranges which are in the proximity of the best or are either larger than it or a portion of it. Most of these redundancies are characterised by having a relatively low value of PF compared with ranges which are likely candidates for being the best range.

For the above reason it is necessary to have a mechanism of flushing most of the above redundancies so that only the best range remains. In general, this procedure could be divided into two. A minor one which is carried out regularly after processing each single score difference, and another which is carried out periodically

Regular flushing include tasks such as removing the duplicate ranges created after each divide/expand procedure. Processing a single score difference could produce a number of duplicate ranges and removing them is necessary since they carry no additional information. Noise buffering mentioned in the previous section could also be considered as regular flushing since it gets rid of a large number of redundant ranges when a noisy score difference is detected.

Periodic flushing is performed at regular intervals specified by the RBP. The size of the RBP is determined by experimentation with a number of values. Sufficiently accurate ranges are detected when the RBP is in the range 20-30.

Periodic flushing gets rid of all the ranges except the best, which remains in the RMB, however, its PF will be reduced by being normalised (divided by RBP). In addition both UB and LB of the range are placed in SB so that they can be traced in case the range is not valid any more (i.e., range shift).

8.3.3.4 Performance Test

The main procedure for assessing the performance of the algorithm is carried out by comparing the *overall* performance of the system with the statistical-ranges technique to that using the statistical technique, and manual tuning. This comparison will be discussed later.

However an initial assessment for the performance of the algorithm was performed by subjecting it to different sequences of uniformly distributed random numbers between specified limits, and comparing these limits with the range deduced by the algorithm. The sequences are made to vary in range at given intervals in order to examine the ability of the algorithm to cope with changing ranges.

The numbers are generated by specifying the range of the numbers to be generated, using mean and standard deviation (std), to an external random number generator for generating uniformly distributed sequences. The generator also incorporated a limited amount of noise with the numbers to test the ability of the algorithm in dealing with noise, since noise is actually experienced in the data which are in turn reflected in the matching score values.

Figure 8-6 illustrates one of a number of random sequences used in testing the algorithm. The numbers presented to the system are monitored at regular intervals by displaying the following information respectively :

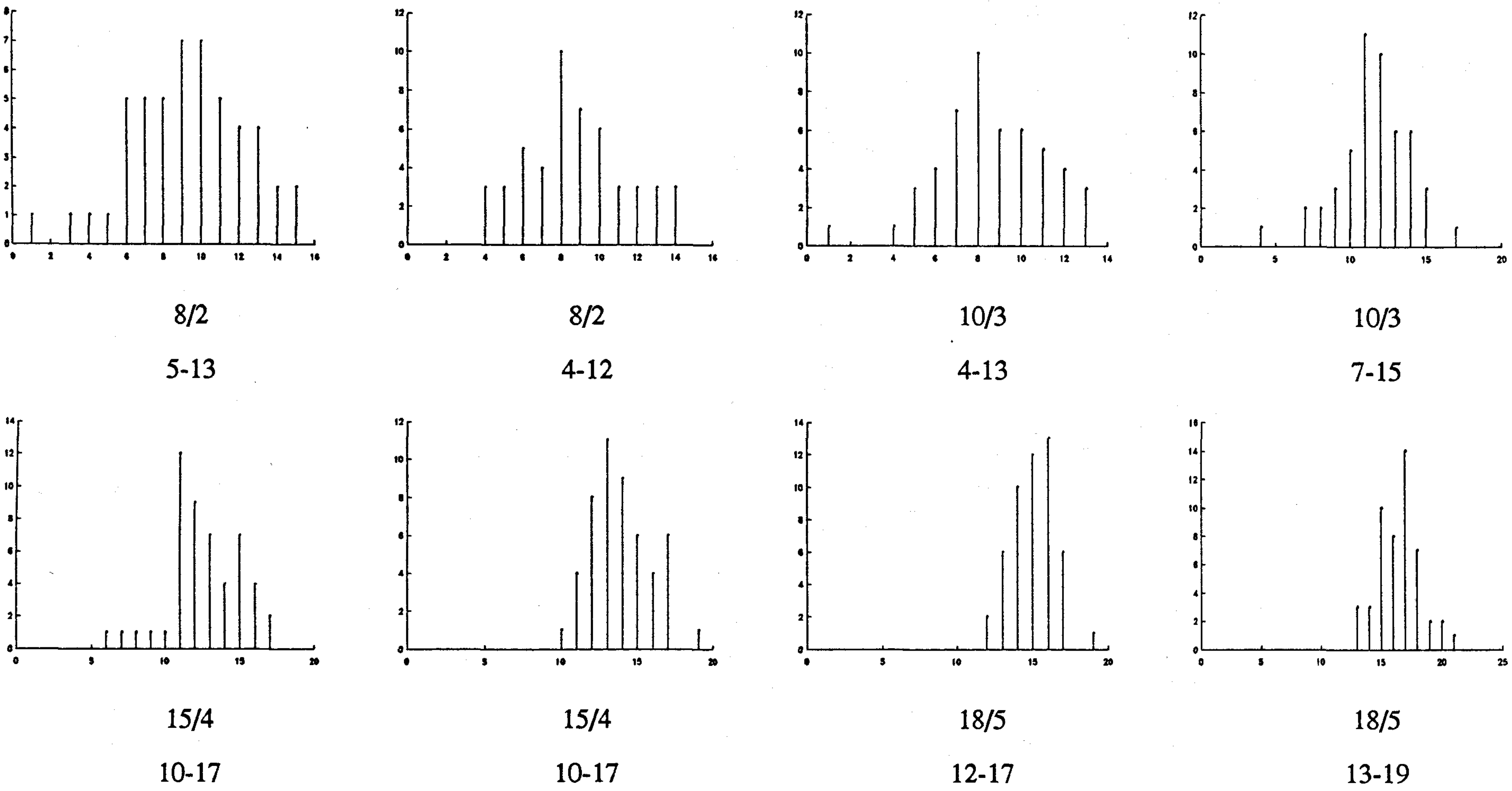
1. The frequency distribution of the numbers presented in a bar chart form
2. The mean/std values used in generating the numbers
3. The range predicted by the algorithm

The results obtained with the sequence of figure 8-6 is typical of that obtained with most sequences presented to the algorithm. Examination of the figure reveals that the range predicted by the system at each interval represents within a relative accuracy a range which is based on the frequency of occurrence of the different numbers within the interval and the proximity of the numbers to each other. A range which is based on the previous elements is termed here the *actual range* and represents the range relatively accurately since it is based on monitoring the presented numbers.

Basics of statistics theory indicate that approximately 97% of a uniformly distributed random data is included within three standard deviations around the mean value. This concept could be used to calculate bounds of an approximate range for the data. A range which is deduced in this manner is termed here the *theoretical range.*

The ranges deduced by the algorithm do not match closely the theoretical range values calculated for most of the intervals in the figure, although the actual range is usually included in the theoretical range. The reason for this difference is in that the theoretical concept, used in approximating a theoretical range, merely indicates where the region (in the distribution curve) where the majority of the data lie without being specific about the values of the bounds (LB and UB). In addition, the inclusion of some noise in the random sequence may effect the calculation of the theoretical range.

The good performance of this algorithm is maintained when used in the diagnosis system to process matching score differences of the test data, although the analysis performed on score differences of a number of rule locations (in the rule-table), indicated a non-uniform distribution behaviour. The reason of this performance is due to the fact that the algorithm is designed to find ranges in sequences of presented numbers based on the frequency of occurrence of the numbers and their distance from each other, which is the case in deducing an approximate range for any sequence of numbers with disregard to their distribution type (provided a range exists).



8/2

5-13

8/2

4-12

10/3

4-13

10/3

7-15

15/4

10-17

15/4

10-17

18/5

12-17

18/5

13-19

Figure 8-6 :An example of a numeric sequence for testing ***Range Construction***.

8.3.4 Implementation

As mentioned in section 8.3.2, the statistical-ranges technique monitors both the statistical frequency of the secondary rules, SC, and the limits for their ranges. Both types of information are incorporated into a *record*, as shown in figure 8-7, for each secondary rule. The statistical information, displayed as a chart, resembles that of the statistical technique and represents an up to date record of the frequency of success for the corresponding secondary rule. Information about the ranges, for the same rule, points to the current range of *each score difference* specified by UB and LB. The bounds of each score difference are continuously adjusted by the ranges algorithm such that the information in the record always point to the most recent range of the particular score difference.

As explained in section 8.3.2, two secondary rule records form the basis of a complete rule representation (record) in this technique. The first records characteristics of positive secondary rules (+), e.g. OK, and the second records those of negative secondary rules (-), e.g. ~OK. The information about each secondary rule type is obtained from the examples pointing to the use of each type and, as experienced in practice, a single rule could meet examples favouring both possibilities although in varying proportions.

Keeping both secondary rules up to date allows either of them to be used when required. This is advantageous in an environment where the likelihood of the two secondary rules is subject to variation.

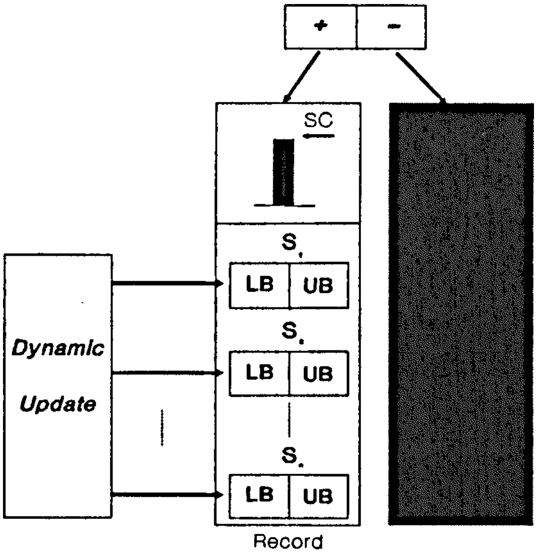


Figure 8-7: The representation of information for a secondary rule

The next two sections briefly describe the way in which diagnosis and update are performed in view of all of the previously described concepts.

8.3.4.1 Rule Execution (Diagnosis)

Some of the features of the mechanism of executing rules resemble that of the statistical technique, see section 8.2.3, in that the rule-base manipulates the suspect list, S, produced by matching. The suspect list is ordered by comparing the scores of each two suspects in the list.

The mechanism of comparing two suspect faults (F_1 F_2) involves comparing their corresponding scores. Now suppose the scores of the two faults are as follows:

F_1	$(x_1\ x_2\ x_3\ x_4)$
F_2	$(y_1\ y_2\ y_3\ y_4)$

Comparing the scores of the two faults, as described in section 8.2.2 will point to the corresponding RL in the rule-table. If the values of SC for the two secondary rules in RL vary *significantly*, then the action part of the secondary rule with the highest SC is applied with no further examination.

However if the values for SC for both secondary rules are *close* then ranges are used to assist in discriminating the appropriate secondary rule. Before describing the procedure involved in using ranges, we define the following :

i. $d = \{d_1, d_2, d_3, d_4\}$

where

$d_1 = x_1 - y_1$

$d_2 = x_2 - y_2$

$d_3 = x_3 - y_3$

$d_4 = x_4 - y_4$

ii. d_i satisfies range R if

$LB_R \leq d_i \leq UB_R$

iii. d satisfies a given secondary rule S, d→S, if each d_i (i is in the range 1-4) satisfies its corresponding range in the rule.

In general using ranges follows the entries in table 8-3 which states each possible combination of d satisfying any of the secondary rules in RL. Maj represents the secondary rule with the slight majority while Min is the one with the minority. Recall that we are dealing with cases where the SCs of the two secondary rules are close since these pose the main difficulty in decision making. Y and N respectively indicate whether d satisfies the secondary rule or not.

d→Maj	d→Min	S
Y	Y	Maj
Y	N	Maj
N	Y	Min
N	N	Maj

Table 8-3: A table for controlling the use of ranges.

As the table illustrates, ranges are used cautiously and the method always favours the secondary rule with the majority to the other, This is because the information obtained using statistical information only is reliable since its based on the historical frequency i.e, experience. This technique tries to maintain the cases which gave good discrimination using statistical information, however, it aims to improve cases where discrimination cannot be achieved using statistical information alone.

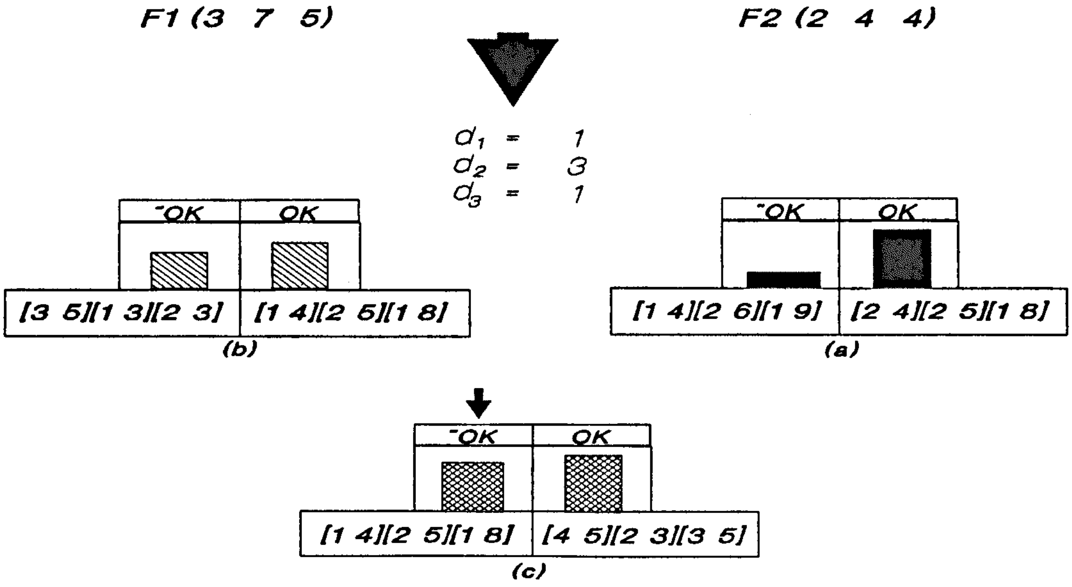


Figure 8-8: Examples of the execution procedure.

The example in figure 8-8 illustrates the case of the two faults F1 and F2, which are compared using three matching scores. The d_is for each of the scores are as calculated in the figure, a, b, and c illustrate three situations which could arise during rule execution.

In (a), the frequency of OK is significantly larger than that of ~OK. Hence the order of the faults is considered to be OK without considering the effect of the ranges.

In (b), the frequencies of OK and ~OK vary slightly (the frequency of OK being slightly more, i.e. OK=Maj and ~OK=Min). For this reason the ranges are considered. d_1, d_2, and d_3 satisfy the ranges of the corresponding scores for OK, however, d_1 and d_3 do not satisfy the ranges of the corresponding scores for ~OK. For this reason the order is considered to be OK again. This situation is an example of the second entry of the table 8-3.

In (c), the frequencies of OK and ~OK are again very close (the frequency of ~OK being slightly more, i.e. OK=Min and ~OK=Maj). In this case d_1, d_2, and d_3 satisfy the ranges of the corresponding scores for ~OK, however, d_1 and d_3 do not satisfy their corresponding range in OK. For this reason, the order of the faults is considered to be ~OK. This situation is an example of the third entry of the table 8-3.

8.3.4.2 Rule Update

The main steps of updating the rules resemble that of the statistical technique, as described in section 8.2.2, except that the updating procedure Adjust(f_T f_i) performs the following :

(i) The rule-table entries RL and RRL are obtained in a similar manner to that in 8.2.2.

(ii) RL and RRL are treated as follows :

RL

The positive secondary rule, +, is updated such that its statistical count is incremented. In addition, each d_i in d (see section 8.3.4.1) is calculated and is used to update the ranges corresponding to the score used to calculate it. Updating each d_i follows the procedure described in 8.3.3.

RRL

The negative secondary rule, -, is updated such that its statistical count is incremented. In addition, reverse values of each d_i,$\bar{d}_i$, is calculated as follows :

$\overline{d}_1 = y_1 - x_1$

$\overline{d}_2 = y_2 - x_2$

.

.

.

Each $\overline{d}_i$ is used to update the ranges of the corresponding score for the negative secondary rule by following the procedure described in 8.3.3.

The reason for using $\overline{d}_i$ instead of d_i is that the order in which the faults f_T and f_i should be presented to RRL in order to use the negative secondary rule is the reverse of that for RL, i.e. the order is $(f_i\ f_T)$.

8.4 Results

The performance of the system is assessed by means of performance charts and learning curves, both of which will be described in the rest of this section.

8.4.1 Performance charts

The performance of the system is revealed separately with each of the two learning techniques using performance charts similar to those used to illustrate the performance of the manually tuned system (see section 5.6).

Figures 8-9 and 8-10 illustrate the charts developed for the system using the two techniques. These should be compared with the charts obtained for the manual system which are illustrated in figure 5-6.

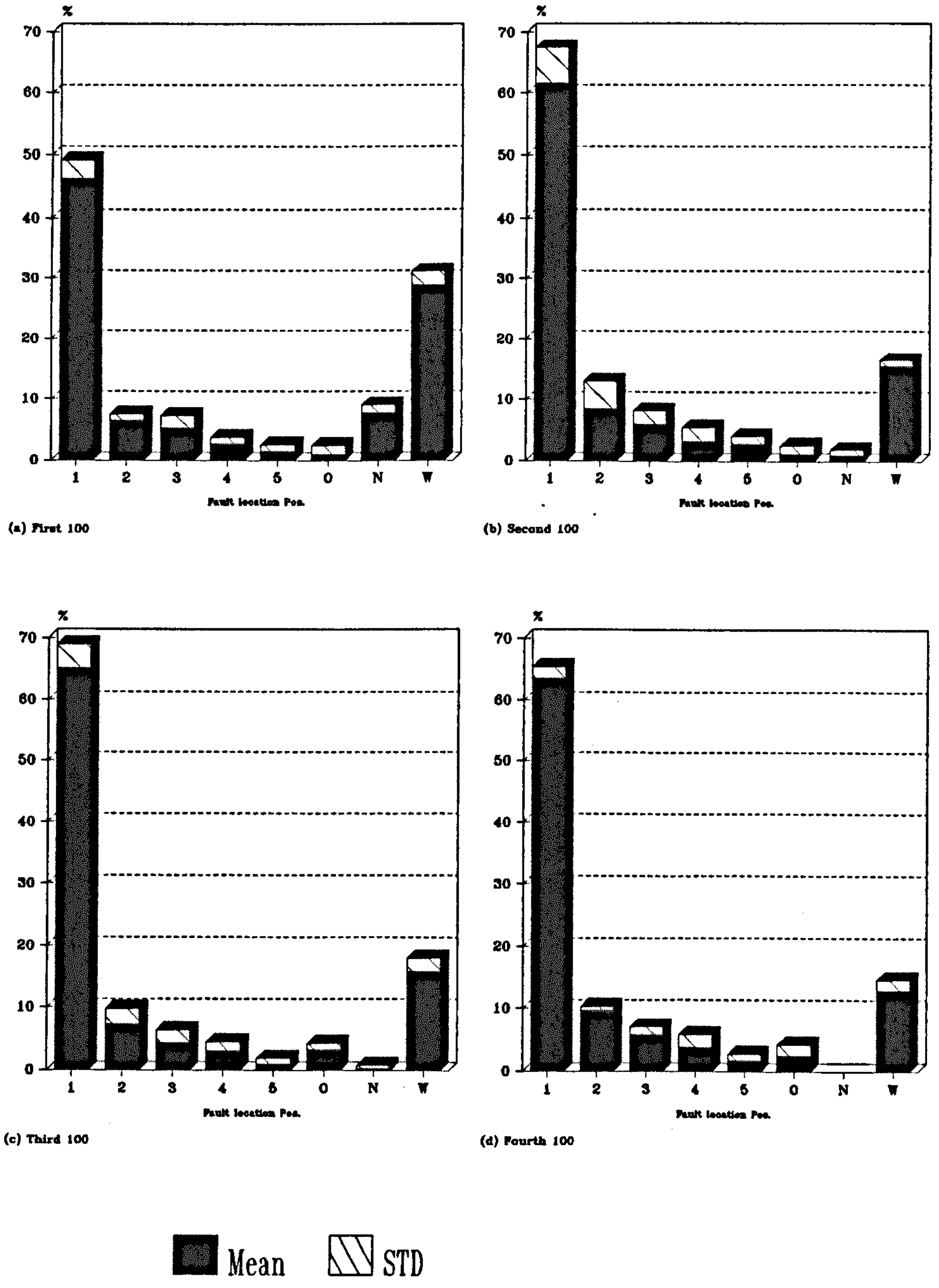
It can be seen from the charts that the ***mean*** fault identifications differ only slightly and show generally similar performance especially in the No.1 position (identified as 1st best suspect), however, *the actual trial results show a wider variation* (when comparing same trials when different techniques are used) which is reflected by the different values of STD (see charts above). This similarity in performance can be explained as being due to the following :

1. The rule-table in each of the techniques is initialized to exactly the same state as the manually tuned table. This avoids incorrect diagnosis in the early stages of the fault finding.

2. The secondary rules placed manually in the table are not chosen randomly, but are based on careful examination of the scores and the outcome of the system in response to examples in the data. Where no examples exist a logical choice is made in terms of the available secondary rules. Hence the secondary rules in the rule-table will not be changed if no examples contradict their use.

3. Examination of the rule-tables produced using the two techniques indicate that only a few rules are used very commonly. These require the same secondary rules to be applied. For this reason these secondary rules accumulate high frequencies in the corresponding rule positions in the rule-table. When the constructed rule-tables are compared with manually constructed ones, the resulting rules look similar (especially in the case of those used commonly). Hence no significant change is noticed with the automation of the two techniques.

The overall resemblance in the ***Mean*** performance of the approaches is an indication that the rule-table is nearly fully tuned and very little improvement is possible. This was deduced during the final stages of manual tuning since no more improvements are possible and the faults which are confused are those which have very close or identical failure patterns.



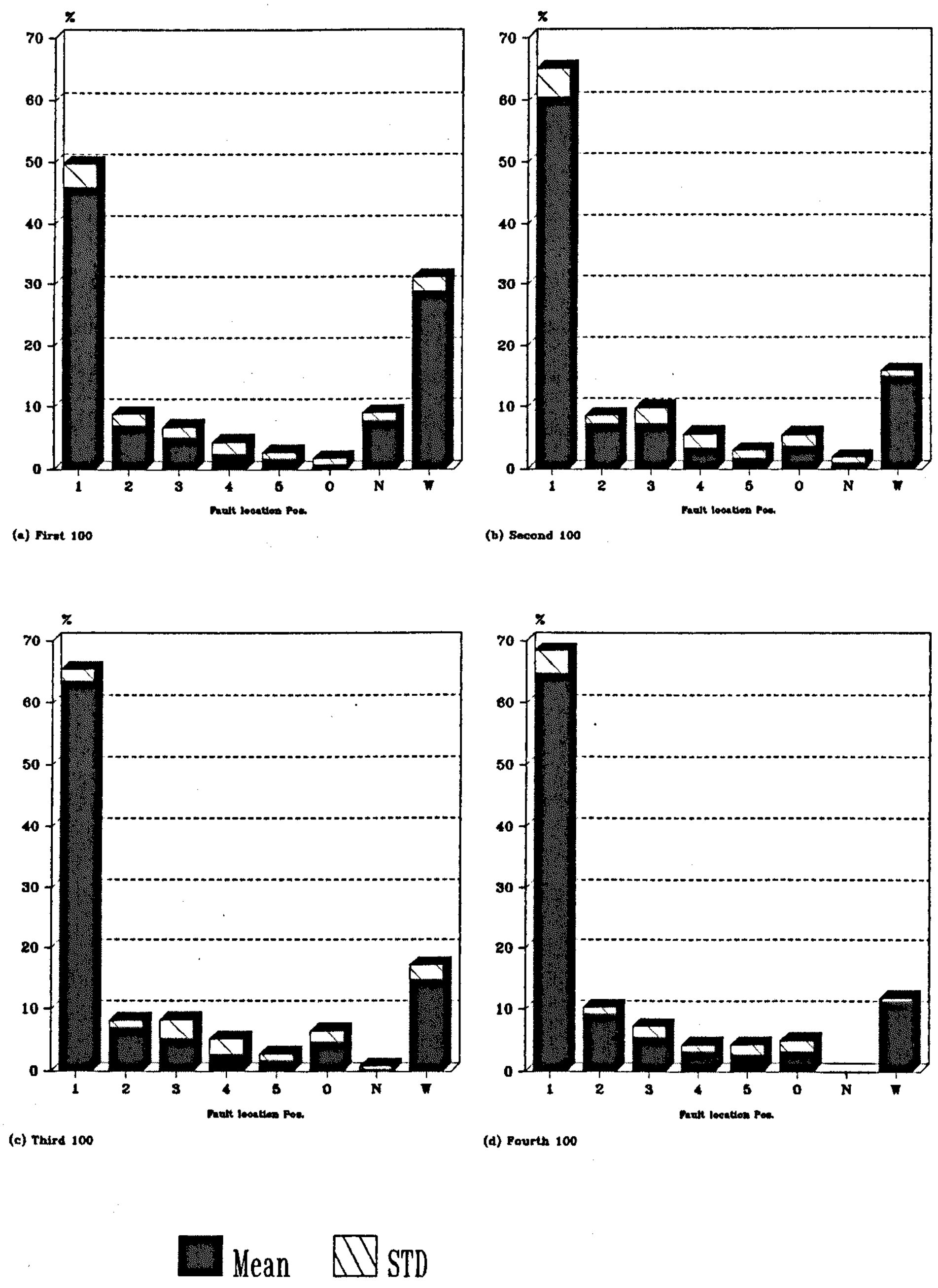
(a) First 100

(b) Second 100

(c) Third 100

(d) Fourth 100

Figure 8-9: Mean fault identification charts using the statistical technique



(a) First 100

(b) Second 100

(c) Third 100

(d) Fourth 100

Figure 8-10: Mean fault identification charts using statistical-ranges technique

8.4.2 Learning Curves

Another way of assessing learning is monitoring the rate at which the system is learning as reports are presented to it. This is performed by constructing plots, termed *learning curves*, which plot the number of faults identified at No. 1 position against the number of reports used in building the system.

8.4.2.1 Initialised Rule-Table

The learning curves are illustrated in figures 8-11, which represents the results obtained with the manual system, and 8-12, which represents the results obtained with the two learning techniques. The performance is monitored at regular intervals specified by a *step* size. For each step size a specific number of reports are introduced to the system and their performance is assessed, with each report being used to update the knowledge-base. Hence the system's knowledge is improved when monitoring the next group of reports. Steps of 10, 30, and 50 are used respectively. The figures in this section include learning curves obtained using steps of 50 reports. Appendix D includes the complete set of learning curves obtained (for the manual system and using the above learning techniques) with all the step sizes experimented.

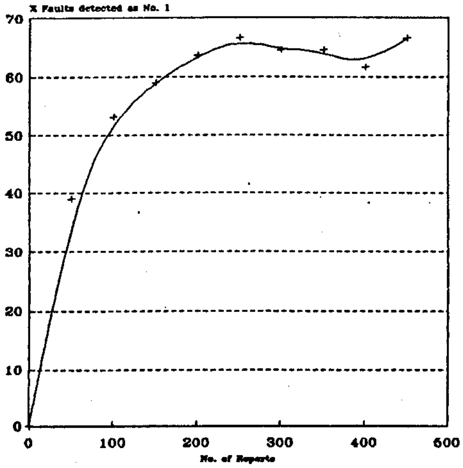
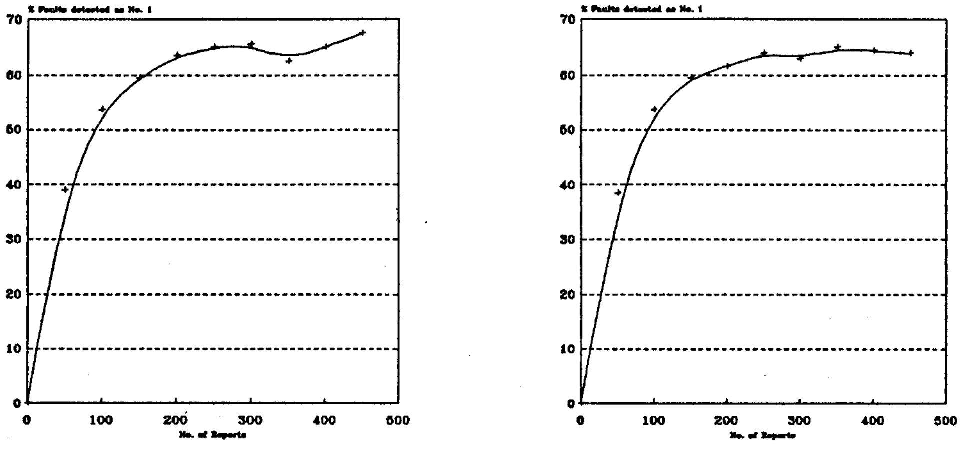


Figure 8-11: The learning curve of the system with a manually tuned rule-table.



(a) Statistical

(b) Statistical-Ranges

Figure 8-12: Learning curves for the different techniques (initialised table)

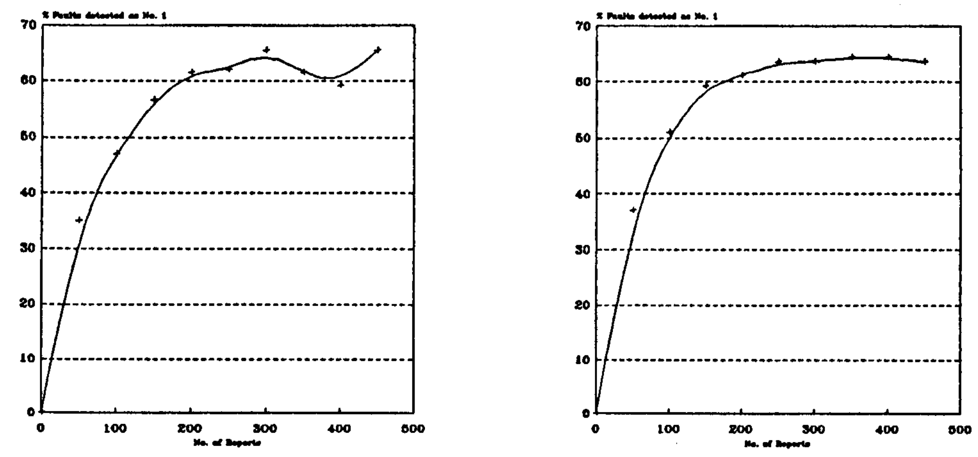
Using steps of 10 reports, the shapes of the curves are not clear, however, as the step size increases to 50 the asymptotic nature of the curves emerges. The curves indicate that learning increases steadily at the start. However it reaches a steady state where learning decreases significantly (or stops). The steady increase at the start is due lack of information in the knowledge-base and hence most reports introduced to the system carry new information which the system learns. However, as the amount of information increases in the knowledge-base the information carried in most of the new reports carry no (or little) new information and hence the learning decreases.

The shape of the learning curve for both manual and statistical systems is similar. However, improved learning could be noted in the system using the statistical-ranges technique, since the rate of learning is higher at the start than the other two and gradually more steady learning is achieved.

8.4.2.2 Non-Initialised Rule-Table

The previously described learning curves are produced by initiating the learning process with the system rule-base initialized to the same state of the manual system. An interesting assessment for the system performance would be by initiating the learning process with an empty rule-base.

The results obtained for the system in this case are represented in figure 8-13. The results indicate a deterioration at the start compared to figure 8-12, however, this improves as the number of reports introduced to the system increases at which stage the performance matches that obtained for the initialised system.



(a) Statistical

(b) Statistical-Ranges

Figure 8-13: Learning curves for the different approaches (non-initialised table)

8.4.3 Comments on the Results

A major success for the two techniques is the fact that they maintain their initial setup after the rule-table has been updated with a large number of examples to which the system is subjected to during fault finding. This is supported by checking different rules after fault finding and comparing them with the corresponding manually tuned table. The table in the statistical technique usually indicates one such secondary rule or group of them which have the same consequence (i.e. the action part) dominating by accumulating high success frequency relative to the rest. While in the statistical-ranges, either the positive or the negative secondary rule can dominate. This shows that most of the secondary rules in the manually tuned system are the ones with high frequency rate in the tables generated using the two techniques. This also indicates that most of the common features for the data are already incorporated into the knowledge-base during the manual tuning.

The statistical-ranges technique traces both upper and lower bounds for a given score. Since all the matching scores (generated by the system here) are *linear* (i.e. the higher the score, the better the match), only the lower bound of each score difference need to be monitored.

Tracing both bounds did not show any significant deterioration in the system performance. This could be due to the fact that the majority of the examples in the data are consistent and hence a range is established after the first few examples of a given rule. This was confirmed during the development of this technique when examples presented to a number of rules were analysed. In addition, the technique uses the ranges carefully and only in certain situations (see section 8.3.4.1).

Tracing both bounds is implemented in order to make the system more general by being able to cope with non-linear scores.

8.5 Conclusions

The work described here was initiated with the aim of achieving rule learning in the diagnosis system and was not carried out for improving the system performance, although this is obviously an advantage.

Two techniques have been developed which vary in the amount of information learnt by the system. The learning concentrates on the secondary rules, since these contain detailed descriptions of the scores regarding certain cases and hence their construction requires accurate analysis of the matching results.

Both techniques achieved a fault finding performance which matched that of the manual system at the start and showed some improvements at later stages. This was referred to in the results section.

Achieving rule learning which matches that of the manual method is an advantage in the system for its use with the current board or in its future use in a multiple board environment since it can gradually adjust its rules to meet the requirement of limited change to the board or the ATE as long as the matching technique can be used in fault finding (i.e., there is not a complete change in the fault finding strategy).

It is thought that one of the main reasons for the success of the two techniques is in the manner they operate and achieve learning. Both techniques are based on monitoring the frequency of various situations. These being scores or secondary rules which is usually the case with human experts (and was actually performed by us during the manual tuning) working with a similar example. A secondary rule is chosen in a given case if the frequency of its success in the case is larger than others and the higher the frequency compared to the others the more confidence exists in applying the secondary rule. The same applies in identifying the bounds of any numerical element from its examples only (provided a bound does exist). Computers remember and count better than us, and hence it is thought that this approach is better than others for this particular task.

The operation mechanism, mentioned above, also enables the two techniques of coping with noise efficiently. Noise is usually represented in corrupt examples in the data resulting from errors in the diagnosis procedure by the technicians or some ambiguities which result in associating reports with the incorrect faulty components. The effect of noisy examples is isolated by detecting their low frequency since it was noted in practice that noisy examples occur much less frequently than those which are noise-free.

Another possible approach could be the use of a pure logical approach similar to that in ***Automatic Decision Tree Generation***. One such approach is experimented within the next chapter and is thought to be less adequate for the type of learning required here since it lacks any robustness and flexibility. Details of this approach will be the subject of the coming chapter which also includes a more comprehensive comparison with the techniques developed here.

CHAPTER NINE
LEARNING WITH VERSION SPACES

9.1 Introduction

The techniques developed for learning matching rules in chapter eight are based on heuristics and were developed to mimic procedures used by human experts in identifying/developing rules in order to sort matching scores produced by the matching algorithm. The techniques are strongly based on the procedures followed by the author in developing the manual rule-table. The results obtained using both techniques are encouraging as the performance matches (and even slightly improves on) that of the manual system with more rapid and steadier learning, as could be seen in section 8.4.

In spite of the good performance obtained with the above techniques it is important to compare the performance with one of the standard methods used in concept learning since this will enable us to evaluate the real performance of the two techniques.

We thus require to compare the results of the previous chapter with a learning technique which is suitable for learning the concept:

$$\bar{x} \text{ is a better set of scores than } \bar{y}$$

Where $\bar{x}$ and $\bar{y}$ are each a set of matching scores (of the form $(x_1 x_2 x_3 x_4)$ and $(y_1 y_2 y_3 y_4)$ respectively) for two different suspect faults. The matching procedure and the meaning of each score is described in section 5.2.2.

This problem is difficult because we do not know the relative significance of each score of the four scores and although we assume each individual score correlates positively with match quality (i.e., the probability that a matched fault is the true fault), we cannot assume that the relationship is linear.

If we make no assumptions whatsoever concerning the correlation of the individual scores with the quality of the match then we can use a learning technique such as decision tree construction. With such a technique we treat each score as a value from a set rather than an ordered domain. However, this approach seems unnecessarily strict and the assumption of positive correlation is a plausible one in at least as much as it models human behaviour as displayed by the diagnostic technicians when comparing failure report.

The adoption of the positive correlation assumption allows us to construct a more economical representation of the concept space by defining a subsumption relation. For example, if (2 4 7 8) is a better set of scores than (3 3 6 9) then we would expect (2 5 8 8) to also be a better set of scores than (3 3 6 9) since each score in the set (2 5 8 8) is greater than or equal to the corresponding score in (2 4 7 8).

This subsumption relation can be exploited in a candidate elimination algorithm (see section 2.4.1.2) whereas it is not clear how this relation can be exploited in a decision tree construction technique.

The use of the version space algorithm (*VSA*) [MIT77 and MIT78] is investigated in developing rules to recognize matching scores. This method is algorithmic in that a number of specific steps are required for its implementation and it is a general method which could be applied to different learning problems. The techniques above were especially developed for learning matching rules for this particular system.

This chapter involves using this algorithm in learning matching rules, investigating its performance, and pointing out to the limitations involved with attempts of dealing with them.

9.2 Learning Matching Rules Using VSA

9.2.1 Basic Procedure

As in the statistical-ranges approach, the aim was to learn the cases in which the true fault (TF) precedes a suspect fault (SF), through the presentation of the different examples of such cases to the algorithm in the form of a pair (TF SF).

The positive examples are the cases in which TF precedes SF, (TF SF), and the negative examples are the reverse cases, (SF TF). As was described in sections 8.2.2 and 8.3.4.2 of the previous chapter, after identifying TF the system performs its learning by using (TF SF) to update the rule position in the rule-table and using (SF TF) to update the reverse location in the table. The same procedure is used here to classify the positive examples (former case) and negative examples (latter case).

The positive and negative examples used for updating the version space are obtained in a similar manner to above and are updated in S and G simultaneously. Efficient representation of the version space required storing the representation of the *complement* of G ($\overline{G}$) instead of G. After each update of S and $\overline{G}$ the two sets will be exact opposites of each other.

Each example is represented in the form $(\bar{x}\ \bar{y})$, where $\bar{x}$ is the vector of scores for the first fault $(x_1\, x_2\, x_3\, x_4)$ and $\bar{y}$ is the vector of scores for the second fault $(y_1\ y_2\, y_3\, y_4)$. Given that $(\bar{x}_t\ \bar{y}_t)$ represents the scores of the faults in the training example and $(\bar{x}_s\ \bar{y}_s)$ represents one of the example descriptions in S. *Then for the training example to be* ***covered*** *by the one in S, the following must be satisfied*:

$\{x_{it} >= x_{is}\}$ and $\{y_{it} <= y_{is}\}$ for all i (i)

Also given that $(\bar{x}_g\ \bar{y}_g)$ is a negative example in $\overline{G}$, *then for the same training example to be* ***covered*** *by the one in $\overline{G}$, the following must be satisfied:*

$\{x_{it} <= x_{ig}\}$ and $\{y_{it} >= y_{ig}\}$ for all i (ii)

(i) and (ii) are termed here the *inclusion equations*

The algorithm starts (as described in section 2.4.1.2) by initialising S to the first positive example and initialising $\overline{G}$ to the first negative example. Any subsequent positive example presented to S is covered by it if it satisfies (i) with any of the example descriptions in S (i.e., is ***better than or identical*** to *any* of the example descriptions in S), otherwise the example is added to the list of example descriptions in S. However, for a negative example to be covered by $\overline{G}$ (and hence *not* covered by G), then (ii) must be satisfied with any example description in $\overline{G}$ (i.e., it must be ***worse than or identical*** to *any* of the examples in $\overline{G}$) in which case it is *not added to* $\overline{G}$.

The procedure for building S and $\overline{G}$ and later using for sorting the suspect faults is illustrated by the example below.

Example

Table 9-1 contains three examples to be used in constructing S and $\overline{G}$ using the procedure described above.

Example	$\{x_e\ y_e\}$
e_1	*{(4 5 4 3) (2 3 2 2)}*
e_2	*{(3 4 4 3) (3 3 3 3)}*
e_3	*{(4 5 4 3) (1 1 1 1)}*

Table 9-1: An example for building $\overline{G}$ and S.

The examples in the table are the positive examples only, the negative examples are obtained by reversing the positive ones shown. In this example we refer to a negative example of a positive example e_i by $\overline{e}_i$.

As described above, initially G includes every possible example and S does not include any example. After processing e_1, S is initialised to e_1 and $\overline{G}$ is initialised to $\overline{e}_1$ ((2 3 2 2) (4 5 4 3)).

S = {(4 5 4 3) (2 3 2 2)}
$\overline{G}$ *= {(2 3 2 2) (4 5 4 3)}*

The above means that S includes every possible example which is *better* than or identical to e_1 and $\overline{G}$ includes examples *worse* than or identical to $\overline{e}_1$. For example, a training example such as {(4 5 5 5) (1 2 1 2)} is in S since 4>=4, 5>=5, 5>=4, 5>=3 and 1<=2, 2<=3,1<=2, 2<=2 according to (i), however, an example such as {(4 2 4 1) (3 2 3 3)} is not in S since in the first match 2 and 1 are not >= than 5 and 3 respectively.

Processing e_2 results in adding it to S since it is *worse* than the example description already in it (i.e., does not satisfy (i)). $\overline{e}_2$ is added to $\overline{G}$ since it is better than the example in $\overline{G}$ (i.e. since it is not covered by $\overline{G}$). $\overline{G}$ and S are now the following

S = {(4 5 4 3) (2 3 2 2)} + {(3 4 4 3) (3 3 3 3)}
G = {(2 3 2 2) (4 5 4 3)} + {(3 3 3 3) (3 4 4 3)}

The final example is covered by S (since e_3 satisfies (i) for any example in S) and $\overline{e}_3$is not covered by $\overline{G}$ (since it does not satisfy (ii) for every example in $\overline{G}$) and hence both remain unchanged.

9.2.2 Dealing with Noise

It was clear from the analysis performed on different rule-tables generated using the statistical-ranges technique, that in some cases up to 20% of the examples for a given situation (a rule position) point to choosing the opposite fault to that indicated by the majority of the examples. These minority cases are *noise.* The statistical-ranges technique remembers the number of examples in favour of each case (experience) and, as a majority emerges, the correct choice is identified by the technique.

However, the version space algorithm does not have any record of the previous cases, other than updating $\overline{G}$ and S, which does not take frequency into account. For this reason a number of measures were taken to isolate *some* cases of noise/inconsistencies. These cases were excluded from building the version space and are as follows:

i. Positive examples already excluded from G.

ii. Negative example already in S.

Since in (i) only negative examples are removed from G and in (ii) S is only constructed using positive examples. Hence the cases point to a contradiction to what is learnt.

The strategy, in doing the above, is to ignore contradictory examples. However, this blames contradiction on new examples only, which is not necessarily true since the contradiction could be in an earlier example already used in building the version space. Hence, the algorithm assumes that most examples are ok and so it comes down in favour of examples seen so far.

In view of the above, if a noisy example (due to an error by diagnostic technicians say) is encountered by the system at an early stage during the construction of the version space, then it is very likely to contradict noise-free examples. This will result in the corruption of the version space. Hence, it can be deduced that *VSA is dependant on the order at which examples are presented to it.*

9.3 Implementation and System performance

This section describes the stages involved in monitoring the performance of the fault diagnosis system, using VSA for developing its matching rules, and the results obtained.

The system performance was monitored in the following two stages :

Stage 1:

Here, the algorithm operates by constructing a version space as described in section 9.2.1. This involves considering each example in the data and using it to update S and $\overline{G}$. The algorithm uses the inclusion relations of (i) and (ii) in testing whether positive and negative examples are covered by S and $\overline{G}$ respectively. The system performance is monitored with each example. At this stage, the system relies mainly on the data in constructing its rules.

The system above showed a poor performance in diagnosing examples from the data. It was clear that it could not generalize (learn) fully the most common of the matching rules such as the case when all scores of a match is greater than the corresponding scores in another match, (i.e. $s_i > d_i$ i = 1..*number of scores*, for two sets of matching scores $\overline{s}$ and $\overline{d}$). In order to learn this rule the system must encounter examples of all the relevant cases e.g. ((2 2 2 2) (1 1 1 1)), (3 3 3 3) (2 2 2 2)), ((4 4 4 4) (3 3 3 3)) . . . etc.

It can be deduced from the above that in order to generalize a certain situation using the above procedure, a comprehensive set of examples are needed (i.e., examples should cover every case of each situation). These are not characteristics of the data used here and hence the poor performance of the system.

Stage 2:

The system in stage 1 uses the inclusion relations (mentioned above) in order to generalise from specific examples. Each inclusion relation is valid only if each score, examined by the relation, is related to match quality in a monotonic increasing relationship (see figure 9-1). Since each of the matching scores used by the system is monotonic increasing then the version space must recognize examples where $s_i >= d_i$ (i = 1..*number of scores*, for two sets of matches introduced to the system ($\overline{s}$ $\overline{d}$)) as positive examples. This means that the system must say *yes* to the order of the matches. Conversely, the system must say *no* to examples where $s_i < d_i$. An example of the former case is ((3 3 3 3) (2 2 2 2)) whereas ((2 2 2 2) (3 3 3 3)) is an example of the latter.

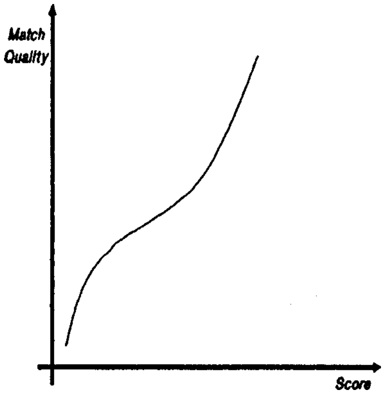
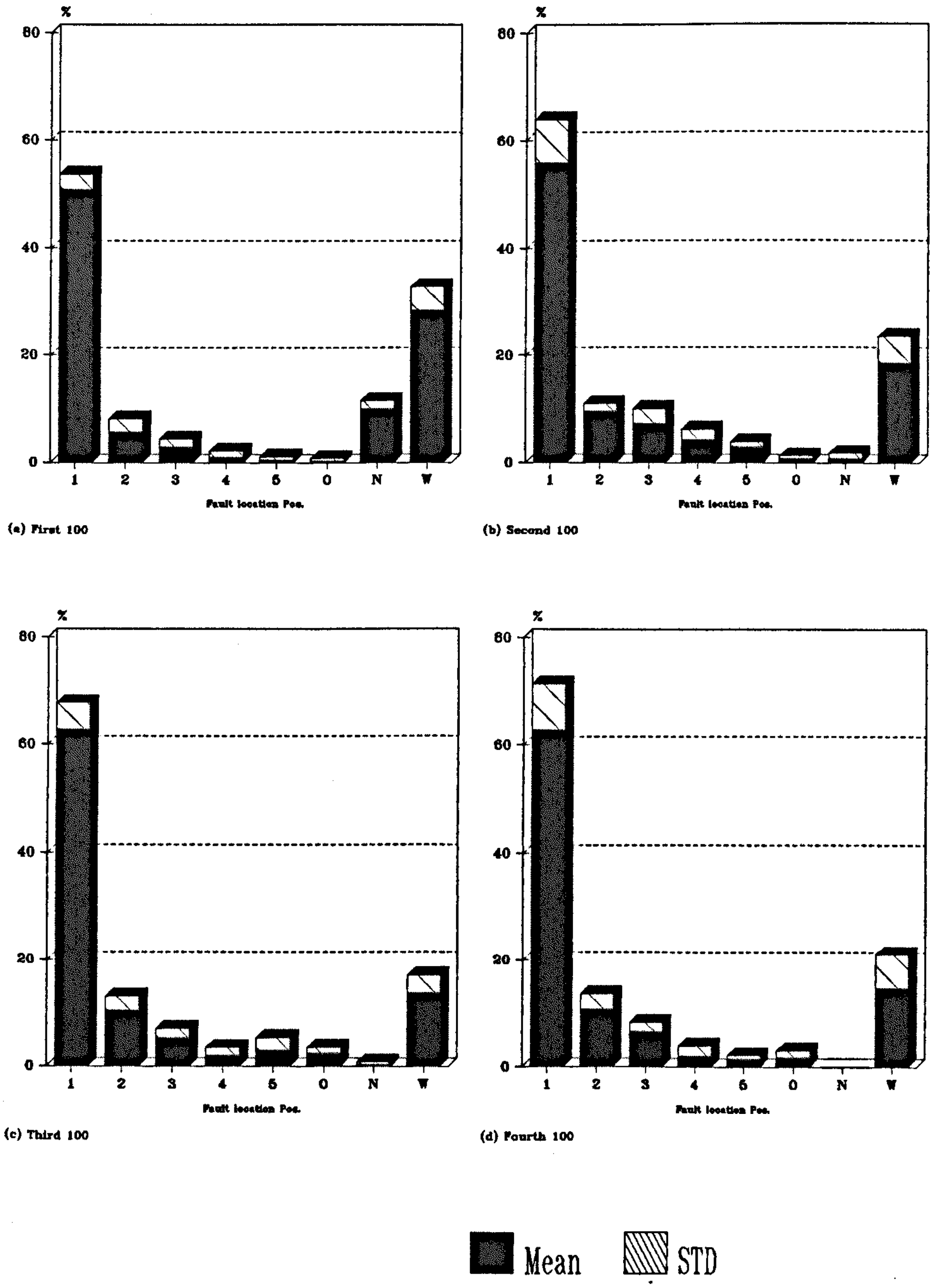


Figure 9-1: An example of a monotonic increasing score.

From all the above, the system here is adjusted to exclude the above cases from building the version space and respond to examples of each case appropriately (when encountered). For this reason learning is restricted only to examples in which respective scores of the two matches have different relations to each other. Examples of such cases are : ((5 3 4 4) (6 8 4

7))and ((7 7 3 2) (3 2 7 7)). Such cases are more difficult to discriminate (human experts require some experience in order to be able to handle such cases) and are cases where additional information (such as the significance of the different scores) might be needed.

Bar charts, similar to those produced for previous techniques, are produced for the results obtained with the above system. Figures 9-2 and 9-3 illustrate the results obtained using the G set description and the S set description respectively. Both results indicate a performance which is better than that of the manual approach. The performance obtained here matches that obtained for the Statistical-Ranges approach, however, this is because the system is manually adjusted to deal with the majority cases while in the statistical-ranges approach these cases are learned by the system.



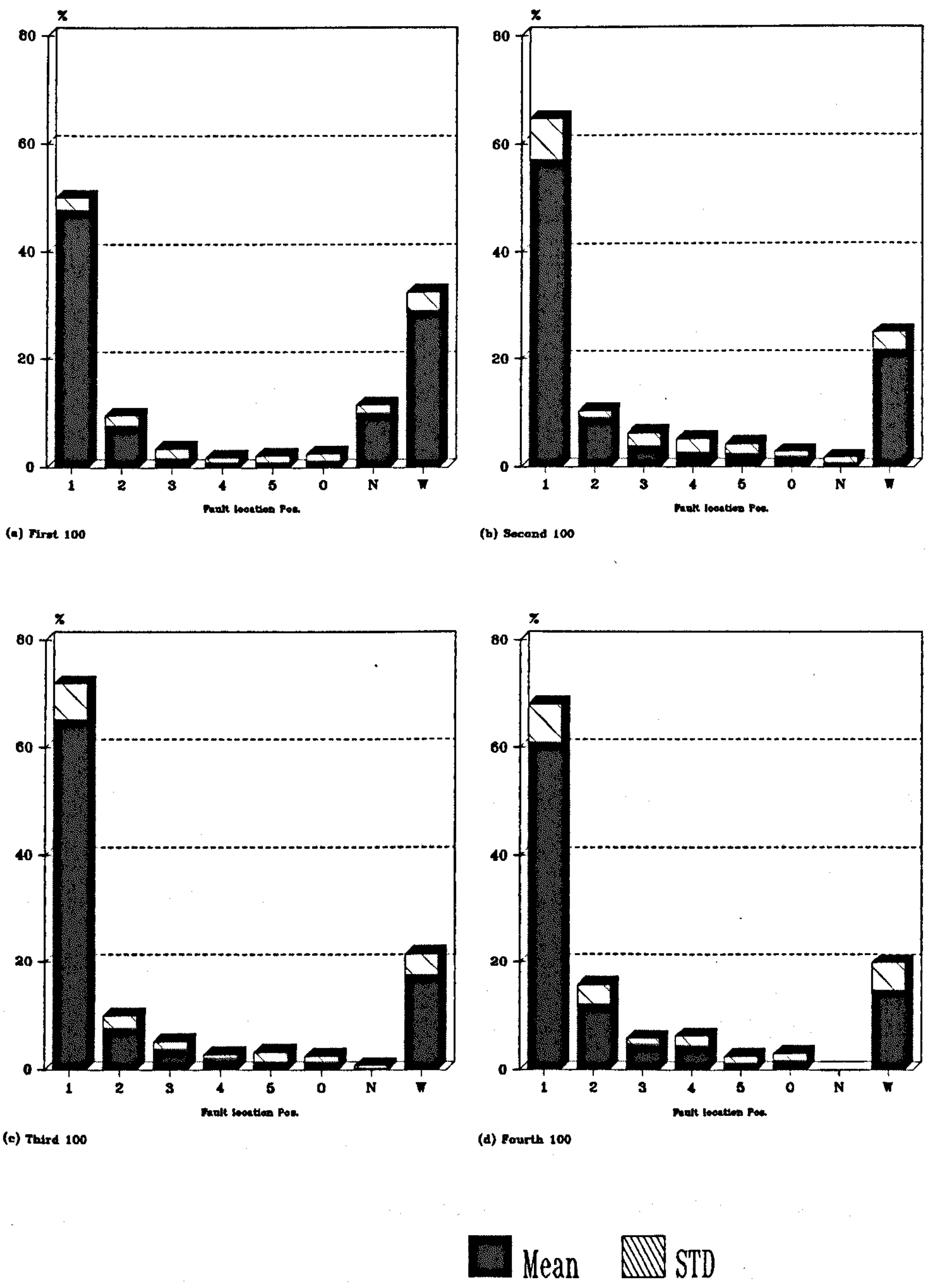
(a) First 100

(b) Second 100

(c) Third 100

(d) Fourth 100

Figure 9-2: Mean fault identification charts using G set descriptions



(a) First 100

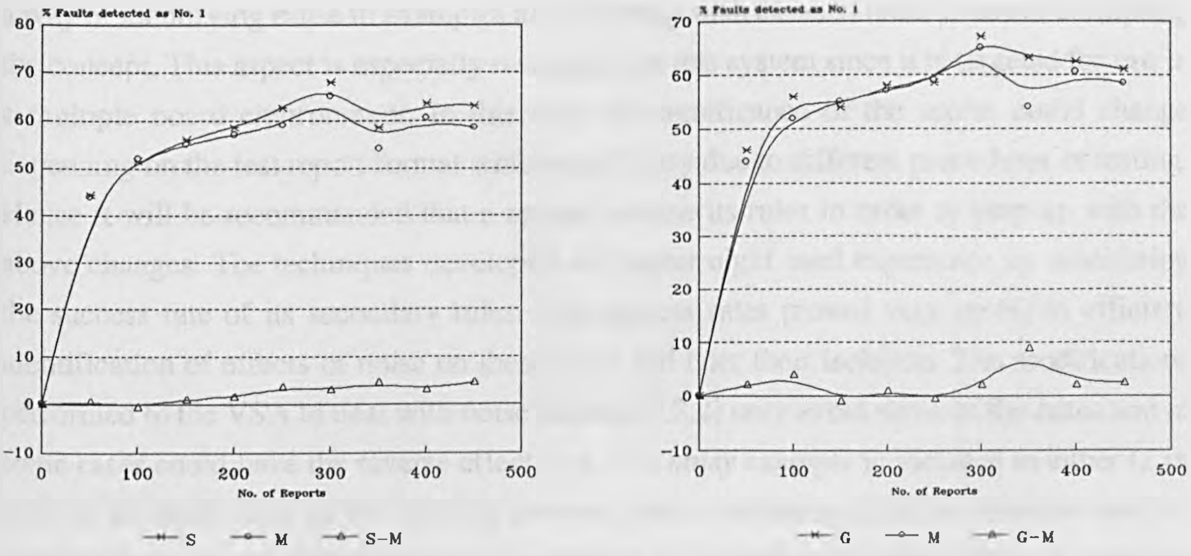
(b) Second 100

(c) Third 100

(d) Fourth 100

Figure 9-3: Mean fault identification charts using S set descriptions

The learning rates of the system using both S and G descriptions are shown in figure 9-4 (a and b respectively). The performance of the system is assessed when none of the version space descriptions are used, i.e. the system only recognizes majority cases (with the manual modification of the system itself, i.e, with no rule learning taking place). This case is associated with M (manual) in the figures.



(a) (b)

Figure 9-4: Learning curves of the system

A more uniform learning is noted using the S descriptions although slight improvement to M can be seen in both cases (with a better performance in S). The learning improvement in S is approximately 3-4%.

9.4 Analysing VSA Learning

This section will comment on the learning procedure using VSA and will compare it with the statistical-ranges approach.

Basically, the major drawbacks of learning by VSA are:

1. The difficulty of detecting/isolating inconsistencies (noise).
2. Slow convergence to the learning concept if the concept space is very large.
3. The *order* in which the reports are presented to the learning algorithm affects the building of the version space.

The rest of this section will consider each of the above in a greater detail.

In an environment where scores are generated by an anonymous system and no knowledge exists about the meaning of these, it is usually desirable to have a learning system which is supplied with all the scores and later identify, as more examples are seen, a concept (or concepts) which can classify these examples. It will be necessary for such a system to have a way of identifying noise in examples and isolating such cases in order to avoid corrupting the concept. This aspect is especially necessary for this system since it is targeted for use in a multiple board environment. In this case the significance of the scores could change depending on the test report format which could vary due to different procedures of testing. Hence it will be recommended that a system re-tune its rules in order to keep up with the above changes. The techniques developed in chapter eight used experience by monitoring the success rate of its secondary rules. The success rates proved very useful in efficient identification of effects of noise on these rules and later their isolation. The modifications performed to the VSA to deal with noise (section 9.2.2) only avoid some of the cases and in some cases could have the reverse effect (e.g., if a *noisy* example is included in either $\overline{G}$ or S set at an early stage in the learning process, then a subsequent proper example may be rejected from the set during sorting the suspects and subsequent update of the S could be incorrect).

The effect of (2) was apparent in that some common cases could not be learned fully unless examples which cover the extreme limits of the case have been encountered at an earlier stage. For example, consider the state of the version space where S is the following :

S=((5 5 4 2) (3 3 3 3))

When a new example e, where e = ((4 4 4 2) (3 3 3 3)), is presented to S to be identified then e will be rejected since it is not covered by S (since (i) is not satisfied - see section 9.2.1). However, comparing e with the example in S reveals that both cases involve two matches in which each score in the first match bare the same relationship with the corresponding score in the second match (i.e. 5>3, 5>3, 4>3, 2<3 for the example in S and 4>3, 4>3, 4>3, 2<3 for e). The only difference in the two examples is the values of the individual scores. Hence a number of examples of the above situation is required (where the values of the scores are different) for this particular situation to be generalised. Another example of such a case was illustrated in section 9.3 (stage 1).

If the two examples above are interchanged, i.e. S=((4 4 4 2) (3 3 3 3)) and e=((5 5 4 2) (3 3 3 3)). Then e will not be rejected by S since it is a better matching case than the one in S (and hence e is covered by S).

The above example illustrates the case in which diagnosis is effected by the *order* in which the examples are presented to the version space. Such cases are handled more effeciently in the rule learning techniques described in the previous chapter since matches with scores which bare the same relationship to each other are treated similarly (since they are directed to the same rule and examined by the same secondary rule). In addition, these techniques are able to identify cases before meeting any examples since they can be initialised manually. The impact of order on VSA is especially effective when considering the effect of noise to system learning. As mentioned in section 9.2.2, meeting noisey examples at early stages of building the version space might result in a corrupt version space caused by the possible rejection of noise-free examples.

VSA is more suitable for situations in which $\overline{G}$ and S are constructed prior to testing since learning is slow, especially if the concept space is very large as is the case here. Since, in such cases, a large number of examples must be seen (during which similar examples are rejected and hence diagnosis fails) in order to cover the whole of a certain situation.

The performance of the system was improved significantly by excluding a large portion of the data from being used in the construction of the version space (see stage 1 in section 9.3). This is because these cases constitute the majority of the examples in the data (if the system is modified to deal with these cases only approximately 50% of the reports will be identified correctly). The VSA learning provides an additional improvement of approximately 5% (this improvement is achieved by learning the difficult cases which the learning was restricted to, see stage 2 in section 9.3) . The performance of the system could be enhanced by using a heuristic approach in searching the membership of a given example in one of the version space sets. This could be performed by implementing a more heuristic comparison in identifying whether a new example is covered by either S or $\overline{G}$. For example, in order to be able to identify e in the example provided previously in this section, (i) in section 9.2.1 must be modified to take the following form :

$$x_{is}-x_{it} >= K \text{ and } y_{is}-y_{it} <= k$$

Where K and k are constant values. In order to cope with the previous example (for e to be covered by the example in S) both K and k could be chosen to be (>1).

In order to see whether the above approach is plausible, more investigations are necessary with the matching scores to find appropriate values for K and k (if it is possible to find such values!).

However, performing the above might result in a non-discrete identification in which relatively large numbers of examples may be wrongly considered to covered by S or $\overline{G}$.

The exclusion of the common examples and inconsistencies from learning using the VSA is in agreement with [TAN90] who says : *"In order to have an effect a new example must be salient; it should either be a near miss or an 'unexpected hit'"*. This indicates that learning only improves if the new examples point to different aspects in the concept space. So examples must not be an inconsistency or a *duplicate*[1] of a case seen before, since in the former the version space is distorted and in the latter no new information is learnt (experience is not significant in the VSA).

Theoretically, learning using the VSA should be able to identify the complete concept space associated with the matching scores (given that the examples are introduced in the correct order), however, for this to happen the training examples must cover every possibility in the space. In order to learn the whole concept space including the majority cases, a large number of salient examples will be required. This seems impractical due to the numeric nature of the scores (continous space) and the restriction imposed to the examples introduced to the diagnosis system by the board failure cases met with the faults found on the PCBs, which do not necessarily produce scores covering the whole space. This will also mean that a longer learning time will be required until learning reaches its steady state. During this time a large number of correct examples will be rejected since examples covering them were not met earlier.

Although the diagnosis results obtained here are acceptable and may be further improved by using some heuristics (as mentioned above), learning is only restricted to a small number of cases, since a large number of examples are required to learn the majority of cases which are associated with a large domain in the concept space. In addition, VSA cannot distinguish noisy examples efficiently.

9.5 Conclusion

In this chapter, the use of one of the standard methods in concept learning was investigated for learning rules discriminating faults based on their matching scores.

Due to the large concept space which includes all the possible combinations of the matching scores, this learning method showed a slow convergence in its search for an optimal rule. For this reason attempts were made to restrict learning to certain cases only.

[1] Duplicates include examples which cover portions of the concept space already covered by a previously seen example.

By restricting learning to minority cases only improved performance is obtained. However, this did not match that obtained using the techniques of chapter eight. Furthermore, the learning in this case required external assistance in isolating noise and buffering common examples and hence led to a less independent system than the one which could be developed with the earlier techniques.

CHAPTER TEN
FURTHER INVESTIGATIONS OF WAYS OF IMPROVING SYSTEM PERFORMANCE

10.1 Introduction

This chapter describes two additional investigations carried out in order to improve the performance of the system. Both attempt to investigate the effect of a more comprehensive matching on system performance by using additional parameters regarding each test failure in a failure report.

The first investigation involves careful analysis of the failure report during matching by detecting similarities *within* parameters of test failure such as test name, mentioned in section 5.2.1.1, in order to be able to further distinguish different faults.

In the second investigation a method of incorporating test measurements into the matching procedure is devised and its effect on system performance is examined.

The rest of this chapter considers each of the above in a greater detail.

10.2 More Comprehensive Matching

10.2.1 Introduction

The matching procedure employed in the system described in chapter five was based on ignoring the conditions, under which tests are carried out, and using only test names in matching. The above matching method could be referred to as ***exact matching***, since it only considers two tests to be a match if they are exactly identical.

By examining the different tests in the data, it can be seen that tests which are related to each other (in either the type of the task they perform or the components they test), contain some common terms which reflect their relationship with each other, examples of these tests are the following :

1) ***ovp-test a-wire thyristor on state*** and

ovp-test b-wire thyristor on state

.

.

2) ***curr feed active slic*** ,

curr feed disable slic , and

curr feed polrev act

.

.

3) ***freq distortion d-a***

freq distortion a-d

.

.

For each of the above examples many other tests exist and also more cases exist in which similarities could be seen. If familiarity exists with the circuit that these tests are applied to, it can directly be seen that the tests in 1 test the protection circuitry and in particular the thyristor, tests in 2 also are designed to test the same device, and tests in 3 are testing a similar function which is the quality of signals produced by a certain device. This similarity in the groups above can be reflected in the common *terms* which exist among the *test name*s of each group. This procedure of closely investigating individual test names forms the basis of a more comprehensive matching procedure, which is the subject of this section.

Further analysis of cases such as the above led the way to investigating the performance of the system if test names are considered more closely rather than just checking whether the test names are exactly identical (match) or not (not a match). This means that matching should be such that, tests could be considered a match even if they only have some common features and are not identical. This type of matching is termed ***inexact matching***. This section describes investigations carried out in order to achieve inexact matching.

10.2.2 Evaluating a Match

In order to incorporate inexact matching into the procedure employed by the matcher, see section 5.2.2, a method of comparison between two tests are required. Such a method must be able to compare the tests of a match-pattern, which is extracted from a failure report, with the tests of each test-pattern in the knowledge-base and produce a score for each test in the match-pattern which reflects the quality of its *similarity* with the tests of the test-pattern. For this reason a ***scoring function*** is required which compares two tests and produces a score of the degree of their similarity. This procedure resembles the exact matching procedure which compares the similarity of different test-patterns, however, here the matching is carried out within the test names.

This scoring function is the basic element of the inexact matching method since it is a means of comparing a single test in the match-pattern to another in the test-pattern of a given fault entry, therefore, the rest of this section will be devoted to it.

Basically, the function compares two *test names* and produces a *score*, which is a number in the range 0-9 which indicates the *degree of similarity* between two tests (9 indicates that the tests are identical). The score represents; **the number of the common terms in the two test (common), relative to the total number of *different* terms in the two tests names (total)**, in a mathematical form:

$$\frac{common}{total} \times scale$$

Scale is the number 9 to make the score in the range 0-9. As with the matching scores of section 5.2.2, this is chosen since it provides a normalised representation for the scores.

Tests mentioned in 1, 2, and 3, see section 10.2.1, are examples of tests which are required to be identified by ***inexactly*** matching them using the technique developed here. The scoring should reflect in its scores the similarities in each of the above cases. The efficiency of the scoring function will be demonstrated through three cases in which the function will be used to compare three common cases in which inexact matching will be required with different scoring in each case.

Case 1

ovp-test A-wire positive diode in diodebridge

ovp-test B-wire positive diode in diodebridge

The above two tests are used to test the protection circuitry in different individuals of the circuit and occur frequently in the data. **Common is 5** and **total is 7**, hence the **score is 6**.

Case 2

For first two tests in 2, section 10.2.1, **common is 3** and **total is 5**, hence the **score is 5**.

Case 3

The tests ***No Offhook*** and ***No Onhook*** are two tests which test the same device for two related functions. Since the two tests test the same device, then in most cases[1] the failure of any of the tests will indicate the failure of the same device.

when the two tests were compared using the scoring function, then applying the definition in the previous section; **common is 1** and **total is 3**, hence the **score is 3**.

The comparison performed by the scoring function is not as efficient as that performed by an experienced diagnostic technician. This is because it is difficult to implement a system which could recognize an indirect relationship between two tests in a manner which resembles that of the expert who develops skills in recognition and comparison of tests based on his/her experience and familiarity with the board circuitry.

From the experience gained by analyzing the data, the tests in case 1 are the ones with greatest relation to each other since they both test the same function except that they are applied on different I/O lines. The score 6 is reasonable since it is less than 9 (identical tests) and higher than average since the two tests are largely related.

The tests in cases 2 and 3 are less related than case 1 with the tests in case 3 being the less related. The scoring function is applied to most of the tests in the data in which a functional relationship exists. When the results are compared with each other, the scores seemed, in majority of cases, representative of the degree of their similarity. In case 3, a diagnostic technician can recognize the relationship between the two tests more efficiently than the scoring function. Since the technician will recognize the common term ***hook*** which is one of the functions tested for in the PCB circuit. Hence a technician might associate a higher score than that provided by the scoring function which cannot detect the above similarity. This is because the function does not examine within the terms of the individual test names. The

[1] If the two tests are not within a large pattern of failed tests, then their failure indicates the failure of the same device.

procedure of adapting the scoring function to perform the above is a complicated task which offers only a little reward (due to the small number of cases in which it is needed). For this reason it is not pursued further in this thesis.

Another drawback of this method is in considering some common terms which have no significance inside test names. Although such cases were not very common because the test names are very brief and are made in such a way that the test task is represented in minimum amount of words, in some minor cases this could have an effect on the matching performance.

10.2.3 The Matching Procedure

It can be deduced from the previous section that one of the critical aspects in performing inexact matching (between two sets of tests) is selecting those tests in one of the sets which do not exactly match the tests in the other set, but are thought to be related (i.e. are inexactly matching tests). One way in which inexact matching could be performed (by a diagnostic technician) is to select exact tests first. Next, the two groups are scanned for tests which are related to each other. The knowledge of the tests and some of the functions of the circuit (developed by experience) are used later to eliminate those tests which are poorly related and the remaining ones are considered as inexact tests. The inexact tests are used together with the exact tests in judging the quality of the match between the two sets.

Based on the procedure above, the matching procedure can be modified such that the scoring function can be used in comparing the tests (in the test-pattern and the match-pattern) during the scanning process. The quality of the score produced for each pair of tests can be used to select those tests considered to be inexacts. As mentioned above both exact and inexact tests are used to evaluate the quality of the match.

From all the above, the overall matching procedure will be similar to that described in section 5.2.2. However, in order to determine the four matching scores, i.e. (S, P, G, F), while comparing the match-pattern extracted from the report with a given test-pattern in the knowledge-base, each test in the match-pattern must be classified into one of the following groups :

Exacts : test(s) which are exactly matched with tests in the test-pattern. Each test must have a score of 9 by the scoring function.

Inexacts: test(s) which are inexactly matched, each test is identified by a score which is in the range 1-8 by the scoring function. Although, the scoring function above allocates scores which could vary in the range 1-8, tests considered as inexacts must have a score which should exceed a threshold value termed the ***score limit***.

Nomatches: test(s) for which no exact or partial match exists. Those tests are identified as the ones which do not fit into either of the above two groups, that is tests which have scores equal to or less than the score limit.

It must be noted that in inexact matching *match-tests* in the match-pattern and the test-pattern are considered to be both exacts and inexacts. *Total-tests* are the individual tests of both the match-pattern and the test pattern i.e.

$$\textit{match-tests} = \textit{exacts} \cup \textit{inexacts}$$

$$\textit{total-tests} = \textit{match-pattern} \cup \textit{test pattern}$$

When comparing a given match-pattern with each test-pattern in the knowledge-base, each test in the match-pattern is compared to each test in the test-pattern. If a given test in the match-pattern exactly matches one or more of the tests in the test-pattern then it is added to the exacts and is not considered any more even if it inexactly matches other tests in the test-pattern. Otherwise, if the test inexactly matches at least one of the tests in the test-pattern then its added to the inexacts. The idea behind such matching is to treat tests as matches even if they are not exactly similar to the tests seen before as long as they share a certain degree of similarity (specified by the score limit). In order to understand how inexact matching is performed, consider the example in figure 10-1. The example illustrates a match-pattern and a test-pattern which are to be matched inexactly. Failed tests in the example are made such that they resemble the actual tests in the data.

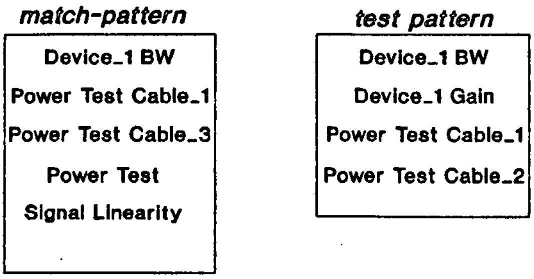


Figure 10-1: An inexact matching example.

With exact matching, match-tests are those with exact matches in the test-pattern, hence:

$$match-tests = \begin{Bmatrix} Device\_1BW \\ PowerTestCable\_1 \end{Bmatrix}$$

However, performing inexact matching without setting any restrictions on the value of the score limit will discriminate the following groups of tests:

$$exacts = \begin{Bmatrix} Device\_1BW \\ PowerTestCable\_1 \end{Bmatrix} \qquad inexacts = \begin{Bmatrix} PowerTestCable\_3 \\ PowerTest \end{Bmatrix} \qquad nomatches = \{SignalLinearity\}$$

$$match-tests=\left\{\begin{matrix}Device\_1BW\\PowerTestCable\_1\\PowerTestCable\_3\\PowerTest\end{matrix}\right\}$$

In the above case, the inexact tests have the scores 4 and 6 respectively. Now, if the score limit is set to be 6, then the three groups of tests are modified such that:

$$exacts=\left\{\begin{matrix}Device\_1BW\\PowerTestCable\_1\end{matrix}\right\}\qquad inexacts=\{PowerTest\}\qquad nomatches=\left\{\begin{matrix}PowerTestCable\_3\\SignalLinearity\end{matrix}\right\}$$

$$match-tests=\left\{\begin{matrix}Device\_1BW\\PowerTestCable\_1\\PowerTest\end{matrix}\right\}$$

The matching scores used for inexact matching are similar to those used for exact matching (see section 5.2.2). However, the scores are modified, where necessary, in order to consider the inexact matches in addition to the exact ones. The new definitions of the four scores are as follows :

S is defined as the number of tests in match-tests.

P is defined as the number of tests in match-tests relative to the number of tests in total-tests.

$$P=\frac{match-tests}{total-tests}$$

***G*, which represents the *Overall Significance*,** is defined as the sum of the significances of match-tests relative to the sum of significances of total-tests, the definition of significance is mentioned in section 5.2.2. In mathematical form.

$$G=\frac{Significance(match-tests)}{Significance(total-tests)}$$

F represents, as before, the frequency count of the test pattern in the knowledge-base.

The utilisation of the above scores in evaluating the quality of a match involves using the same rule-table as that described in chapter five.

10.2.4 Analysis of the Results

Here, the performance of the system is monitored for different values of score limit. The following changes can be observed in the performance of the system :

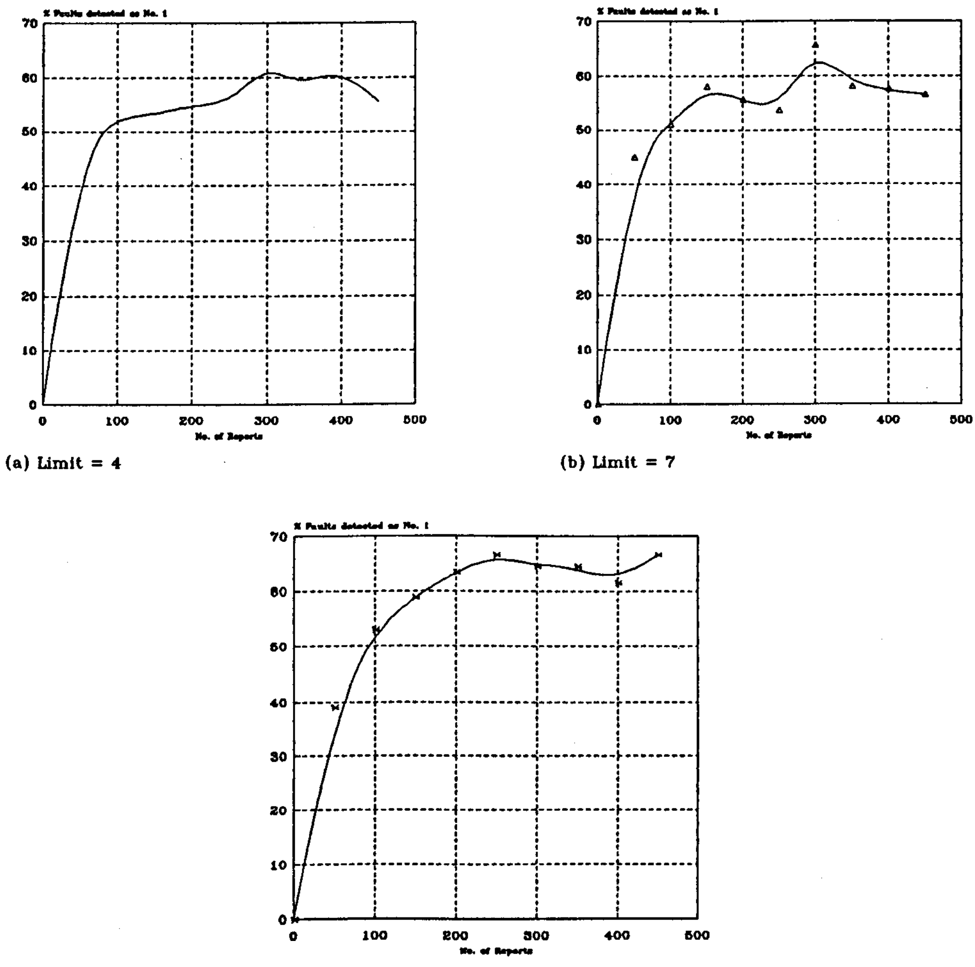
i. Choosing low values for the score limit, i.e. including all the tests in a report with a test score, see section 10.2.2, which is in the range 1-2 as ***inexacts***, the system showed a poor performance at most stages of the diagnosis in comparison with the performance of the original system described in chapter five.

ii. As the value of the score limit is increased up to 4, an improvement is noted especially at early stages of the diagnosis, i.e. during the diagnosis of the first 50-70 reports. The performance achieved with early reports improves on the performance of the original system and on the performance achieved when rule learning was employed, see chapter eight. The best performance with the early reports is achieved when the score limit is 4 where 8% improvement is achieved in comparison with the original system. The performance at later stages of the diagnosis does not match either that of the original system or the system using rule learning. The performance of the system is illustrated in the learning curve of figure 10-2 (a) and (c).

iii. At values of the score limit which are greater than 4, the good performance with the early reports deteriorates slightly, however, it remains better than that of the original system. Figure 10-2(b) illustrates the performance of the system with one of the values in this range, the score limit = 7. As it could be seen from the figure about 6% improvement is achieved in comparison with the original system in figure 10-2(c).

Setting the score limit to be 1 or 2, as in (i), allows those tests which are poor matches to corresponding tests in the knowledge-base to be included as inexacts and hence effecting the matching scores which lead to a deterioration in performance. In (ii) most of these tests with poor scores are not included and hence an improved performance is observed.

The situation in (ii) reveals one of the main advantages of the inexact matching by considering only those tests with a relatively good score (tests with scores over a score limit of 4) as inexact matches. It can be deduced from the learning curve obtained for this situation (figure 10-2a) that in the early stages of the diagnosis procedure, when the system has encountered only a few reports, inexact matching can provide a better diagnosis than the exact matching employed by the original system. However, it must be noted that this improvement is only experienced within the first 100 reports introduced to the system. As the learning curves confirm, the performance deteriorates in comparison to the exact matching and more turbulent learning is experienced. The reason for this deterioration is thought to be as follows.



(a) Limit = 4

(b) Limit = 7

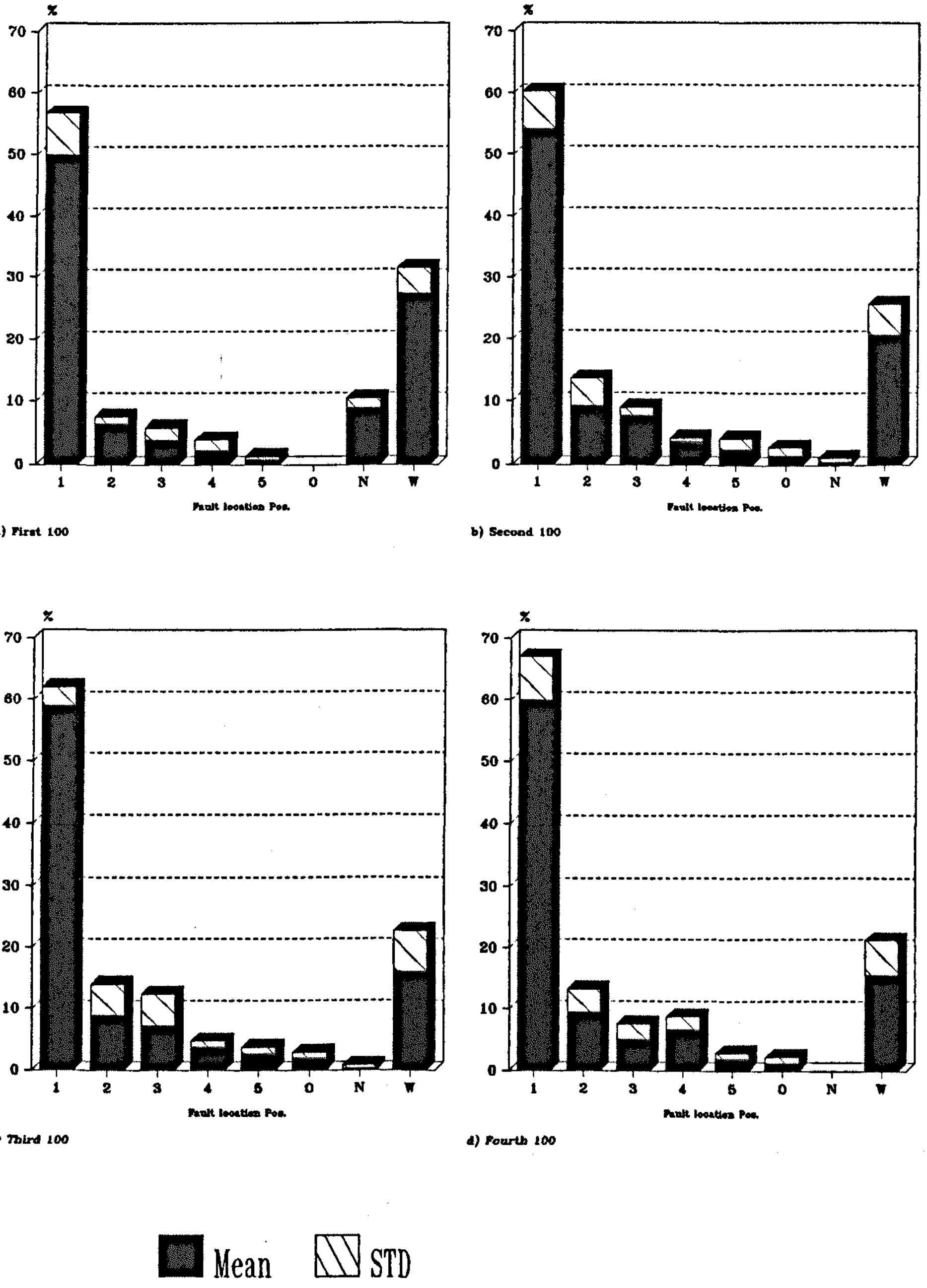
(c) Exact

Figure 10-2: Obtained learning curves.

As mentioned earlier in chapter five (section 5.2.1), most of the reports in the data confirm that the failure of some tests are the characteristics of the failure of certain devices. Such characteristic test sequences were termed match-patterns. The whole matching process is based upon associating a match-pattern on a failure report with previously seen ones. The use of inexact matching involves including some additional tests in match-patterns by including those tests which are similar, but not identical to the characteristics tests in the match-pattern. This will cause the test-pattern to be more general and less specific, we term this as the blurring effect. This means that the test-pattern will no longer be the characteristic of the failure of a single device, but a number of devices. This is because the tests included by inexact matching, although similar, might not point to the failure of the same device. An example is the group of tests within "ovp-tests". These tests are common tests which are used to test different components of the protection circuitry within a given individual in the board. The tests are usually very similar to each other and could provide high scores using the scoring function. If these tests are counted as matches then the correct faulty component could not be recognized since the two *ovp* tests could be for different devices in the protection circuitry.

Hence, the blurring effect caused by inexact matching is hepful at early stages during the operation of the system (when the knowledge-base has very limited amounts of information. At this stage blurring will help the system in identifying suspects since exact matching alone might not provide enough suspects for the system to choose from. However, as more examples are encountered by the system many of the well known test-patterns of the different suspect faults will be known to the system. Hence, when a well known case is encountered the exact matching can identify the fault exactly, but by continuing to use inexact matching the identification will be less specific by including other components with similar likelihood to the real fault (caused by the inexact tests). This will provide difficulty in discriminating the different suspects and hence will lead to a wrong diagnosis.

Figures 10-3 and 10-4 illustrate the performance of the system (at score limit values of 4 and 7) using bar charts. The charts confirm the information obtained from the learning curves in that the number of faults identified in the 1st 100 is more significant when the score limit is 4. In addition, the charts carry information about faults diagnosed with less likelihood by the system. One of the observations made during experimentation with inexact matching is the relative increase in the values of std compared to previous bar charts obtained when exact matching and rule learning were employed. This could be noted in the bar charts. The increase in std values point to the fact that the percentage fault detection values, obtained in the different random trials during analysis of system performance, show more variation compared to other methods. The increase in the values of std were investigated by varying the number of random trials over which mean and std values are calculated, however, no significant changes were noted.



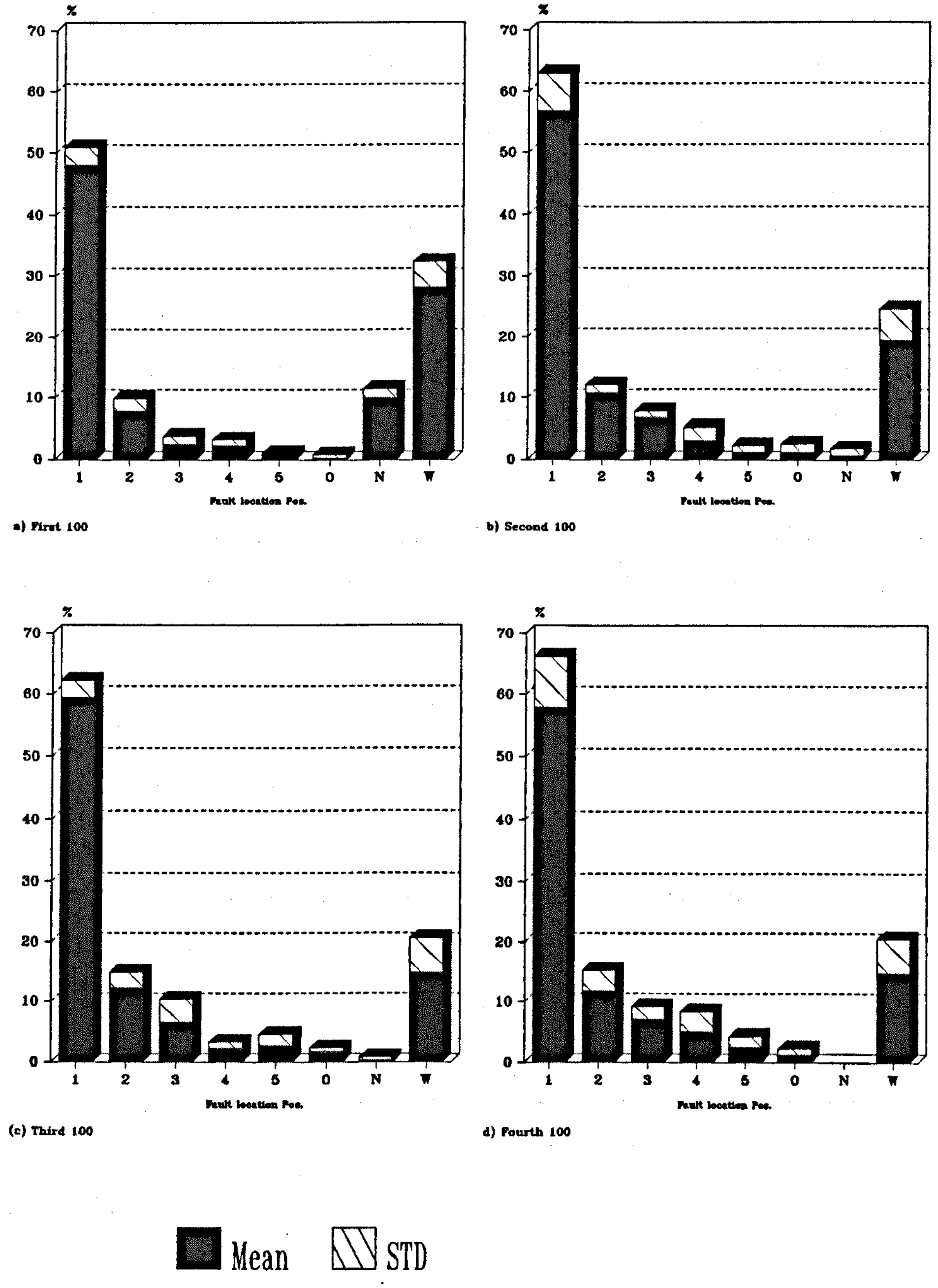
a) First 100

b) Second 100

c) Third 100

d) Fourth 100

Figure 10-3: Mean fault identification charts, score limit = 4.



a) First 100

b) Second 100

(c) Third 100

d) Fourth 100

Figure 10-4: Mean fault identification charts, score limit = 7.

10.2.5 Investigating Alternative Approaches

This section describes further investigations which are carried out in order to find alternative ways of performing inexact matching. One approach which was investigated involves calculating an additional score representing the quality of inexact matching. This score is used with the other matching scores, which unlike the previous approach, are not altered here.

The additional score is a number which reflects the number and quality of the inexactly matched tests in the match-pattern with the test in the considered test-pattern in the fault entry.

The score is calculated using the scoring function, described in section 10.2.2. A match-pattern is compared with a given test-pattern and the inexactly matched tests are identified by their scores (scores 1-8). The scores of these tests are added together to form an integer number. Later, a final score is obtained by normalizing the value previously obtained by dividing it by the number of inexact tests. The *normalization* is useful in comparing scores obtained for a given match-pattern. As in the original matching technique, the process is repeated by comparing the match-pattern with all the patterns in the knowledge-base. Hence, each suspect is evaluated with five scores (i.e. the above score and the previous four).

Since each score value obtained by the scoring function is an indication of the quality of the match and since it is added for all the inexact tests in a single report when compared to a pattern, and also it is performed for each pattern in the KB, therefore, the final score for the pattern should be proportional to the number and quality of the inexact matches in the pattern (as compared to the report).

This overall score was just observed on the debugging system output, to see whether an improvement is possible to the matching result. However, by examining different set of results it was clear that this additional score is not effective since it was significant in test-patterns of the wrong faulty component and vice versa. Hence, the diagnosis system was not adapted to use this fifth score.

A more comprehensive inexact matching is to start breaking the single terms inside the test names and try to find some more common terms in them. Since the failure of tests such as:

false xdata
false bdata
false zdata

indicate the failure of the same device, however *xdata, bdata, and zdata* will be completely different terms for the method in this chapter. The task of looking inside the terms in test names require certain amount of time and effort computationally and manually and this is not advisable considering the small amount of tests in which such action is required. The success of this method is not guaranteed since as seen from the results of this section, the more similarity is searched in side the test names the less is the performance of the system.

Another method could involve including the names of different tests which cause the failure of the same component in the body of the program. This method can allow us to group the tests which are observed in the data commonly as being similar types, although they have different names. However, this will make the system usable only with circuit boards which have similar components and are tested with the same test program, i.e. the system will not be a general fault finder which can be used with different circuit boards. The system without using the specific tests in the program, is a general one since if the knowledge base is cleared and new faults and patterns are introduced to the system, then after a certain period of time, when the KB has enough data, the system will start recognizing the faulty components.

10.3 Utilization of Measured Parameters

10.3.1 Introduction

Matching utilizes only a portion of the information provided on a report about the failure of each test. Specifically, matching mainly considers the test names with individual numbers and test numbers being used external to matching. Although such use of matching was decided after thorough investigations using both test names and conditions, such investigations never included the measured parameters on the failure report due to their non-discrete nature which makes them unsuitable for a process such as matching.

Measured parameters include parameter values measured during the testing procedure. These are different for different tests, examples of these parameters are gain, voltage, and current values. It was thought that such information could assist in boosting and analyzing the performance of the system. This chapter describes the modifications performed into the knowlege-base in order to incorporate measured parameters. Later, the chapter investigates a method for utilizing these in the diagnosis procedure.

10.3.2 Modification to Knowledge-base Structure

In order to adapt the system such that measured parameters can be utilized, the knowledge-base structure had to be modified for accommodating such information. However, prior to reconstruction of the knowledge-base a number of tools were developed for analyzing these parameters. Such tools included examining the range of values, if such a range exists!, regarding the failure of each test for a certain faulty component and later comparing the result with the failure of the same test in a different component.

Different analysis tools such as the one above revealed that for most tests in the data; if the measured values of each test are considered individually, i.e. irrespective of the whole test-pattern of which the considered test is a member, then no clear range of values emerge. However, if the tests are considered within identical or similar test-patterns then in most cases a range could be determined by examining the values of these tests. This is expected due to the significant important of test-patterns in the matching process. It must be remembered that the original matching technique is developed in order to identify sequences of tests, i.e. test-patterns, rather that single tests due to the common occurrence of specific patterns for different component failures. In addition, it is thought that different test-pattern failures for a given component indicate different circumstances (conditions) of failure and hence the result of the analysis is logical.

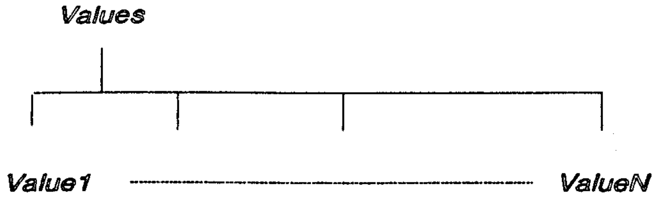


Figure 10-5: The added structure for storing values.

For the reason above, the new knowledge-base structure stores measured values of each test separately for each different test-pattern seen in a given fault. The new knowledge-base structure is similar to that of figure 5-1. The only change is to the structure of the test-pattern which is made to store each value of each test for any test-pattern encountered by the system. A structure such as the one shown in figure 10-5 is added to the original test-pattern structure for storing the values.

10.3.3 Investigating the Diagnosis

Although measured parameters cannot be matched directly, they could be used such that they effect the outcome of the diagnosis process. For this reason the method investigated is developed such that for a given match-pattern and a test-pattern a score (or a value), termed ***measurement score*** (S_m), is developed which reflects the amount in which each test in the match pattern varies from the values stored in the test-pattern within the knowledge-base. Hence, it is assumed that a more suspicious fault will have a smaller score for its values. It must be remembered that this score should be considered with the other matching scores discussed in chapter five.

Supposing that a match-pattern $\{T_1, T_2, \ldots, T_N\}$, which consists of N tests, is compared to a identical test-pattern in the knowledge-base. S_m for such a pattern is calculated as follows :

$$S_m = \frac{1}{N}\sum_{i=1}^{N} S_i$$

Where

$$S_i = \frac{1}{n}\sum_{j=1}^{n} | V_i - \hat{V}_{ij} |$$

N is the number of tests in the match-pattern (and test pattern).

V_i is the measured value of the ith test in the match-pattern.

$\hat{V}_{ij}$ is the jth seen measured value for the above ith test stored in the test-pattern of the knowledge-base.

n is the number of seen measured values for a given test in the test-pattern.

Consider the following example in which a match-pattern consists of the tests $\{T_1\ T_2\}$. Supposing that the measured values of the tests in the match-pattern are such that V_1 is 0.5 and V_2 is 0.2. The corresponding vector of values in a given test-pattern of the knowledge-base is such that $\hat{V}_1$ is $\{0.4\ 0.45\}$ and $\hat{V}_2$ is $\{0.25\ 0.2\}$. The calculation of S_m is as shown below :

$$S_1 = \frac{| 0.5 - 0.4 | + | 0.5 - 0.45 |}{2} = 0.075$$

$$S_2 = \frac{| 0.2 - 0.25 | + | 0.2 - 0.2 |}{2} = 0.025$$

$$S_m = \frac{0.075 + 0.025}{2} = 0.05$$

10.3.4 Results

The system was modified such that the new value score is calculated for each suspect fault (with the previous four matching scores). The results obtained by the system were analysed by examining the effect of the value score in comparison with the rest of the scores. The aim of this analysis was to see whether the new scores could recover the cases where the other four matching scores fail, i.e. the aim was to see whether any additional information is obtained from these new scores. If the analysis here points to a possible improvement in performance then, as with the previous scores, the rules in the rule-base can be modified to include the new score.

However, the analysis indicated that the value scores obtained mainly confirm the cases which could be distinguished by the previous scores whereas in the cases where the four matching scores fail to distinguish the correct fault the new score carries no additional information. For this reason the score is ignored and is not incorporated into the matching rules.

Analyzing the data provide some indication as to the failure of using measured parameters which is summarized by the following. It was clear from the results of chapters five and six that matching, using the four scores, could discriminate most of the faults by their common test-patterns, however, failure occurs mainly with new, less common, test-patterns. From the analysis it is clear that the number of measured values for these less common cases are small, which is expected. For this reason a reliable score could not be determined especially if an out of range value exists, i.e. noise. In the cases where a common test-pattern exists for more than one fault, the value score shows a contradiction to most of the other four scores which are more reliable.

For this reason the value score is ignored for the time being and is suggested to be re-introduced as more data are provided, which will lead to more abundance of values per test in the knowledge-base. It is also useful to investigate a different method of involving the measured values in the diagnosis procedure.

The main advantage of this work is in the adaptation of the knowledge-base for the accommodation of the measured parameters since they represent valuable information which must be remembered in case of future modifications to the system. Furthermore, the system knowledge-base represents a source of information for the company concerned and hence could be analysed by the company periodically in order to monitor their manufacturing procedure. For this reason it is more efficient if most of the failure information is incorporated to the system.

10.4 Conclusion

This chapter detailed the work involved in carrying out two additional investigations which aim to improve system diagnosis by performing a more comprehensive matching.

First, the use of inexact matching was investigated. Tests are considered to be inexactly matched if one or more terms exist which are common in their test names. Hence in this case test names do not have to be exactly identical in order to be considered a match.

In order to perform inexact matching a scoring function was required in order to evaluate the quality of inexact matching among different tests. The scoring function is used in matching tests of a given match-pattern against the various test-patterns in the knowledge-base. The original matching scores developed in chapter five are modified in order to include inexact matches.

It was noted that for best inexact matching results the score generated by the scoring function for a given pair of tests should be greater than a certain limit in order to avoid including poor matches.

Results obtained with inexact matching indicated an improvement in system performance at early stages of the diagnosis in comparison to exact matching. However, this improvement is not sustained as more reports are introduced to the system.

The improvement achieved by inexact matching at early stages of the diagnosis is an important feature which should be employed efficiently. For example, *the use of inexact matching could be restricted to early stages in diagnosing new boards.*

The second investigation described in this chapter concerns adapting the system such that information about measured parameters can be manipulated by the system. In order to process these measured parameters, the knowledge-base had to be modified such that this information can be accommodated. Next the effect of using these parameters in the diagnosis procedure was investigated, however, no significant improvement was achieved. One of the reasons for this is thought to be due to insufficient amount of information (including measured parameters) especially for cases where improvements are required, since most of these cases are cases which occur less frequently. One of the reasons for the deficiency of measured parameters per test is due to the new structure of the knowledge-base which stores measured values of similar tests only within similar test-patterns. Although such a structure reduces the number of values per test, investigations showed that it is an efficient way of storing values since most common faults can be identified with a number of test-patterns each of which is caused by different condition of failure and hence the values of similar tests within different test-patterns are mostly inconsistent. It is thought that efficient use of measured values could only be made when sufficient number of values exist for the tests within their test patterns. This requires additional examples of the failure patterns currently in the knowledge-base.

The structure developed for the knowledge-base is maintained even though measured parameters are not actually utilized in matching. This is because if such information is stored then there is a possibility of using them in the future when more information is accumulated in the knowledge-base. Furthermore, the knowledge-base acts as a source of information which could be analyzed for monitoring the actual manufacturing process for these boards, i.e. frequency of some faults might point to a certain malfunction in the actual manufacturing process. Hence it is desirable to store as much as possible of the information about the failure cases since this will enable a better and more thorough analysis.

CHAPTER ELEVEN
DIAGNOSIS OF MULTIPLE PCB STYLES

11.1 Introduction

The next stage in the development of the system is to examine its performance when used in diagnosing failures of different PCB styles, i.e. boards with variation in their design with respect to each other.

This chapter describes the use of the system in diagnosing two additional PCBs. The first of which is similar to the PCB used in the previous chapters. The second, however, is completely different.

For each board, this chapter describes the differences in design and in the failure report format pointing out, in each case, the results obtained in comparison to those obtained with the initial board. The Chapter concludes by describing the required modification to the system in order to operate in a multiple board environment (MBE).

For simplicity, the rest of this chapter will refer to the three boards used according to the following :

PCB_1	The initial PCB used in the previous chapters.
PCB_2 & PCB_3	The new boards, which will be explored here.

11.2 Board PCB_2

11.2.1 Characteristics

This board resembles PCB_1 in the modularity of its circuit design and the functionality of its circuit, i.e. the task it performs. The difference in this board is mainly represented in the addition of some new components and the removal of others from the circuit.

In addition, the format of the failure report for this board is similar to that of PCB_1 in that the information mentioned about the failure of each test in the report is identical; including parameters such as test number, test name,. . .etc. (see section 5.2.1.1). This similarity is due to the use of a similar ATE controlled by a similar test program.

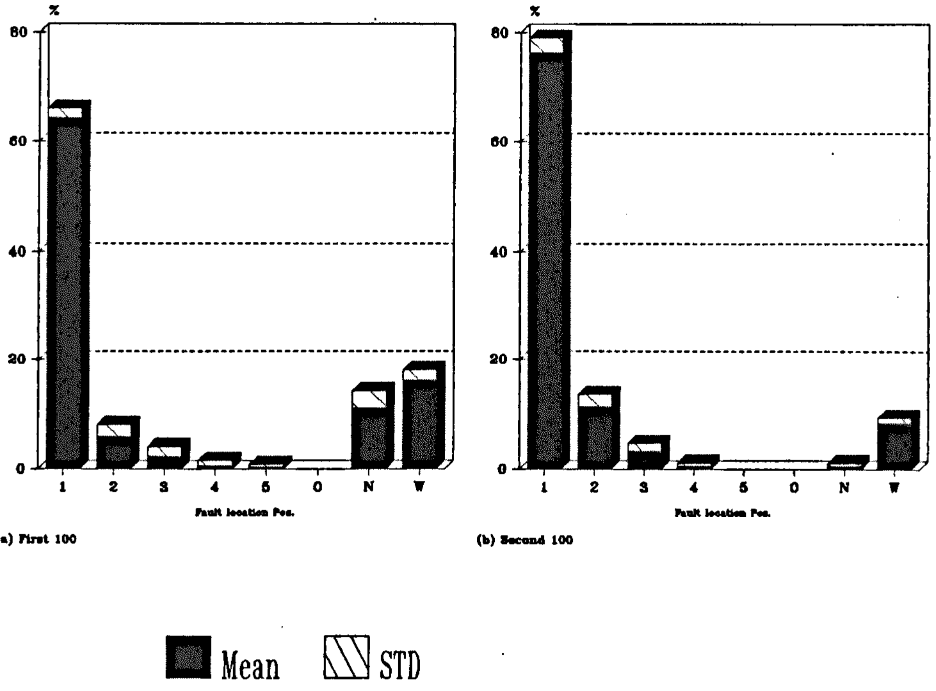
Due to the similarities between this board and the initial one, a knowledge-base with an identical structure to that described in section 5.2.1.1 is used. Since the matching procedure and the matching rules are implemented to cope with various report formats (providing enough information exists for matching), therefore, the same matcher and the same rule-base format are employed.

11.2.2 Results and Analysis

In excess of 200 failure reports were provided for this board. These include reports diagnosed to be due to a fault in some of the newly added components.

Due to the similarities in this board with PCB$_1$, the results are obtained in two stages as described below :

1. First, the reports of PCB$_2$ alone are presented to the system for diagnosis. The results are presented in charts similar to that used in chapter five and are shown in figure 11-1.

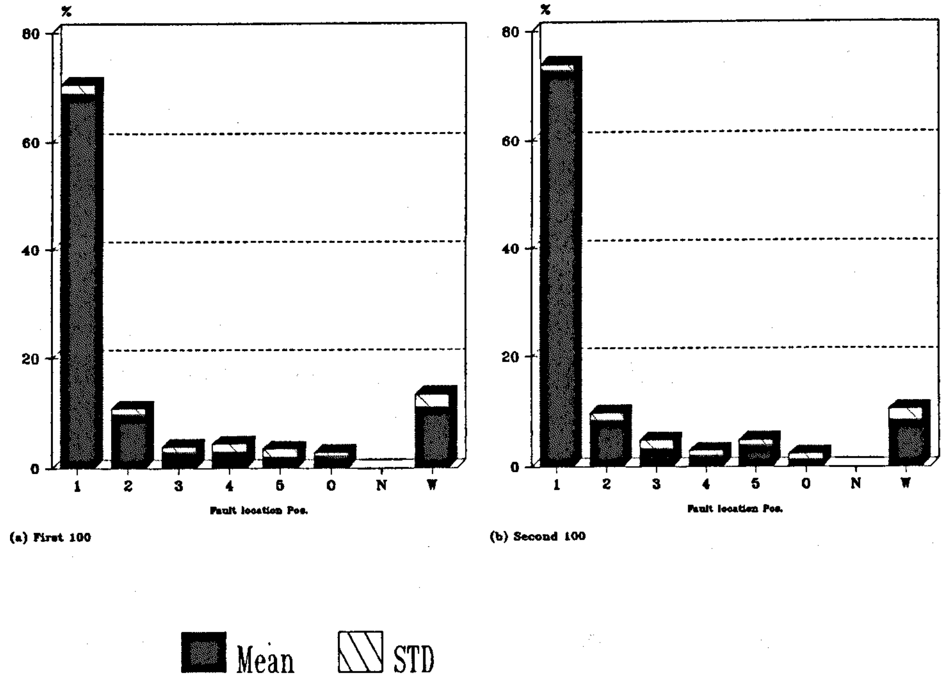


(a) First 100

(b) Second 100

Figure 11-1: Results of PCB$_2$ alone.

2. Next, a knowledge-base is constructed using the reports available for PCB$_1$ (approximately 500 reports). Later, the reports of PCB$_2$ are presented for diagnosis and the performance is monitored. The results are illustrated in figure 11-2.



(a) First 100

(b) Second 100

Figure 11-2: Results of PCB_2 with PCB_1

Note that the charts of both figures 11-1 and 11-2 conform to the same axes of the charts for chapter five and each chart illustrates the mean and std obtained over a number of random trials.

The results obtained in (1), see figure 11-1, are slightly better than the corresponding ones obtained with PCB_1 for the first and second 100, see figure 5-6. This improvement can especially be noted in the faults diagnosed as the first best fault. It can be seen from the charts of figure 11-1 that approximately 64-76% of the reports are identified if position number one alone is considered while 72-90% of the reports are identified in considering the first three positions. One of the reason for the improvement in performance with PCB_2 is because of the consistency of the reports in the data. This consistency could be the result of adjustments in the manufacturing procedure by the company.

Considering the results obtained in (2), see figure 11-2, the number of reports diagnosed correctly in the first three positions remain high, however, the pattern of fault detection in the first and second 100 vary as follows.

During the diagnosis of the *first 100*, the number of faults diagnosed in the first position (and the first three) increase by about 5% (9%). This is expected since the reports of PCB_1 provide the knowledge-base with enough information to recover some of the regularly occurring faults which repeat in both boards. Hence the system can identify these reports even in the early stages of diagnosing PCB_2. Whereas during the same stage in (1), these reports are still new, although they are frequent cases, early in the first 100 and hence some of these cases are wrongly diagnosed, however, update informs the knowlege-base of these cases hence leading to an improvement in diagnosis later.

The diagnosis of the *second 100*, however, show a slight deterioration in performance compared with corresponding one in (1). The deterioration is about 3% in the first position (6% in the first three positions). In order to understand the reason for this deterioration, the individual matches in 1 and 2 are compared and analysed.

The above investigation revealed that this deterioration in performance is mainly due to the components which are added to PCB_2, as replacements to other components or circuitry in PCB_1. In a number of cases which involved diagnosing a newly added component, the system indicated the old component as the main suspect with the new component in a position other than the first. It was found that this is especially the case if the report is a common symptom for the failure of the two components in both boards. In such a case the scores of the removed component, especially the frequency count, are better than the new component and a relatively large number of reports are needed to shift the high likelihood of the old component to the new one.

In very few of the cases, the deteriorations could be due reports which contain test-patterns common to more than one fault with the frequency of it occurring being shifted in the reports of PCB_2 from that of PCB_1. However, as mentioned earlier these cases are less frequent and could be due to the small number of reports for PCB_2 relative to that of PCB_1.

The procedure in (2) tests the efficiency of the system in coping with changes in the design of a given board which has been previously diagnosed by the system. In practice, it is common to modify the design of the boards (in the company) by either the addition or the removal of some components. The results above indicate that the system can cope with the changes between the design of PCB_1 and that of PCB_2. This is because the system maintains its high failure identification rate. However, equipping the system to cope more effectively with changes in board design require new modifications to the system. Such modifications include the addition of a mechanism for removing information, related to the removed components, from the knowledge-base (temporarily or permanently). This mechanism will be discussed further in chapter thirteen.

11.3 Board PCB_3

11.3.1 Characteristics

This board can be distiguished from the other two, PCB_1 and PCB_2, in the non-modularity of its circuit design. This means that unlike the other two boards, in which the circuit is divided into blocks of components, this board consists of a single block of circuitry.

The board is tested by an ATE which is different from the one used in testing the other two. The format of the failure report also varies from the other two in that it lists additional parameters about each failed test such as; The test points in the circuit and the line number in the ATE program at which the test failed. However, the tests used by the ATE for testing this board are not classified according to the function they perform in the circuit, as in the other two boards. Hence, test numbers are not included within the parameters listed in the report about each test.

In view of the above changes in the parameters presented about a test failure from the previous two boards, the only change necessary to the system is to the representation of a single test in the knowledge-base. The changes are performed such that the new parameters are considered and the old ones are ignored.

11.3.2 Results and Analysis

The number of reports available for this board is significantly less than that provided for the previous two boards. In total approximately 140 reports were provided by the company concerned.

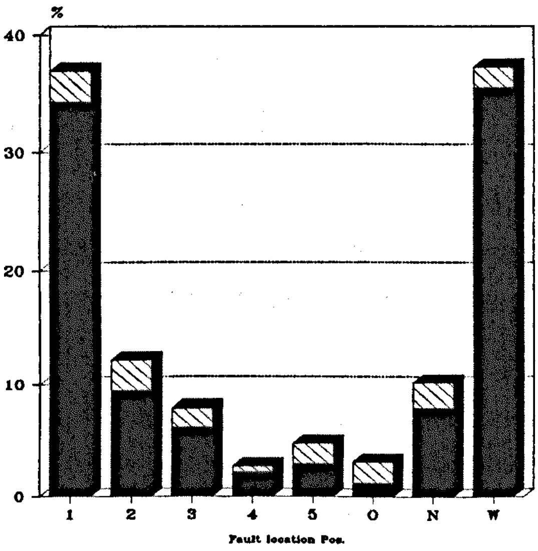


Figure 11-3: Results of PCB_3

After performing the necessary modifications to the system, in order to accomodate changes in report format, the performance is tested with the new data. The results of a number of random trials are illustrated in figure 11-3.

The figure indicates a significant deterioration in performance compared to the results obtained with the previous two boards. Investigation of the performance is performed by thorough examination of the results. The investigation lead to a number of factors which could cause the deterioration in performance. These are listed below :

1. *Lack of sufficient number of examples*

   The number of reports obtained for this board is smaller than the reports obtained for both PCB_1 (~500) and PCB_2 (~200). However, the deterioration in performance can be noticed even if the results of the first 100 of the three boards are compared.

   The number of examples could still be the main factor for this deterioration in performance although similar number of reports of each board are compared. This is because of the modularity of the circuit design in the case of PCB_1 and PCB_2.

   The reason why modularity of the circuit in these boards help in multiplying the number of examples for them is due to the following. Each module represents identical circuitry consisting of identical components to the other modules. From our basic understanding about the operation of the circuit, these modules operate in an identical manner to each other and hence during the testing procedure, similar tests are applied to each individuals which involves operating the individual circuitry under similar conditions. For this reason, examples regarding the failure of a given component in a given module could be used as an example of failure of the corresponding component in any other module. Thus examples of components of the same type in different modules are used together in diagnosis and update, hence leading to quicker learning.

   For PCB_3 however, examples of similar components cannot be used, since they are part of different circuitry responsible for carrying out different tasks. For this reason when these components are tested they are not tested similarly and hence cannot be used as examples of each other.

In order to illustrate the effect of circuit modularity on the diagnosis procedure, consider the following analysis performed with PCB_1 (since this is the board which initiated the work in this thesis and it has the largest amount of data associated with it).

In general, the circuit of PCB_1 consists of 21 Integrated Circuit (IC) chips and approximately another 450 devices (which include relays, rectifier bridges, thyristors, inductors, fuses, resistor networks, resistors, and capacitors).

The reports associated with this board were presented to the system in order to generate a knowledge-base to assist in the analysis. Examination of the information in the knowlege-base indicated the following. The knowledge-base includes failures of approximately 200 components (42% of the total components of the circuit) which includes all the ICs in the circuit. Most of the components which have not failed at all are either resistors or fuses which belong to the control circuitry or the power circuitry.

The analysis also revealed that the above faults include a total of approximately 210 different failure conditions (situations). This is deduced from the number of test-patterns associated with each fault in the knowledge-base (since test-patterns are the main objects of identifying faults in different components for the system. In addition, it is thought that different test-patterns associated with the failure of a given component indicate different situations under which the component fails). The failure conditions above ranged from approximately 60 such conditions for some of the complex ICs to 1-2 for resistors and capacitors. The use of test-patterns in order to deduce the failure conditions of the different devices point only to those conditions encountered so far. However, it is more accurate than other ways of deducing the number of failure conditions for the different devices. This is because most of the components in the circuit are analogue and hence could have a very wide range of states at which they could fail. Furthermore, many of the ICs perform a wide range of tasks some of which are not provided for the technicians or the engineers. For this reason, details of all the conditions of their failure cannot be deduced.

The above knowledge-base stores all information related to the failure of similar components in different individuals together. This assists the system in making a better diagnosis. In order to appreciate this consider the following. Approximately 192 of the faults in the knowledge-base (96% of the total faults in the knowledge-base) are that of components in the individual circuitry. By treating faults in similar components together. The system reduces the number of faults to 24 (i.e. 1/8th of the 192 faults).

Hence, examples in excess of 210 failure conditions are used to identify approximately 30 faults (including those which are not in the individuals). However, if modularity was not used then the same number of failure conditions would be used to identify 200 faults.

2. *Insufficient number of tests*

   Analysis performed on the data for all the boards revealed that the overall number of tests available for this board is significantly less than that of the other two boards. This factor is significant since all of the circuits are of similar sizes.

3. *Short report length*

   It is noted that reports of this board include the failure of smaller number of tests in comparison with the other two boards. The *average* number of tests failed per report is 1-2 tests, in comparison with 6-7 tests for PCB_1 and PCB_2. The maximum number of tests failing in a report for this board is 3 in comparison with more than 50 tests failing in a single report for the other two. The small number of tests per report could be due the test strategy incorporated in the ATE program, since some test programs could be made to halt after the failure of the board for one or more of the tests. The idea is to concentrate the attention of the technician to the first (or first few tests) as they are more likely to direct the diagnosis to the correct faulty component. However, as experience with different boards show, this is not always the case. Since in some cases the final failed tests point out to the correct fault (as in multiple individual failure, see chapter six), whereas in others a larger group of tests are required for finding the best test pattern during matching in the knowledge-base.

   The short report length added to the small number of the overall tests, mentioned in 2, allow for a smaller test-patterns. Since our diagnosis technique is mainly based on storing test-patterns and recognizing by matching these test-patterns and since the number of possible faults are approximately equal to the other two boards. Therefore, only a small number of test-patterns are possible, for this board, in order to recognize same number of faults as the other two boards.

11.4 Operating in a Multiple Board Environment

So far the system has been used separately for each board. The modifications required by each board were performed to the different parts of the system, such as the KB, by treating the system to be specific for diagnosing the board under consideration. However, in order to operate in a MBE a single system is required which recognizes each board style and performs a diagnosis appropriately.

This section considers the necessary modifications in order to enable the system to recognize and diagnose boards of different design tested by different ATEs.

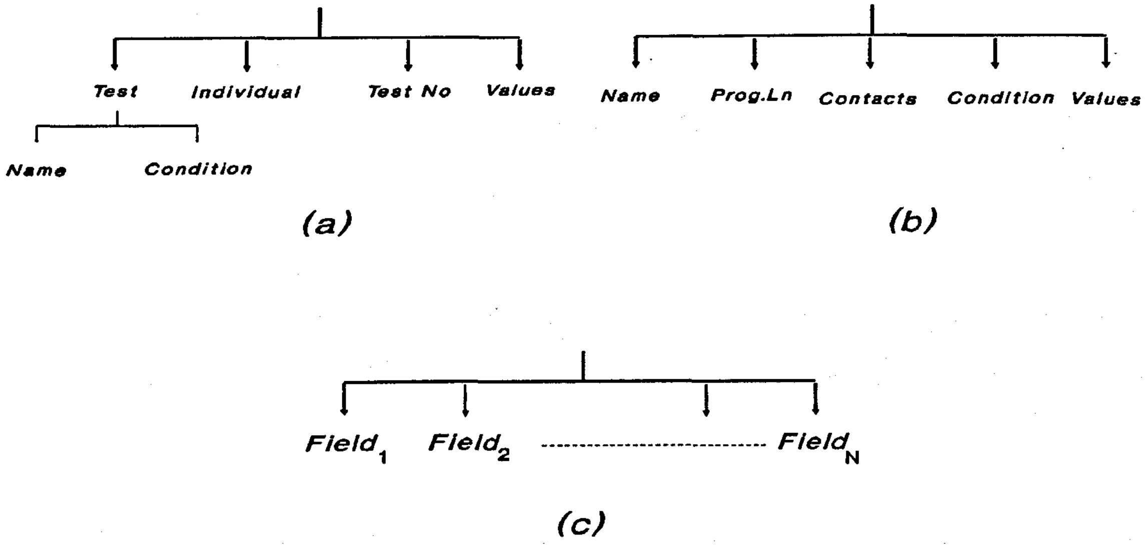
11.4.1 Modifications

Currently, the system must be informed (by the test engineer) about the specific field (in the failure report) which should be used in the matching procedure. As mentioned in chapter five, the best field for matching was found to be the *test name*. This was deduced by manually informing the system about each field to be used in matching and comparing the performance with the different fields. In a MBE, the system will be required to deal with reports which are related to the failure of different PCBs produced by different ATEs. For this reason, the information in a failure report could vary depending on the test program running the ATE, such as the type of test program e.g. functional. The above points to a need for automating the process of selecting the field in the report which is to be used in the matching procedure.

As it was seen in the previous sections, the variation in the report format is mainly caused by the difference in the information provided by different ATEs about each test failure in a given report. Considering PCB_2 and PCB_3, for example, the failure report for PCB_3 include information such as test points and line of failure in test program, which are not provided for PCB_2. Another difference in the reports of the two boards is caused by hierarchical classification of tests, by the ATE program of PCB_2, according to the functionality of the tests. This lead to including information such as test group numbers in the failure report.

The above represents some of the features in which reports produced by different ATEs differ from each other. For this reason efficient automation of the field selection procedure (mentioned above) for different reports can only be achieved by modifying the representation of a single test within the system. This representation will be employed during matching and in order to store the appropriate information about each test in the knowledge-base. In order to deal with different report formats, a general test representation, GTR, is required which can be used for dealing with maximum number of test representations in different failure reports. Due to our limitations to the board styles provided, GTR is chosen to best suit their reports.

GTR is required to be simple avoiding the nested structure chosen for PCB_1 and PCB_2, shown in figure 11-4(a), and more like the structure used for PCB_3 as seen in figure 11-4(b), i.e. flattened, however in a more general form such that any number of test parameters can be incorporated within it. The structure chosen for GTR is shown in figure 11-4(c). The different test parameters can be allocated to one of the *fields* shown in the figure. This structure offers simplicity and efficiency during manipulation for ease of access to any of the parameters.



·Figure 11-4: Test representations.

11.4.2 Use of Circuit Connectivity

We demonstrated in chapter seven, how the use of basic circuit connectivity information was employed in the diagnosis of cases which could not be recovered using matching alone and hence assisting in boosting the performance of the system. The routines developed then are general and can be used with different circuit designs.

This chapter makes further use of connectivity in order to assist the system in dealing with boards of different design.

The circuit compiler, described in section 7.2, can be used to introduce any new board to the system by creating a record for it and storing it with records of all the known boards, introduced earlier. When boards are modified in design, they could either be recompiled to replace old design or be introduced as a new board, this is helpful if the old design remains in circulation (use) since the possibility of maintenance repairs exist.

In addition to introducing new/modified boards to the system, connectivity could also be used to inform the system of the features of a board currently under consideration. Hence, when a report indicates a failure in a given board it is presented to the system. This is performed by finding the appropriate board record and placing it in the current pcb buffer, see figure 11-5.

Some connectivity handling routines were developed which inform the system of specific information about the board design such as; whether the board is modular or not, whether a certain device is heavily interconnected in the board. Such information assists the system in the following manner:

1. Extracting an appropriate match-pattern, see section 5.2.2, from the report. This is especially effective when multiple module/component failures are reported.
2. Providing the different connectivity manipulation routines with the appropriate information required about the circuit of the board under consideration. Without connectivity information, a number of specific routines will be necessay to deal with specific cases of each board. As new boards are introduced to the system new routines must be added to its program in order to cope with the board. Hence, providing connectivity information to more general routines will assist the system to be used with multiple boards and hence a more general system is produced.

Figure 11-5, illustrates the system components required in order to cope with diagnosing more than one board design. The system operation can be summarized as follows.

When a report is introduced to the system, the pcb type is identified from the information in the report and hence the appropriate PCB information is occupied in the *current PCB* buffer, this will also lead into activating the appropriate knowledge-base (KB) and rule-base (RB). Information about the basic features of the PCB (e.g., Type and number of modules) are used to extract a match-pattern. Matching uses KB and RB to identify a list of suspects. Appropriate connectivity analyses are performed where needed by identifying required features in the suspects/report (see chapter six).

Each time a new PCB is introduced to the system, its circuit description is entered in the appropriate format (currently, is a Lisp list format - similar to EDIF and is readily translatable to a standard format) using an editor. The compiler checks for basic syntax errors and puts the circuit in a form which is easy to manipulate by the system. Some of the connectivity analysis are performed at the compilation stage to avoid delay during fault diagnosis.

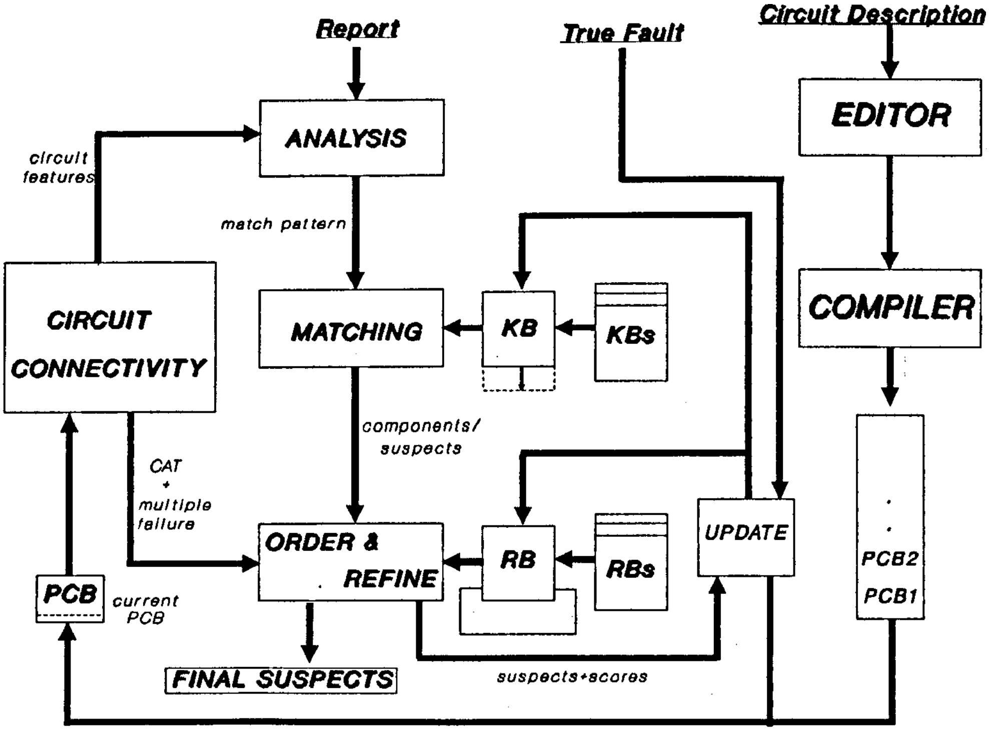


Figure 11-5: The full system operation in a MBE.

11.5 Conclusion

This chapter further examines the performance of the system developed in the previous chapters by using reports which are associated with the failure of two additional PCBs.

The two boards are different in their variation from the initial PCB, i.e. PCB_1. The first board is slightly different from PCB_1 while the second one is completely different from it.

Results show that the performance of the system with PCB_2 matches (and is even slightly better than) earlier results obtained with PCB_1. In addition, experiments were performed by presenting reports of PCB_2 to the system after constructing a knowledge-base with the reports of PCB_1. The results indicated that the system can cope efficiently with limited changes in board design. This is one of the advantages of the system since, in practice, boards are commonly modified in their design. This point is pursued further in the final chapter.

However, significant deterioration in performance was noted with the final board, PCB_3. Investigations showed that this deterioration could be due to number of reasons such as; insufficient number of tests to distinguish test-patterns for different faults and lack of examples.

The chapter concludes by describing the required modification to the system in order to operate in a multiple board environment mentioning among many things further use of connectivity information.

Although our experimentation is restricted only to a system which creates/uses a separate KB and RB for each different board style (even if the boards are similar to a certain extent like PCB_1 and PCB_2), it is recommended in the future to utilize connectivity more by using it to monitor similarities in different boards and using such information as a guide in sharing appropriate information from their knowledge-bases. For example, if a new board (which has a number of common components and modules with a board which is known to the system) is introduced. Then the system would be able to recognize the common components and modules and use the appropriate information already available for the old board in diagnosing failures in these components with the new board. This is useful since it makes efficient use of gained information and is especially significant where insufficient information exist regarding the new boards (especially at earlier stages of introducing the board to the system).

The above suggests the sharing of information regarding different boards in a controlled manner, decided by connectivity. In this way, the system can detect components which are replaced in one of the boards by others corresponding ones. This will avoid the deterioration in performance caused by such replacements, as it was noted when diagnosing PCB_2 reports with a knowledge-base initially built with reports of PCB_1.

The system could be used in diagnosing a limited number of boards. This is due to the difficulty of obtaining information (failure reports + Design information) about PCBs, since companies restrict the access of their designs and their operation strategy to the outside world. However, it will be interesting to examine the performance of the system with boards and test data produced by different companies in the future in order to compare the performances.

CHAPTER TWELVE
FINAL CONCLUSIONS AND SUGGESTIONS FOR THE FUTURE

12.1 Summary and Main Conclusions of the Thesis

The main goal of the work described in this thesis was to enable the utilization of the failure reports in the data, and those continuously produced by the ATEs, in the actual failure diagnosis procedure. The aim being to increase the efficiency of the diagnosis procedure on the factory floor by avoiding the loss of information, since technicians cannot remember failure cases specifically. In addition, the utilization of the data in the diagnosis process can reduce the diagnosis time by avoiding the delay introduced by manual probing of the failed PCBs.

Initially the use of a standard machine learning technique was investigated in processing the reports. The technique analyses the failure reports to produce a decision tree. The analysis is performed by clustering then making generalizations by similarity detection. However, this technique had a number of drawbacks which questioned its suitability for a diagnosis system based on the analysis of test data used here. One of these drawbacks is the failure of the technique in completing the decision tree when an identical report is shared by more than one of the faults. Due to the common occurrence of such reports in the data a technique was sought which could deal with these cases automatically with the rest of the data and does not require their manual isolation/manipulation (by test engineers) as with the construction of a decision tree. Another drawback is in the diagnosis procedure which involves searching down the tree for a recognition function which includes the report. Our experimentation with this technique revealed that this procedure is effective only if the report is identical or slightly different from the example reports used in constructing the tree. Furthermore, the tree could be out of date with the introduction of new examples and will require re-building. This is not practical since the construction of the decision tree takes a relatively long time (this will increase with the increase in the number of examples) and hence cannot be carried out regularly during diagnosis.

The decision tree technique, mentioned above, is based on identifying different groups of faults from each other by developing a recognition function for each group by comparing it with the other groups (chapter four). The technique operates efficiently if the examples of the different groups are well separated from the examples of the other groups (i.e. if the boundaries of the groups are well defined) by producing recognition functions which are good generalizations of their respective groups. However, if the boundaries of the groups are not well defined then the generalization will result in recognition functions which could classify examples of certain groups with others. A number of fault groups in the data could be considered as examples of the latter case since a number of their example reports closely resemble examples in other fault groups. Other cases exist in which identical reports identify different faults. As mentioned above the technique could not operate with the latter cases and required their manual separation from their respective fault groups.

Due to the above drawbacks further investigations were carried out in order to look for a technique which could avoid the above. Investigations carried out on the data revealed that reports of a given fault share common sequences of test failures, however, the reports need not be identical or very similar. In addition, identical (or similar) reports which are common in more than one fault usually occur with varying frequencies in these faults. This could be used in deducing the most likely fault if a similar report is encountered.

The results above lead the way to the development of a technique which makes use of the above features in the data. The technique is based on automatically generating a fault dictionary in which each test report in the data is stored, together with its identified fault, in a knowledge-base. When a new test report is presented, the knowledge-base is searched for a sequence of tests which match the test(s) in the new test report. The knowledge-base is updated as each new fault is confirmed by the technician. This approach can make use of various heuristics in matching the test(s) in the report with the ones already in the knowledge-base. Initial results obtained with such a system were encouraging in that 65-70% of the faults in the data were diagnosed correctly. For this reason the technique was developed as a basis for a fault diagnosis system.

The results obtained with the diagnosis system above revealed that about 20-25% of the faults in the data could not be identified (incorrectly matched or no matches are found) with about 5-10% being identified as least likely suspects (such as 4th position, 5th . .etc). Analysis of the results revealed that a proportion of the latter cases could be recovered by using circuit connectivity information in the diagnosis procedure. In this thesis, a number of heuristic techniques were developed which utilize connectivity information. Such techniques proved especially useful in the following cases :

1. Where the failure of multiple components/modules are indicated in the report.
2. Where diagnosis using matching cannot discriminate among a number of suspect faults.

In order to incorporate connectivity information into the system a number of routines were developed for introducing and manipulating connectivity information.

A small proportion of the faults, which could not be recognized by matching, were recovered by carrying out a more detailed matching (inexact matching) especially at early stages in the diagnosis procedure when the information in the knowledge-base is not complete. Our investigations revealed that this type of matching could prove especially useful at early stages of introducing reports of a new board to the system, however, as information is added to the knowledge-base its effect reverses and normal matching must be resumed.

For the system to perform effective matching a number of matching rules were developed and were stored in a table like rule-base for fast access. These rules were developed manually by studying the output of the system. A number of techniques were developed in this thesis which attempt to enable the system to learn these matching rules and re-construct its rule-base automatically. The results obtained indicated that the system can construct its rule-base to a state similar to that of the manually tuned system. However, one of these techniques improves on the manual system by learning faster and in a more uniform manner.

The performance of the rule learning techniques developed were compared to that obtained with a standard concept learning algorithm, 'version spaces'. However, the algorithm could not cope with the large concept space covered by the matching rules which required manual intervention such as restricting learning to a limited number of reports and excluding noisy examples. The techniques developed were based on mimicking the procedure followed by a human expert in developing manual rules. This procedure allows the techniques to use some heuristics (which are based on experience). These heuristics equip the techniques to deal with noise and the large data space. Another advantage of the heuristic techniques is the possibility of initialisation. This could be performed by providing a previously prepared rule-base which the system can use initially. During its operation the system can adjust some of these rules if they contradict the majority of the encountered examples. This is useful to improve system performance at early stages.

Finally, the performance of the system was investigated with additional board styles. The investigations revealed that the performance of the system is dependent on the number of examples in the data and the test information available in the report with respect to each board style.

12.2 A Final Comment on System Performance

Initial results obtained with the diagnosis system (using matching alone) indicated that approximately 65-70% of the faults in the data were identified correctly. Additional improvements to the system (such as the utilization of circuit connectivity information and the use of inexact matching) only improved specific cases which are considered a minority in the data. For this reason, no significant improvement occurred on the overall performance of the system (around 70% of the faults identified as the best match).

The results obtained in this thesis agree with that mentioned in [EVA91], which is a report based on a number of AI-based diagnosis systems used in industry. The report mentions that : "It is possible to approach 90% fault coverage with a rule-based system but it could take up to three man years to reach that level. The first 75%, say, could be achieved with around 200 rules, but to achieve the extra fault coverage would probably need around 2000 to 3000 rules. And even then, there could be internal conflicts with such a high number of rules - given different diagnosis for particular faults".

Although the above report considers traditional rule-based systems which are developed by manually incorporating rules (developed by experts) into the system, it could also apply to the system developed here. Since the heuristic matching rules can successfully identify the majority of the cases in the data, additional modifications can only recover certain minority cases in the data. In fact, the analysis of the data (such as that carried out in chapter six) indicate that a number of such minority cases exist in the data which includes most of the 20-25% not identified as the best match. The analysis also indicated that such cases occur regularly, but less frequently than the majority cases. This explained the lack of sufficient examples for a number of these minority cases which prevented their analysis and hence their treatment. It can be deduced from all of the above that a number of modifications may be required (in the form of connectivity heuristics, matching heuristics, or a heuristics based on different approaches) to treat each of the minority cases in the data. For other cases, a relatively long time may be required to collect sufficient examples for the matching procedure to be able to identify these cases.

In addition to the above, the data includes a number of inconsistent cases, referred to as noise. These are thought to be due to errors by the diagnostic technicians and/or some other ambiguous reasons. If the ATE input is directed to the system (instead of passing through the technician), then technician errors (such as, incomplete test reports and wrong diagnosis) could be eliminated. Currently, the system installed at the company (its operation manual is included in appendix E) can accept only a limited number of the boards manufactured in the company directly from ATEs.

12.3 Comparison of Standard and Heuristical Techniques

In a number of occasions in this work, standard machine learning techniques were employed so that their performance could be compared with corresponding heuristical techniques developed specifically for the different aspects of the diagnosis problem here. Such a comparison was made with the decision tree technique and the fault dictionary based technique and later during the automation of the rule-base where the version space algorithm was compared with both statistical and statistical-ranges techniques. In both cases the heuristical techniques showed an improvement over the standard ones. This was mainly due to the inability of the latter techniques in coping with the relatively large *data space* (or example space) in both of the cases. The data space is the set of all possible failure reports which includes all possible failure test patterns for all the possible faults which could occur.

The comparison of this section will concentrate on the decision tree technique and the fault dictionary based technique because these were intended for the backbone of the system since they can achieve learning and diagnosis. Some of the differences between the two techniques had already been mentioned in section 12.1.

The possibility of obtaining a complete data space for our diagnosis problem is remote. This is because the examples provided in the data are based on actual failure cases on the factory floor manifested in processes such as manufacturing the components, assembly on boards, soldering . . etc. Hence failures *over certain periods* of time are restricted to those which occur due to a malfunction in one of these processes. In addition, as was noticed from the data, failures are restricted to some common faults which occur frequently, others which occur less frequently, but regularly, and those which occur once or twice and do not occur any more. It was noticed that approximately 20-25% of the components in the circuit have never failed. Furthermore, it is unlikely to encounter every possible sequence of failed tests (test-patterns) for each of the faults encountered in the data. In view of the above, obtaining a complete data space may only be achieved over a relatively long period of time over which aspects such as manufacturing procedure and/or test procedure could change. In addition, in a situation such as our diagnosis problem information are introduced to the system progressively as reports are available rather than providing a complete example space at once.

The above suggests that the problem requires a technique which can cope with an incomplete data space as efficiently as possible and can learn progressively (make best use of information provided as quickly as possible, so that it could be used as soon as possible, without delaying the system). It is impractical to construct a decision tree during the diagnosis process. This is because each time a new example is introduced, the tree must be reconstructed with both old and new examples. This is a lengthy process especially as the number of examples in the data increases. In contrast updating the knowledge-base in the case of the fault dictionary based technique is a relatively fast process since updating a single report affects the section of the knowledge-base related to the diagnosed fault only.

In addition, during diagnosis in the decision tree technique, the decision tree is searched for a recognition function which either identifies or does not identify the considered report (indicated in boolean terms; True or False). This suggests that approximate identification cannot be performed i.e. the expression "the report is about 90% recognized by the recognition function" is invalid. However, such an approximate identification is one of the advantages of the fault dictionary based technique since initially matching could select a number of suspect faults, some of which do not have exactly matching test patterns, to the report in question. Later these are sorted, by the rule-base, according to the quality of their match as reflected in their corresponding matching scores.

The large concept space covered by the matching rules is also the reason for the limited performance of the standard machine learning technique represented in the version space algorithm in comparison with the superiority of the heuristic techniques. Constructing a rule-base by providing examples covering the whole of the concept space will produce a better rule-base, however, as with the case of the decision tree technique this could only be achieved after a long period of time (the time required for encountering all the possible numeric combinations of the four scores between two faults could be very long). In summary, The main reasons of the limited performance of the standard techniques as opposed to heuristical ones are the following :

1. Inability to handle noise (inconsistent examples), hence the need for manual intervention in separating cases of noise individually.
2. Inability to use heuristics and other parameters. Tracing and manipulating the accumulative frequency of different cases allows distinguishing between common cases and cases of noise. Once identified each case could be treated appropriately. Accumulative frequency closely corresponds to the experience

factor in humans. The use of frequency and heuristics is the main reason which enables heuristical techniques to overcome the obstacle introduced by the large data/concept space in both of the cases above.

3. Being less suitable for situations requiring progressive learning as in the diagnosis case.

12.4 Originality of Research

The diagnosis procedure of the system developed here is distinct in that the core of the diagnosis is performed by matching information on the failure report with previously stored cases. No such diagnosis procedure was found during the literature survey which was carried out up to the time of publishing [ARS90]. This could be due to the difficulty of obtaining test data from companies for commercial reasons caused by competition.

A number of heuristical techniques was developed here. These use connectivity information for recovering cases which could not be isolated using matching alone. Such techniques are CAT and treating multiple component/module failures as described in chapter seven.

In addition a number of rule learning techniques are developed for learning the matching rules. These techniques are especially developed to operate on scores produced by the matching procedure. The techniques are developed to mimic the procedure used by test engineers in analysing matching scores and developing rules appropriately in order to recover individual cases. The techniques are based on monitoring frequency of individual cases (the different conditions imposed by the scores with their corresponding consequence for each encountered example).

Most of the techniques developed as part of this work apply to any board style as long as the board is tested by an ATE which produces information which could be matched adequately by the system. The circuit information regarding each board can be introduced to the system via the circuit compiler developed here.

The work aimed to develop a general system which could handle different board styles and is as independent of external intervention as possible. Although, this aim is achieved, a number of other steps might be required in the future for preparing such a system for commercial use. Some of these are described in the next section.

Finally, as mentioned in section 12.2, approximately 70% of the faults in the data are identified successfully. This performance matches that of other systems which mainly perform their diagnosis by using circuit specific information [EVA91]. In such systems, circuit information is obtained by simulation or asking the experts (see chapter two). Both of the previous tasks are lengthy and complex (for example, simulating a circuit as complex as the Slac (chapter three) could last for days). However, the research described in this thesis shows that diagnosis could be achieved by using simple heuristic matching (with other heuristic techniques developed here) and yet the performance could match that of systems which use more technical information.

12.5 System Implementation

The research in this thesis was implemented using the Lisp programming language. In excess of 7000 lines of lisp code were developed. These routines include the following:

1. The construction of decision trees and performing diagnosis using these (Approx. 800 lines).
2. The construction of a fault dictionary based diagnosis system which can utilize different matching techniques; i.e. exact and inexact, with different test parameters (Approx. 1500).
3. The development of different rule-learning techniques (Statistical, Statistical-Ranges, Version Spaces) which include mechanisms for rule execution and update procedures (Approx. 1000).
4. Circuit compilation routines for introducing circuit connectivity information to the system (Approx. 800).
5. The development of a number of techniques which exploit connectivity information for improving diagnostic performance (Approx. 1400)
6. Data analysis tasks for identifying characteristics of the failure reports in the data (or only that of individual groups). Examples of such tasks include finding out the different conditions under which a given test (or a group of tests) can fail, identifying the range of voltage (or current) measured values for a given test (Approx. 1500).
7. Utility programs for analysing the diagnostic performance of the system in response to the different improvement attempts investigated. These include statistical information regarding the diagnostic performance at different stages

of building a knowledge-base. Routines were developed for constructing bar charts and putting performance results in a form which could be used by the different packages, available in the department, for more thorough examination of the results (Approx. 800)

The routines above do not include the modifications performed in order to enable the system to be used in the company. These include tasks such as building a menu interface for the users, interfacing the system with the ATEs in the company, and adapting a new form for information storage which is consistent with the standard format used in the company (these in turn total up to 4000 lines, 500 in C and the rest in Lisp). These are not included in this thesis since they do not contribute to the research work in it. Appendix E describes the above in greater detail.

The use of a lower level programming language (such as C) would have increased the amount of time required for developing the routines mentioned above. This would have decreased the amount of research by an increase in the tasks involved in programming. For example, by considering aspects such as variable declaration and memory allocation.

Frame-based tools, such as expert system shells, were an alternative to be used in this work, however, the main reason for not using such tools is that they were not available.

Lisp is available in our department and offers a reasonable alternative to frame-based tools. It also offers the flexibility of a programming language in developing general routines such as connectivity analysis and circuit compilation, statistical analysis,...etc. This is in addition to being able to develop traditional AI based systems. Using Lisp meant choosing a more computationally expensive method (since tasks such as memory allocation and garbage collection are performed automatically). In our research, however, up to 15 Megabytes of RAM could be exploited and hence the use of lisp could be afforded (this did not effect the diagnosis time of the developed techniques, which usually takes seconds). In addition, The flexibility of dynamic Lisp data structures and the interpreter means that Lisp is a good language for prototyping.

12.6 Suggestions for Future Work

This section proposes a number of tasks which could be carried out in order to increase the efficiency of the system developed here. The tasks aim to create a more independent system able of performing a better diagnosis.

12.6.1 Automatic Disposal of Old Data

Currently, the system accumulates all the data in its knowledge-base. However, it is possible that some of the information in the knowledge-base might get out of date especially in the case of some temporary faults appearing in one (or more) of the board batches due to a malfunction in processes such as manufacturing, assembly, soldering, . . . etc. Such faults vanish after adjusting the responsible process and might not occur again.

It is recommended that a mechanism be developed for treating data associated with such faults in the system. It is suggested that this is accomplished by adding a date-related parameter in the knowledge-base, such as the date at which a given test-pattern was last encountered in a given fault. This requires the modification of the test-pattern structure within a fault-entry (see section 5.2.1) such that an additional field, representing the date (date and time), is added. Each time a known test-pattern is encountered, its frequency count will be incremented and the date will be updated. For a new test pattern, however, a whole test-pattern structure must be created (with a frequency count of 1 and the date being the current one).

Later, test-patterns in the knowledge-base could be removed if they have not been encountered over a certain *period* of time. The duration of such a period must be investigated which may suggest monitoring the actual manufacturing and diagnosis procedure in the factory floor to obtain a realistic time period.

A more efficient procedure could be transferring the data (instead of removing them) to another location (file) from which they could be recalled if the fault happens to appear again (or for analysis purposes).

Another aspect of the test-pattern structure which need to be investigated is the frequency count. Currently this field is incremented with each test-pattern encountered. It may be necessary to normalize (i.e. reduce to a certain level) the frequency counts regularly. This will decrease the effect of patterns which occurred frequently in the past and will prevent them from effecting new patterns. Again this requires further investigation in order to find the best conditions under which normalization should be executed and how (i.e. whether to affect all the test-patterns in the knowledge-base simultaneously or only those which grow larger relative to other test-patterns).

Without normalization some test-patterns may be very large such that even if they occur no more their effect may remain dominant. However, it must be noted that normalization may not be required if the period at which old test-patterns are removed from the knowledge-base is relatively short since the above dominant patterns will be removed.

It is thought that the period at which old test-patterns should be removed from the knowledge-base would be relatively long (weeks or months depending on the procedure at the company). For this reason a limited amount of normalization may be required.

12.6.2 Automatic Selection of a Match Parameter

As seen in the previous chapters matching only uses a limited number of the test parameters provided on the report. Investigations carried out in chapter seven revealed that the use of test names only provides the best results.

The investigations above were carried out by instructing the system to use a specific test parameter (see section 3.3) in the matching procedure and examining the results over a number of trials. It is more effective if the system can carry out such investigations independently, as more data are available, and deduce the parameter which provides the best results. This is especially significant for a system which handles a number of board styles tested by different ATEs. The information for different boards may be different and hence the best parameter(s), for matching, must be chosen.

It is recommended that a mechanism be developed which carries out matching separately with each parameter (or sequence of parameters) and compares results over a number of trials for the different cases. Such investigations could be carried out at a stage when the system is off-line (durations when not in use e.g. at night). The procedure could be performed each time certain amount of reports are added to the ones already accumulated for a given board.

The system is already developed such that all the information on the report is stored (in separate fields). In addition, matching implemented such that any of the fields could be used. Only the mechanism of automation is required for the above.

12.6.3 Investigating the Use of Neural Networks

Neural Networks [RIC91] proved successful in areas such as image processing and speech recognition. In both these areas most of the problems are cases in which a large number of examples exist and neural networks try to adapt internally (mainly by adjusting the weights on the network connections) to the specific features in the examples. With each new example the network will adjust itself such that the new case is included.

The process of learning from examples is performed in both the decision tree technique and the fault dictionary based technique. Neural networks could be trained with the test sequences of the different faults in the data and the diagnosis performance using such a network could be compared to the techniques above. It must be noted that for this to be achieved, the data may require some transformation/coding to a form more adequate for manipulation by a neural network.

In addition, neural networks could be employed in order to learn the matching rules (section 5.2.3). Matching scores could be fed into the inputs of the network and the network could adapt itself by learning from the scores of all the reports in the data. The performance could be compared with standard and heuristic techniques.

12.6.4 Continuation of Analysis with New Data

Most of work carried out in this thesis was based on the analysis of the data. These analysis revealed groups of examples which share common features. By both manual analysis and developing programs to analyse some of these cases different heuristics were developed to detect and deal with similar examples when encountered by the system in the future.

However, a number of other cases exist which currently have only few examples in the data hence analysis could not be carried out effectively. It is necessary to carry out such analysis as more data are provided in order to treat them if possible.

An example of a case where more examples are necessary is the investigations to incorporate measured parameters in the matching since currently the number of values for cases which cannot be recovered with current way of matching is not enough. Since these are the cases where additional means are necessary for improvement, more examples must be collected.

More examples are also necessary to carry out further experiments with the diagnosis of multiple board styles. Furthermore, it is useful to test the performance of the system with data and boards obtained from other sources.

APPENDIX A
DATA FOR BIASING THE DECISION TREES

This appendix reveals the data used in deciding the initial specifications of the decision trees discussed in chapter four. Once the initial specifications were chosen (in terms of minimum or maximum sparseness and the structure of the tree being symmetrical or non-symmetrical), they were used in constructing the decision trees with the real data.

These data are relatively small in size and contain only eight faults which are represented by *fault1*, *fault2*, *fault3*,..., *fault8*. Associated with each fault are a list of dummy test reports, which contain a number of failed tests. A single failed report is presented in the form :

(f*no* test1 tests2......etc.)

no represents the fault number

The rest of the report contains a number of failed tests which are represented by one of the following symbols:

t0 t1, t2 t3 t4 t5 t6 t7 t8 t9 s0 s1 s2 s3 s4 s5 s6 s7 s8

A report which caused the failure of *fault1* and consist of the failure of 3 tests, which are *t4*, *t5* ,and *t7* is presented in the following form:

(f1 t4 t5 t7)

Note

The tests which did not fail are not shown in the report.

The following represents the faults with their corresponding reports:

fault1 :

(f1 t3 s1)
(f1 t2 t3 s1)
(f1 t3 s1)
(f1 s1)

fault2 :

(f2 t7 s2)
(f2 t7 s2)
(f2 t7 s2)

fault3 :

```
        (f3 t0 s3)
        (f3 t0 s3)
        (f3 t0 t9 s3)
```

fault4 :

```
        (f4 t6 s4)
        (f4 t6 s4)
```

fault5 :

```
        (f5 t2 s5)
        (f5 t2 s5)
        (f5 t2 s5)
```

fault6 :

```
        (f6 t1 s6)
        (f6 t1 s6)
        (f6 t1 t8 s6)
        (f6 t1 t8 s6)
        (f6 t1 s6)
```

fault7 :

```
        (f7 t9 s7)
        (f7 t9 s7)
        (f7 t9 s7)
```

fault8 :

```
        (f8 t6 s8)
        (f8 t6 s8)
```

The above data were used to construct both a symmetrical and non-symmetrical trees. Each tree was in turn constructed with a minimum and a maximum sparseness. Testing the trees was performed by subjecting them to eight examples of each of the above faults. Examples of each fault were implemented by using all or some of the characteristic tests of the fault. In some cases some additional tests were also added to the examples. This implementation is followed in order to make the examples above resemble those in the actual data.

The rationale in using the dummy data is in their simplicity in comparison with the real data. Although . Using the dummy data, trees could be produced faster and easier. The simplicity of the data will assist in producing a more efficient tree (since it is easier to understand the procedure of identifying each of the trial examples and spot any wrong identifications). Additional detail, regarding the above, could be found in chapter four.

APPENDIX B
THE MANUAL RULE-TABLE

S	*P*	*G*	*F*	*SR*
>	>	>	>	OK
>	>	>	=	OK
>	>	>	<	OK
>	>	=	>	OK
>	>	=	=	OK
>	>	=	<	OK
>	>	<	>	OK
>	>	<	=	OK
>	>	<	<	OKif(S_5F_{25})
>	=	>	>	OK
>	=	>	=	OK
>	=	>	<	OK
>	=	=	>	OK
>	=	=	=	OK
>	=	=	<	OK
>	=	<	>	OK
>	=	<	=	OKif(S_2)
>	=	<	<	OKif(S_2F_2)
>	<	>	>	+
>	<	>	=	+
>	<	>	<	+
>	<	=	>	+
>	<	=	=	+
>	<	=	<	+
>	<	<	>	+

>	<	<	=	+
>	<	<	<	+
=	>	>	>	OK
=	>	>	=	OK
=	>	>	<	OKif(F_2)
=	>	=	>	OK
=	>	=	=	OK
=	>	=	<	OKif(F_2)
=	>	<	>	OKif(S_2)
=	>	<	=	OKif(S_2)
=	>	<	<	OKif(S_2F_2)
=	=	>	>	OK
=	=	>	=	OK
=	=	>	<	OKif(F_2)
=	=	=	>	OK
=	=	=	=	OK
=	=	=	<	~OK
=	=	<	>	+
=	=	<	=	~OK
=	=	<	<	~OK
=	<	>	>	OKif(P_2)
=	<	>	=	OKif(P_2)
=	<	>	<	+
=	<	=	>	+
=	<	=	=	~OK
=	<	=	<	~OK
=	<	<	>	~OKif(F_7)
=	<	<	=	~OK
=	<	<	<	~OK

<	>	>	>	+
<	>	>	=	+
<	>	>	<	+
<	>	=	>	+
<	>	=	=	+
<	>	=	<	+
<	>	<	>	+
<	>	<	=	+
<	>	<	<	+
<	=	>	>	+
<	=	>	=	~OKif(S_2)
<	=	>	<	+
<	=	=	>	+
<	=	=	=	~OK
<	=	=	<	~OK
<	=	<	>	~OK
<	=	<	=	~OK
<	=	<	<	~OK
<	<	>	>	~OKif(S_5F_{25})
<	<	>	=	~OK
<	<	>	<	~OK
<	<	=	>	~OK
<	<	=	=	~OK
<	<	=	<	~OK
<	<	<	>	~OK
<	<	<	=	~OK
<	<	<	<	~OK

S, P, G, F

refere to the relationship of the corresponding matching score (section 6.2.2) in the two test-patterns presented to the rule-base. The relationships of each set of scores in a given table entry define $\text{condition}_{level1}$ for the main rule (section 6.2.3.2) defined by the table entry.

SR

represents the secondary rule of the main rule.

OK

The two test-patterns are in the correct order.

~OK

The two test-patterns are in the wrong order, swap them.

+

Add the scores of each test-pattern and compare the total. The correct order is if the test-pattern with the largest total is first.

OKif(X_i)

The two test-patterns are in the correct order as long as the following is satisfied:

$X_{pattern1}-X_{pattern2} <= i$

X is one of the matching scores
i an integer value deduced by inspection of the data

~OKif(X_i)

The two test-patterns are in the wrong order as long as the following is satisfied:

$X_{pattern1}-X_{pattern2} > i$

This is the opposite case of the previous one.

OKif(X_iY_j)

The two test-patterns are in the correct order as long as the following is satisfied:

$X_{pattern1}-X_{pattern2} <= i$ and
$Y_{pattern1}-Y_{pattern2} <= j$

X and Y are one of the matching scores
i and j are integer values deduced by inspection of the data

~OKif(X_iY_j)

The two test-patterns are in the wrong order as long as the following is satisfied:

$X_{pattern1}-X_{pattern2} > i$ and
$Y_{pattern1}-Y_{pattern2} > j$

This is the opposite case of the previous one.

APPENDIX C
INVESTIGATING THE USE OF CONNECTIVITY ANALYSIS TABLES WITH EXTRACTED EXAMPLES FROM THE DATA

This appendix illustrates the use of the connectivity analysis tables (see chapter eight) with examples extracted from the data. The examples could not be improved with the matching rules since they have identical scores (and in some cases the scores are close).

The case of each example is represented in a separate case (see below). Some of the cases contain detailed explanation others have small comments since the procedure followed is the same. a number of symbols are used to assess the result of applying connectivity analysis and are listed below:

R Full Recovery

N No Recovery

I Improved, however, true fault still not in No.1 position.

C Confirmed, True fault is already in the No.1 position. This is purely accidetal, since it depends on the order the faults are met by the matching algorithm.

As mensioned in the previous section, Connectivity analysis in CAT looks at the following between each two components (say; x & y):

i. Min connectivity relationship between x and y are used FIRST: e.g, the type of connection exsts between the two components:
direct (d),
passes thr. one component (1),
passes thr. two components (2),
.... N .... (N).
(Initially we look at the most direct (shortest) path between x and y).

ii. Infomation such as how many direct paths, How many indirect paths (and the number of components passed through) are considered if (i) did not provide enough to discriminate.

N.B:

The entries in the table were made by observing the connections on the circuit diagram.

CASE::1

Suspect List :

(534 (44 (5 9 9 1)) (494 (5 9 9 1)) (514 (5 9 9 1)) (54 (5 9 9 1)) (534 (5 9 9 1)) (542 (5 9 9 1)) (541 (4 7 9 1)) (214 (4 7 9 1)) (31 (5 6 5 1)) (114 (5 6 5 1)) (74 (4 3 2 4)) (394 (3 5 9 1)) (334 (3 5 9 1)) (504 (2 3 0 1)) (35 (5 1 0 1)))

CAT :

44	1	1	d	d	3
	494	2	2	2	2
		514	2	2	x
			54	1	x
				534	x
					542

{I}

order deduced from above is 44 54 534, with the last two being in any order.

CASE::2

Suspect List :

(500 (516 (1 9 9 3)) (506 (1 9 9 3)) (266 (1 9 9 2)) (496 (1 9 9 1)) (76 (1 9 9 1)) (456 (1 9 9 1)) (56 (1 4 4 1)) (256 (1 4 4 1)) (566 (1 4 4 1)) (46 (1 4 3 1)) (540 (1 2 0 1)))

{R}

The dispute here is about the positions of 516 and 506 since they both have the same scores. Although the components proceeding them are different in the overall scores,however, they share the same first 3 scores which are major scores. when these faults 266 496 76 456 are used to to determine which of 516 and 506 is the best, connectivity analysis shows by a narrow margin that 506 is a better choice. This is expected since the components are closely connected to each other and are both relays which perform many tasks commonly

CASE::3

Suspect List :

(536 (536 (1 9 9 1)) (406 (1 9 9 1)) (446 (1 9 9 1)) (46 (1 4 2 1)))

536	1	2
	406	2
		446

=> *{N}*

536	1	2	d
	406	2	d
		446	1
			46

=> *{N}*

536	13	22	d1
	406	24	d2
		446	11
			46

=> <u>536</u> $\boxed{1322d1}$ *{R}*

=> 406 1324d2

The case above is interesting, since it is solved in THREE iterations of using connectivity.

1. *Using only the components with the same score connectivity did not distinguish between 536 and 406*
2. *Including 46 gives equal connectivity scores to both since 46 is connected directly to both*
3. *However when considering all the connections between each 2 component instead of taking the most direct connection. 536 has closer connections with all the rest than 406. (smaller numbers=> more interconnections).*

CASE::4

Suspect List :
(40 (530 (3 9 9 2)) (40 (3 9 9 1)) (510 (3 9 9 1)) (541 (3 9 9 1)) (210 (3 9 9 1)) (542 (3 7 7 1)) (50 (3 7 6 1)) (490 (3 7 5 1)) (70 (3 2 1 4)) (390 (2 6 9 1)) (31 (2 4 5 1)) (330 (2 4 5 1)) (110 (3 3 4 1)) (500 (1 2 0 1)) (35 (3 1 0 1)))

Although the first score has a frequency-Count differing by 1 from the others the scores could be considered as similar. Connectivity reveals the true fault as No1.

CASE::5

Suspect List :
(504 (504 (2 9 9 2)) (264 (2 9 9 2)) (74 (2 9 9 1)) (254 (2 9 9 1)) (514 (1 4 9 3)) (54 (2 4 4 1)) (454 (1 4 9 1)) (494 (1 3 4 9)) (541 (1 3 4 1)) (44 (1 3 3 1)) (540 (1 2 0 1)))

{C}

Connectivity provides confirmation rather than chance as above see paper

CASE::6

Suspect List :
(397 (47 (2 9 9 1)) (397 (2 9 9 1)) (337 (2 9 9 1)) (497 (2 6 8 1)) (537 (2 4 5 2)) (57 (2 4 5 1)) (542 (2 4 4 1)) (117 (2 3 3 1)) (517 (24 1 1)) (541 (1 4 9 1)) (77 (1 4 8 1)) (31 (2 3 3 1)) (217 (1 3 6 1)) (447 (2 2 2 1)) (35 (2 1 0 1)))

CAT :

47	*d*	*dd*	*1*
	397	*1*	*2*
		337	*1*
			497

{N}

CASE::7

Suspect List :
(44 (54 (3 9 9 4)) (44 (3 9 9 1)) (314 (3 9 9 1)) (404 (3 5 4 1)) (454 (3 4 3 1)) (35 (3 1 0 1)))

54	*d*	*1*
	<u>44</u>	*d*
		314

{R}

CASE::8

Suspect List :
(532 (512 (3 9 9 1)) (541 (3 9 9 1)) (532 (3 9 9 1)) (212 (3 9 9 1)) (42 (3 7 7 1)) (542 (3 7 7 1)) (52 (3 7 6 1)) (492 (3 7 5 1)) (72 (3 2 2 4)) (392 (2 6 9 1)) (31 (2 4 5 1)) (502 (1 2 0 1)))

CAT :

512	*d*	*2*	*2*
	541	*3*	*1*
		532	*1*
			212

{N}

According to above a possible order for the faults would be :

(512 d22) (541 d31) (532 123) (212 112)

CASE::9

Suspect List :
(491 (491 (1 9 9 1)) (501 (1 9 9 1)) (71 (1 9 9 1)) (41 (1 4 6 1)) (261 (1 4 4 3)) (51 (1 2 2 1)) (540 (1 2 0 1)))

491	*d*	*1*
	501	*2d*
		71

{N}

CASE::10

Suspect List :
(517 (517 (1 9 9 2)) (507 (1 9 9 2)) (267 (1 9 9 2)) (497 (1 9 9 1)) (457 (1 9 9 1)) (257 (1 4 5 1)) (77 (1 4 2 1)) (57 (1 2 2 1)) (540 (1 2 0 1)))

CAT :

517	*d*	*d*	*2*	*x*
	507	*11*	*11*	*x*
		267	*33*	*x*
			497	*x*
				457

{C}

the fault with two direct connections is 517 probably followed by 507 with (d 11 11) and others to follow

where n is a number: Means n components separate the two components with the connection passing through the n components

two numbers indicate two connection paths exist between the two components. each having some number of components separating them.

our decision gives priority to the number of direct connections the other are based logic and subject to experimentation.

CASE::11

Suspect List :
(251 (51 (2 9 9 1)) (251 (2 9 9 1)) (511 (2 6 5 2)) (41 (2 6 5 1)) (36 (2 6 5 1)) (501 (2 6 4 15)) (431 (2 3 2 3)) (31 (2 4 2 1)) (71 (2 1 1 4)) (531 (2 1 0 1)))

CAT :

51	*2*	*2*	*d*	*1*	*1*
	251	*1*	*d*	*1*	*d*
		511	*1*	*1*	*d*
			41	*1*	*d*
				36	*d*
					501

{R}

Although connectivity analysis shows that most of the components are connected to 501, it sould be remembered from previous cases that we use the faults with relatively (convinsingly) close scores only to show which of the two faults with best identical matching scores is the most deservant of the No1 position. since sorting according to matching already decided the position of the others and we use connectivity to change position of the identical faults with making least disturbance possible to the matching decision, since it is reliable (sine 60% diagnosis is achived using matching only).

CASE::12

Suspect List :
(51 (151 (1 9 9 1)) (51 (1 9 9 1)) (311 (1 9 9 1)) (451 (1 3 1 1)) (41 (1 2 1 1)) (71 (1 1 0 1)))

151	d	1
	51	11111
		311

{R}

both 151 and 51 have two direct connections to them and a path which passes thr. one component,however,the number of such passes are much more in 51 making it the no1 coice. 311 has no single direct hence it is not a candidate for No1 position.

CASE::13

Suspect List :
(56 (496 (7 9 9 1)) (56 (7 9 9 1)) (536 (7 9 9 1)) (46 (7 8 7 1)) (31 (7 6 6 1)) (541 (5 6 9 1)) (216 (5 6 9 1)) (76 (7 4 2 1)) (396 (3 4 9 1)) (516 (2 3 9 22)))

CAT :

496	2	2	1	2
	56	1	d	d
		536	d	1
			46	d
				31

{R}

- * *The dispute is about the position of : 496 56 536 since The rules cannot distinguish among exact matches (Rules determine best Match/fault best on the matching scores only, if these are equal then there is no way of discriminating among faults).*

- * *Constructing the CAT for only the 3 gives no sufficient information to discrimanate. Hence, The Faults with the Next highest scores are used To discriminate among 496 56 536 (this is important, these faults will not be candidates for the No1 position since the Rules already distinguished them from the others), however, they will only be used to get a confident confirmation about the position of the 3 contenders for No1.*

The ASSISTANT FAULTS are :46 and 31 since these :
**have the next highest scores*
**their scores are close to each other as shown below :*
sizInt(most significant score) are identical
PropInt difference is 2

Sig difference is 1
freqC are identical
and hence are chosen both.

CASE::14

Suspect List :
(537 (497 (6 9 9 1)) (537 (6 9 9 1)) (47 (6 8 7 1)) (31 (6 6 6 1)) (541 (4 6 9 1)) (217 (4 6 9 1)) (517 (2 3 9 14)) (397 (2 3 9 1)) (77 (4 2 1 1)))

CAT :

497	2	*1*	2
	537	*d*	*1*
		47	*d*
			31

{R}

537 has a direct connectio to it from the other good quality matches failed components, whereas,497 does not. Since, dispute is about 497 and 537 then clearly 537 deserves The No1 position.

CASE::15

Suspect List :
(530 (541 (3 9 9 1)) (530 (3 9 9 1)) (490 (3 7 6 1)) (40 (2 6 7 6)) (390 (2 6 9 1)) (510 (1 3 9 12)) (31 (3 3 2 1)) (70 (3 2 1 1)))

CAT :

541	3	1
	530	2
		490

{R}

Even by taking only 490, (since it is the only fault with next scores, the rest which follow vary in their scores and are not close to 490s score, but 40 and 390 are the next best), it is clear that 530 has the best connections with 490 and 541 than the connections of 541 with 530 and 490.

CASE::16

Suspect List :
(53 (43 (3 9 9 1)) (53 (3 9 9 1)) (453 (3 2 1 1)))

CAT :

43	*d*	*1*
	53	*d*
		453

{R}

53 has two direct connections, while 43 is only directly connected to 53.

CASE::17

Suspect List :
(42 (42 (1 1 2 1)) (31 (1 1 2 1)) (532 (1 1 1 1)))

{C}

CAT confirms choice which otherwise depends on order

CASE::18

Suspect List :
(517 (517 (1 9 9 1)) (507 (1 9 9 1)) (267 (1 9 9 1)) (257 (1 4 4 1)) (77 (1 4 3 1)))

{C}

CAT confirms choice which otherwise depends on order

CASE::19

Suspect List :
(266 (506 (2 9 9 1)) (76 (2 9 9 1)) (266 (2 9 9 1)) (256 (2 9 9 1)) (516 (1 4 9 2)) (456 (1 4 9 1)) (496 (1 3 4 3)) (541 (1 3 4 1)))

CAT :

506	*d*	*1*	*d*
	76	*2*	*1*
		266	*2*
			256

{R}

An example where this technique fails to improve results since 506 has two direct connections with the others failed.

CASE::20

Suspect List :
(73 (493 (1 9 9 2)) (73 (1 9 9 2)) (513 (1 9 9 1)) (503 (1 9 9 1))(263 (1 4 4 3))(43 (1 4 6 1)) (53 (1 2 2 1)) (540 (1 2 0 1)))

CAT :

493	*d*	*2*	*1*
	73	*1*	*d*
		513	*d*
			503

{R}

**The positions of 493 and 73 are disputed*

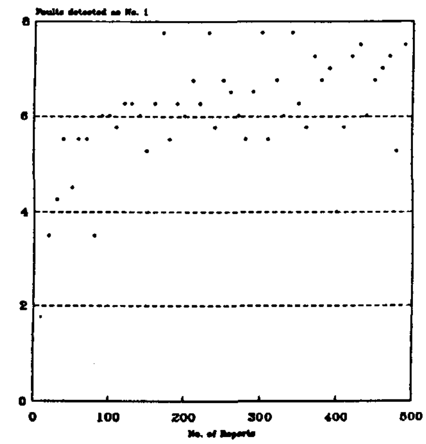
**The aim is to use 513 and 503 ,since they are close, to deduce which of 493, 73 deserves the No1 position*

**Using connectivity it can be argued that 73 is a better choice (see CAT)*

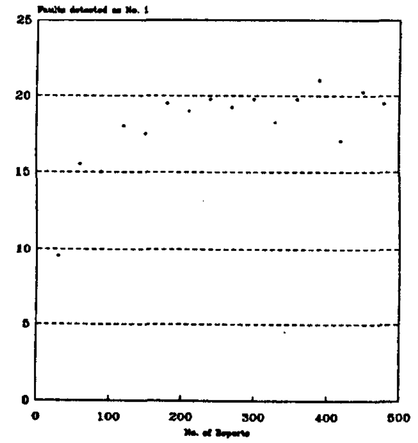
APPENDIX D
LEARNING CURVES

This appendix includes all learning curves obtained for the manual system and those using the two learning techniques developed in chapter nine. For each system, curves are plotted at steps (see chapter nine) of 10, 30, 50.

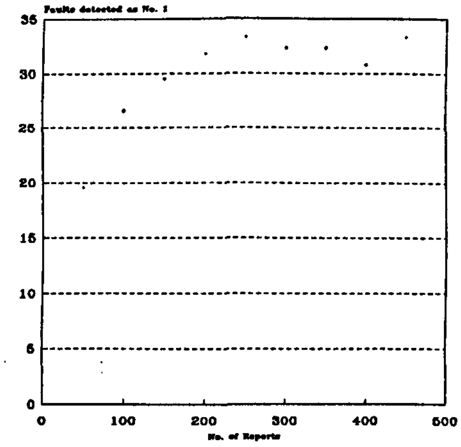
D.1 Using Manual Rules



(a) Steps of 10 Reports



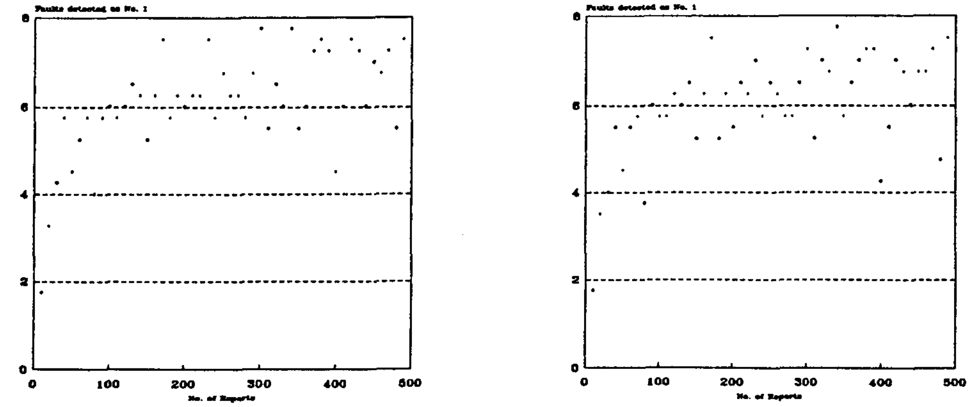
(b) Steps of 30 Reports



(c) Steps of 50 Reports

D.2 Using Rule-Learning Techniques

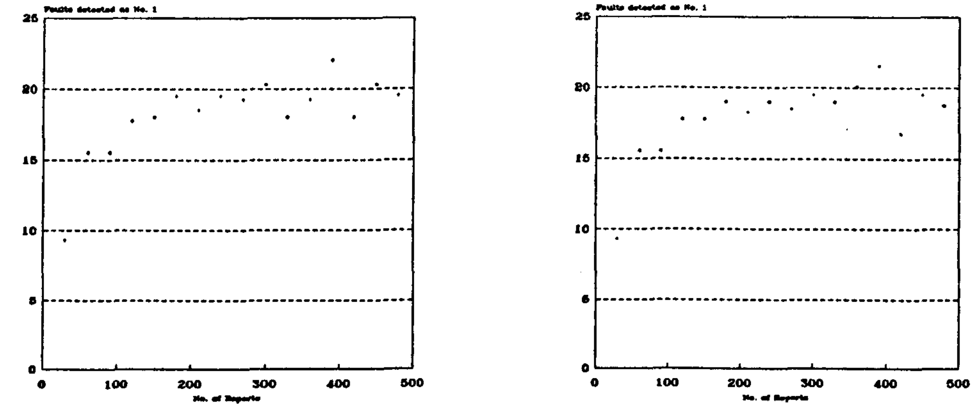
D.2.1 Initialised Rule-Table



(a) Statistical

(b) Statistical-Ranges

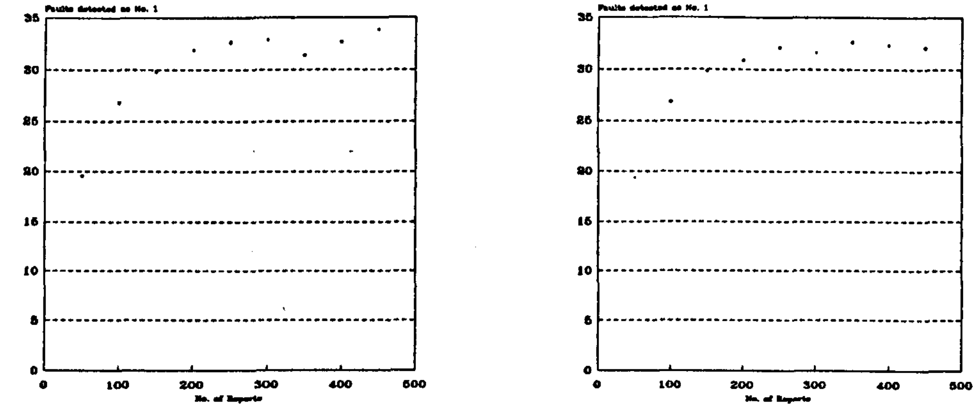
Steps of 10 reports



(a) Statistical

(b) Statistical-Ranges

Steps of 30 reports

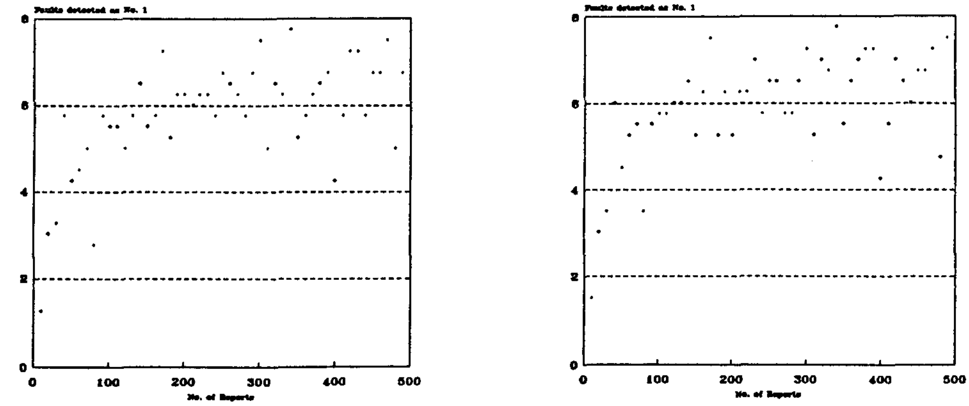


(a) Statistical

(b) Statistical-Ranges

Steps of 50 reports

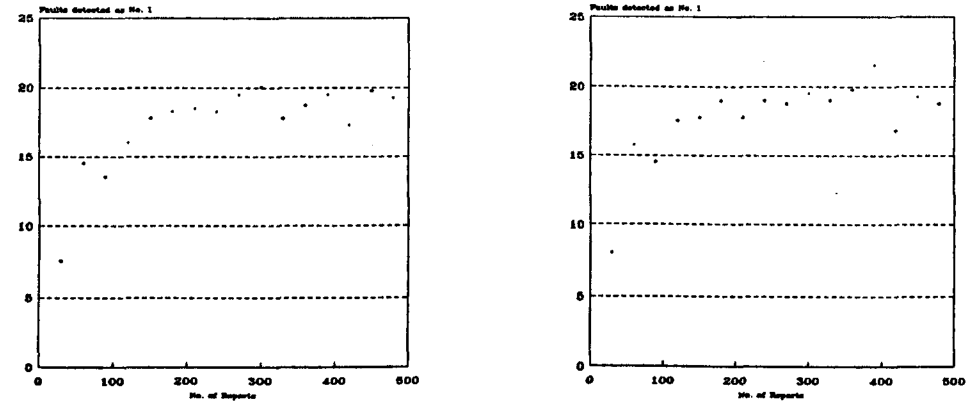
D.2.2 Non-Initialised Rule-Table



(a) Statistical

(b) Statistical-Ranges

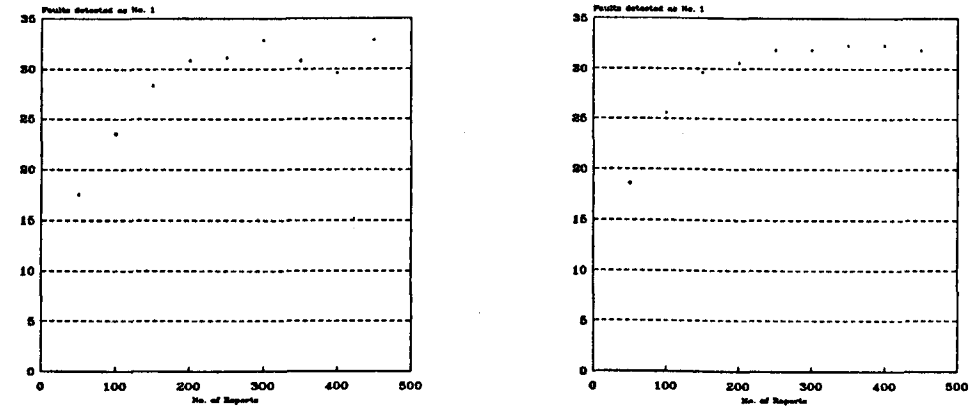
Steps of 10



(a) Statistical

(b) Statistical-Ranges

Steps of 30



(a) Statistical

(b) Statistical-Ranges

Steps of 50

APPENDIX E
INTELLIGENT FAULT DIAGNOSIS SYSTEM
(IFADS)
User Manual

E.1 Introduction

The system described in this manual was developed for Ericsson Telecommunications Ltd. at The University of Hull as part of a research contract for using AI techniques in board failure diagnosis.

The manual describes the basic use of the *Intelligent Fault Diagnosis System* (**IFADS**). It refers to the different menu options and the restrictions involved regarding the user input.

The manual starts with a *walk through menu options* describing only the restrictions on the user input at each stage since the menu options are self explanatory.

Next, the system environment is described with emphasis on the different files which are used by the system and their location within the system environment. The user hierarchy is described in section 3.3. Section 4 briefly describes system maintenance and trouble-shooting.

Finally, a brief theoretical background is provided about the fault diagnosis technique used by the system with indication on the significance of the system output.

E.2 A walk through menu options

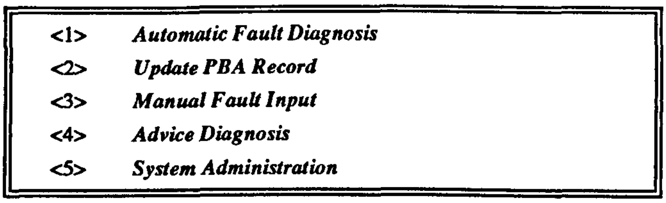
The system has two main menus; The OPERATOR MENU and the ADMINISTRATION MENU. The first section will be devoted to the former menu and the second section will describe the latter.

E.2.1 Menu selection error recovery

A key is available at most menu options which will exit the user from the current menu to the previous one in the hierarchy. In most cases the key must be confirmed by the user. The user can exit from any position in a menu by typing *q* or *Q*.

E.2.2 Operator Menu

The operator menu displays the following options :



Note

The user is warned to observe the restrictions associated with each menu option (mainly regarding input field locations and the input form) mentioned at the appropriate sections. Failing to comply, will lead to feeding the system with erroneous data or system failure during diagnosis or performing one of the administration tasks.

A global requirement in all menu options is:

Fault Position No.	***3 digits/characters***
Board Serial No.	***10 digits***

The following sub sections will consider each individual option in the operator Menu.

E.2.2.1 Automatic Fault Diagnosis

This option is used to perform a diagnosis for a given board and failure report. In general, to perform a diagnosis the following conditions must be satisfied :

i. ***A PBA file must exist with a valid failure report in it.***

ii. ***In the case where more than one failure report exist, diagnosis is always performed on the last non-diagnosed report. For this to be performed successfully, the previous diagnosed report must be updated (if the previous report is not diagnosed then diagnosis will proceed).***

On choosing this option, the operator is prompted for the ***Board Serial Number*** which is used to retrieve the contents of a given PBA file and introduce a single failure report (if more than one report exists) to the Diagnosis system.

When the serial number is input and confirmed by the user the system starts its fault identification process by searching the knowledge-base (see section 5) during which the screen displays the following (during which which no key should be pressed) :

```
*********************************
Identifying Fault .......WAIT
*********************************
```

The result of the diagnosis displays, at most, the best 3 suspects. Each suspect will be associated with a percentage score indicating the quality of its match. In most cases, *choosing the fault according to order only should point to the most likely fault* (for more information about evaluation of system result, refere to section 5.4). The following is an example of the system diagnosis output.

```
*******************************
RESULT OF FAULT DIAGNOSIS
*******************************
```

Fault	%
41	60
531	30
32A	10

Example of diagnosis output

In addition to the above, the diagnosis procedure will place up to ***seven*** of the best suspect components in the appropriate fields of the *diagnosis flags* at the end of the report in the PBA file (see section 3.1.3.1).

In cases where no diagnosis is found, the first diagnosis flag position is changed to the sequence of characters ***DIG*** for the possibility of updating (section 2.2) when the fault is located manually, otherwise updating is not possible according to the diagnosis procedure followed in the factory.

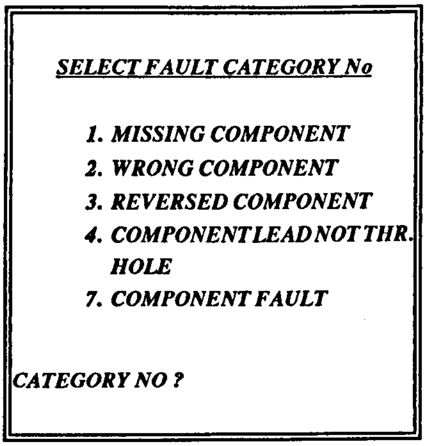
E.2.2.2 Update PBA Record

Once the fault is known, this option can be used to enter the component/grid in the appropriate field in the PBA failure record. This, however, does not involve updating the component in the knowledge-base of the system.

If more than one report exists in a PBA file, then the last diagnosed (and not updated) report is selected for update.

The faulty component must not be greater than *3 digits/characters*, this is due to the internal structure of a PBA file. Entering a number which is greater than 3 digits will lead to system failure (to recover from system failure, refer to appropriate section).

A fault could either be a component position number (***numeric***) or a grid identifier (***alpha-numeric***). Depending on the type of input for the fault (see above) the system displays one of the following menus :

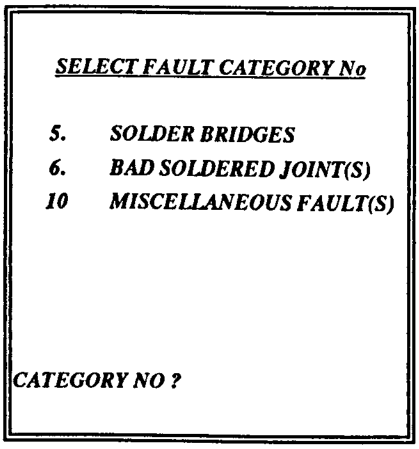


SELECT FAULT CATEGORY No

1. *MISSING COMPONENT*
2. *WRONG COMPONENT*
3. *REVERSED COMPONENT*
4. *COMPONENT LEAD NOT THR. HOLE*
7. *COMPONENT FAULT*

CATEGORY NO ?

(a) Numeric fault categories



SELECT FAULT CATEGORY No

5. *SOLDER BRIDGES*
6. *BAD SOLDERED JOINT(S)*
10 *MISCELLANEOUS FAULT(S)*

CATEGORY NO ?

(b) Alphanumeric fault categories

Each menu displays the possible categories any corresponding fault could belong to. When the *CATEGORY No* is entered, the user will be prompted for the same number again as confirmation.

When the update is complete, the fault is updated in the *7th* field of diagnosis flags of the report record in the PBA file (see section 3.1.3.1) while the category no will be set in the *10th* field representerd in the form :

F+ | ***CATEGORY No*** | ***(e.g, F10)***

The user will be prompted if another board is required to be updated.

E.2.2.3 Manual Fault Input

This option is used to enter the details of a failure report manually, instead of sending the failure report via the LPA and through the serial port. A PBA file will be created if the board has not been seen before otherwise the appropriate file will be updated. This option will not perform a diagnosis on the entered report, to perform a diagnosis, the option in section 2.1 must be used.

The user will be prompted for the information in the following table, and should conform to the specifications mensioned regarding the input size :

Required Information	*Expected no of Digits/chars*
LPA Test Station No	*2*
Program type (GN/TP)	*2*
Program Revision	---
Serial Number	*10*

Next, The *Fault Code* for ***each*** failed *test* must be entered. Information about the test must be presented in the form below (separated by semicolons). Each field is associated with its size.:

No	***ind***	***name***	***condition***	***flg***	...	***val/sig***
2	***2***	***2***	***2***	***1***	...	*****

*

A value component (val) is represented in 6 digits.

A signal component (sig) is represented in 3 digits.

The following are few examples :

01;02;01;01;

01;02;01;01;0;-1.232;x;x;

01;02;01;01;0;-1.232;25.334;74.000; ***(Value)***

01;02;01;01;1;003;255;041;007;008;025; ***(Signal)***

Note

The fields must be represented in the required no of digits since the system depends on the positions of the different fields in its recognizing the different fields.

Inconsistent field length will lead to system failure during diagnosis or/inaccurate diagnosis. In case of system failure refer to the appropriate section.

The following are examples of incorrect fault code :

01;02;01;01 (end ; missing)

;01;02;01;01; (extra ;)

01;02;01;01;0; (incomplete)

01;02;01;01;0;23.4;x;x; (value represented in 4 digits instead of 6)

E.2.2.4 Advice Diagnosis

This option can be used if a diagnosis was required for a failure report without entering the report into the system records (PBA files/Knowledge-base). Hence, the user will be prompted only for a fault code for each of the failed tests in the report.

The fault code representation will be identical to that described in section 2.2.3. Refere to section 2.2.1 on the explanation of diagnosis output.

E.2.2.5 System Administration

This option is provided for the system manager to perform some tasks such as; updating complete reports in the PBA files in the knowledge-base, backing up passed boards ....etc.

The user will be prompted for a password inorder to access the administration menu.

E.2.3 Administrator Menu

The administrator menu displays the following options :

<1> ***Update PBAs to KB (Sweep to KB)***
<2> ***Backup Updated PBAs (Sweep to .bk Files)***
<3> ***MS-DOS Window***
<4> ***Monitor Serial Port***
<5> ***Exit System***
<6> ***Return to Main Menu***

E.2.3.1 Update PBAs to KB

Choosing this option will check *each failure report in each PBA file*, and will perform the following :

(i) If the report is updated with a valid fault position number (in the 8th field of the diagnosis record), then the report will be updated in the system knowledge-base and the 9th field in the last record of the PBA file will be set (to the same fault number).

However, if the report is updated with the special set of characters ***, then only the flag is set without updating the report in the system knowledge-base.

(ii) The different knowledge-base elements are backed up to disk from system memory. This will prevent loosing the information if system exit was performed at this stage.

E.2.3.2 Backup Updated PBAs

This option will check each PBA file for the following two conditions:

(i) Board passed test.

(ii) Last report updated in the knowledge-base.

If a PBA file satisfies the above conditions, then the following actions will be performed :

(i) The contents of the PBA file will be appended to the end of the *current* backup file, as long as the backup file size is within its size limit (see sec. 3.1.4), otherwise a new backup file will be created.

(ii) The PBA file will be removed from the system PBA files directory.

E.2.3.3 MS-DOS Window

This option provides an MS-DOS window for different system management tasks, such as copying backup files to floppy, viewing files, ...etc. When this option is chosen the user will be prompted with the following :

VP/ix Z:\USR\USERS\*Current-User*>

Most DOS system commands can be used to manipulate different system files/directories. To Edit a file, the Unix ***vi*** editor can be used. To edit simply type:

vi *filename*

To exit MS-DOS, type QUIT <Return>

E.2.3.4 Monitor Serial Port

This option allows the manager to re-start the serial port monitoring program. This is necessary because if the system is shutdown or rebooted the serial port monitoring process will terminate and hence the process must be re-started without re-booting the system.

The system will warn the user if the previous process is still running.

E.2.3.5 Exit System

Choosing this option will exit the Diagnosis system into the Unix environment.

E.2.3.6 Return to Main Menu

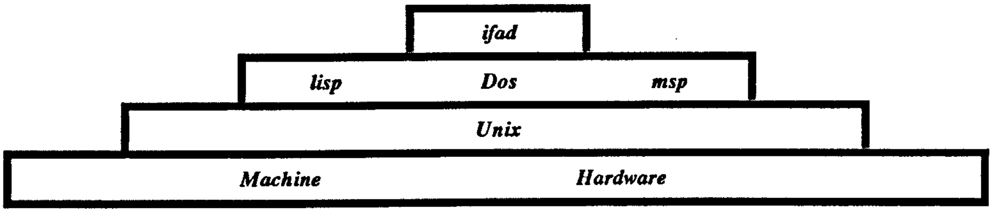
Returns the user to Operator menu.

E.3 The System Environment

This section describes the different system files and the interaction with the Unix environment.

The system environment is formed by the different files/directories which construct the system and/or allow it to perform its task and the user hierarchy upon which the system can be accessed.

The different system files are either (*Information files*) which are used for information back up or initialisation, or the source code files for the system (*Source files*).



The overall system hierarchy

E.3.1 Information files

E.3.1.1 Initialisation files

These are initially generated by the system and are later used to recover some parameters (file names , passwords, . . . etc.) when the system restarts. These are continuously updated as the value of these parameters are changed by the system. These files are described briefly in the following sub-sections.

E.3.1.1.1 .backup.init

Path	***/usr/users/sys/fdsys/***

Contains the number of the current backup file (.bk) used by the system to backup PBA files. If this file is deleted the system uses a backup file with a number which is the first in the range 1-99 (i.e, 1) and creates a new file.

E.3.1.1.2 .pswd.ff

Path	***/usr/users/sys/***

Contains the password for the system administrator in an encrypted form. If the file did not exist, then the administrator will be prompted for a new password and a new file will be created.

E.3.1.1.3 .spinit

Path	***/usr/users/sys/sp/***

Contains the process number of the current serial port monitoring process. The process number is only for processes initiated from within the administrators menu (see sec. 2.2.4).

E.3.1.2 Knowledge-base files

The system starts by reading these files (with the Initialisation files). The existence of these files is important for the system to startup, since they contain the basic knowledge-base parameters which are important for the diagnosis process. The absence of any of these will lead to system failure.

E.3.1.2.1 ft.ff

Path	***/usr/users/kb/***

Contains the component failures with the different symptoms seen for each component. The contents are described in a lisp-specific construct which can be efficiently loaded and manipulated by the system.

E.3.1.2.2 tdist.ff

Path	***/usr/users/kb/***

Contains the statistical information for the different tests encountered by the system. As with the previous file contents, a lisp-specific construct is used to describe the information in this file.

E.3.1.3 PBA files

Path	***/usr/users/boards/***

These files contain the failure/pass records of the different PCBs tested via the LPA (input via the serial port) or entered manually using the appropriate system menu option (see section 2.1.3).

The file names are of the form :

BoardNo.ext

where:

BoardNo: Represents the last 6 digits of the board serial number

ext: brd

E.3.1.3.1 File format

A typical PBA file consists of one or a number of *Board Records* (BRs). A BR could either be a *Failure Record* (FR) or a *Pass Record* (PR) both of which consist of a number of *lines* (Line Records, LRs) which in turn consist of a *sequence of distinct character blocks.*

A FR contains information about the board (represented by the PBA file) at certain instance when it was presented to the tester and failed. However, a PR contains information about the board if no test failure was indicated.

A LR could take one of the following forms :

Test Record

This is contains information about a single test failure during the testing of the board and is of the following form :

LPA No	***Serial No***	***Test Code***	*)*

For information about the different blocks and their sizes refere to section 2.2.3.

Report Header

(	***LPA No***	***Serial No***	*)*

Report Tail

LPA No	***Serial No***	***!/?***	***Ptyp***	***Rev***	***]***	***Date***	***Diagnosis Falgs***

Here, *!/?* is a single character flag indicating either a pass (*!*) or a failure (*?*). *Ptyp* referes to the test program type and *Rev* is its revision. *Date* is the time and at which the report was recorded in the PBA file. Each fields are of two digits seperated by (;). The *Diagnosis Falgs* are a sequence of characters to be during diagnosis and update (refere to the appropriate sections). The Flags are in the following form :

-1	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*	*-1*

The first 7 flags are reserved for the 7 most suspected faults. The 8th flag will be set to the correct fault when updated. The 9th is set when the fault is updated in the knowledge-base. The 10th will be set to the fault category number during fault update.

NB

In all of the cases above, separate blocks and fields within must be isolated by semicolons.

Examples

The following is an example of PBA file content (*686402*.brd) :

```
[10;9201686402;)
10;9201686402;02;05;01;01;)
10;9201686402;02;05;02;01;)
10;9201686402;?;GN;333Z;];21;22;55;08;06;92; -1; -1; -1; -1; -1;
-1; -1; -1; -1; -1
[10;9201686402;)
10;9201686402;02;05;01;01;)
10;920168640202;05;02;01;)
10;9201686402;?;GN;444Z;];21;22;55;08;06;92; -1; -1; -1; -1; -1;
-1; -1; -1; -1; -1
```

E.3.1.4 Backup files

Path	***/usr/users/backup/***

These are large files containing PBA files which satisfied certain conditions (see sec. 2.2.2).

(i) The files have a size limit of 500K for ease of backup to a floppy.

(ii) The file names are of the form

x.bk :*where x is a number (1-99)*

E.3.2 Source files

These are made of a number of files which collectively perform the tasks of fault diagnosis, monitoring the serial port, data organization, backup, ...etc.

E.3.2.1 ifad

Path	***/usr/users/sys/fdsys/***

The lisp code which performs the main fault diagnosis, data entry, and all user interaction procedures. The theory behind the diagnosis technique will be briefly discussed in latter sections.

E.3.2.2 usrid

Path	***/usr/users/sys/fdsys/***

Written in C to interact with the user when the users identity is required to be check by *ifad*. Interacts with the user if the users identity is not known.

E.3.2.3 msp

Path	***/usr/users/sys/sp/***

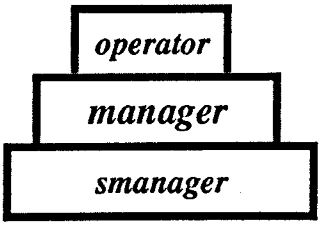
An independent program for monitoring the serial port, buffering the incoming data, organizing them in the appropriate PBA file depending on the serial number of the board, and performing some additions to the data depending on the type of input.

The program starts monitoring the serial port when the Unix is booted up, however, the system administrator, can start the process from within the system menus (as described in 2.2.5).

*Data blocks recognised by **msp** must have a start flag which could either be [or (and an end flag of either) or]. It must be noted that the start falg is not echoed to the PBA file except for **Report Header** blocks (see section 3.1.3.1).*

E.3.3 System Users

Users in the system are classified according to the authorisation hierarchy over the system use and its environment. Three users exist with different authority in the system environment. The users, in the order of increasing authority, are the following :



The following sub-sections will briefly describe each user and his rights within the system.

E.3.3.1 operator

User name	*Home directory*	*Group*
Ericsson operator	*/usr/users/operator*	----

This is the login for the system *fault finder*. This the user with the least authority in the system and hence has no permission to alter any file other than in his own directory.

When an operator logs in, the system starts directly (without providing access to the operating system) and then the operator menu is displayed. An operator cannot exit the system and hence cannot logout (only a person with an access to the system administrator password can be presented with the appropriate menu to exit the system.

Warning

> The only case where an operator could have an access to the lisp interpreter is in the case of system failure. With the appropriate knowledge of Lisp and Unix commands (if!) an operator can end up in the operator home directory, However, the operator cannot perform any modification to files other than his/her own.
>
> In the case of system failure. One of the managers must be informed and the procedure for system recovery must be followed (see section on System failure).

E.3.3.2 manager

User name	*Home directory*	*Group*
sub manager	*/usr/users/manager*	*managers*

The manager has the right of access for editing any of the files which belong to the following; PBA files, Backup files, Knowledge-base files (here, care must be taken). However, he is not the owner of these files.

At login the manager issues commands to the unix system, hence to run the system the command *ifad* must be issued at the shell prompt.

E.3.3.3 smanager

User name	*Home directory*	*Group*
super manager	*/usr/users/manager*	*managers*

The super manager is the owner of all the system files and has the right of access for changing or removing any file or directory which is part of the system environment.

At login the smanager issues commands to the unix system, hence to run the system the command *ifad* must be issued at the shell prompt.

E.4 System maintenance

This section describes some of tasks which occasionally will be required from the system manager during the operation of the system on the factory floor.

E.4.1 System Powerdown

This is a procedure which must be followed before switching the computer off. The procedure performs tasks such as emptying system buffers into the appropriate files, closing all the open files, terminating system demons which start at re-boot, ...etc.

The procedure for a system shutdown is as follows :

1. *login using the user name* ***powerdown***.
2. *Enter powerdown password.*
3. *Answer the questions appropriately, when the system displays the following :*

```
The system is down
Press any key to reboot
```

Either switch off computer or reboot

E.4.2 Trouble-shooting

E.4.2.1 System failure

Currently the system performs some error checking, however, it is not thorough especially when checking the input such as the *test code* (due to the numerous possibilities), fault number, serial Number, program revision, ..etc (these could differ in format for different boards and LPAs and in many cases are alphanumeric). For this reason a system failure is possible.

Such errors could lead to system failure, in which case the system will display the following prompt :

[n] <cl>

At this stage the user is at the lisp level. To recover, the steps in section 4.2.1.3 must be followed.

E.4.2.1.1 Initialisation failure

This could occur at the start when the system starts reading the different files which enable it perform its tasks.

Although, such failures are not possible unless the knowledge-base files are manually edited (the structure of files are corrupted) or permissions are changed manually.

In such cases, the *smanager* can check the initialisation files and knowledge-base files using one/both of the following checks :

1. Check the file content, this could be performed using the ***vi*** editor. The user is advised to make a back up before trying to alter the file contents (e.g, *cp filename filename.bk*).
2. Check the file permissions. This could be done by typing :

   ```
   ls -ls
   ```

   This should only be performed if the permissions in the original system setup were altered. However, the user is warned not to alter these unless it is necessary since the system is adjusted to run under certain conditions and altering these might lead to system failure or lack of system security.

E.4.2.1.2 PBA failure

Another case in which a system failure could occur is during reading a PBA file and identifying the components of the different reports. The reason for such failures is usually an incorrect form for the records in PBA files which is caused by the inability of the system of finding a specific field in a particular record. This could be due to a mistake during manual data input or via the serial port (less likely). Since, error checking is not thorough during receiving/inputting of data.

This kind of failure could occur during any of the following options :

Automatic fault diagnosis

Update PBA record

Advise diagnosis

Update PBAs to KB

Backup Updated PBAs

All the above tasks try to identify specific fields at constant locations, hence if the required field was not found in a report, a failure will occur.

In such cases of failure, the corresponding PBA file could be viewed and edited.

E.4.2.1.3 System recovery

In case of system failure, a system manager must be called to recover the system. It is recommended that the following steps are performed :

1. Note the failure message at system level
2. Type at the error prompt

   :ex (this exits to Unix)

3. Type *ifad* (to start the system)

E.4.2.2 Admin password failure

If the user is not allowed into the administration menu, then the corresponding password initialization file is corrupted. In such cases the super manager must delete the .pswd.ff file.

When the system administration option is chosen next the user will be prompted for a new password.

E.4.2.3 Serial port errors

The serial port monitoring program performs some basic checks on the records received (usually records with unexpected number of characters, however, individual record elements are not examined). If such a record is encountered during reading the serial port, then the whole record will be recorded in an error log file with a name in the form :

Current_date.*elg*

Current_date

A sequence of characters representing the current date (e.g, 100192 for 10-01-1992)

All subsequent erranous records for the current date will be recorded in this file. The file will provide the system manager with a limited information about the source of error (e.g, checking the PBA files for a specific date).

E.5 Theoretical Background of The Diagnosis Technique

In the technique used by the system, 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.

E.5.1 The Knowledge-Base (KB)

The knowledge-base consists essentially of two parts. One part holds the information about each fault. This 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.

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 D-1.

The second part of the knowledge-base stores statistical information regarding the occurrence of each test with respect to each fault.

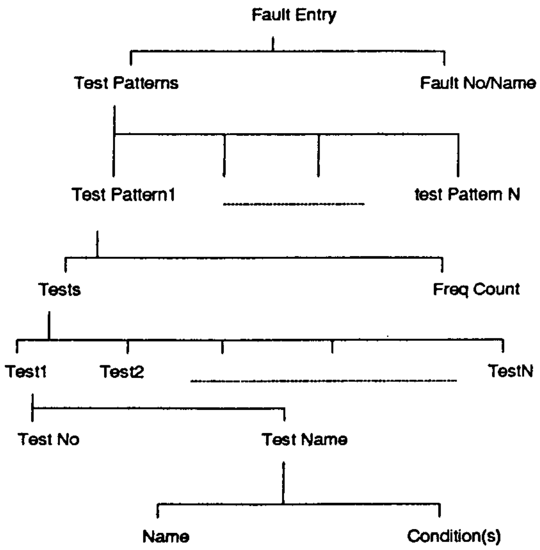


Figure E-1 : The structure of a fault entry

E.5.2 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 belonging to a given fault entry. This process is repeated for all the fault entries in the knowledge-base. The system outputs the fault entries which produced the closest matches.

During comparison, scores for the following are noted :

1) The number of common tests (S_1).

2) The number of common tests relative to the total number of tests in both the match-pattern and the fault entry test pattern (S_2).

3) The sum of the ***Significances*** of the common tests relative to the sum of the ***significances*** of all the tests (S_3).

The Significance of a single test is the relative frequency of the test failing for the fault in question compared with its relative frequency over all the faults. The significance is obtained from the statistical information maintained in the knowledge-base.

4) The frequency count of a test pattern (S_4).

E.5.3 The Rule-Base

The choice of the "best" match from the set of matches is performed by comparing the scores of the different patterns. About 80 *rules* [3] are used to compare the scores. With each rule in the rule-base specifying which of the 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.

Example:

An example rule, which discriminates between two suspect faults F_1 and F_2, each associated with a set of matching scores (see section 2.2), would be of the form:

IF	$[S_{1F1} > S_{1F2}$ & $S_{2F1} < S_{2F2}$ & $S_{3F1} < S_{3F2}$ ...$]$
THEN	***Most Suspected*** $= F_1$

E.5.3 Diagnosis output

The Diagnosis outputs at most 3 suspects with associated percentage confidence scores for each. These scores are derived from the most important of the matching scores relative to each other in the different suspects.

However, when inspecting the matching results the order of the suspects should be the main factor in deciding the best fault since the ordering is the main outcome of the sorting performed by the rule-base which is based on a logical comparison of the scores and their importance in matching rather than a quantitative comparison.

It is recommended to use both attributes in deciding the best fault with emphasising more on the order. For more information about diagnosis refere to section 2.2.1.

REFERENCES

[AMB87] Amble T.: "*Logic Programming and Knowledge Engineering*", Addison-Wesley, 1987, ISBN 0-201-18043-X.

[APF85] Apfelbaum L.: "*An Expert System for In-circuit Diagnosis*", Proc. Int. Test Conf., 1985, No. 23.3, pp. 868-874.

[ARS90] Arslan T., Bottaci L., Taylor G.E.: "*An AI based approach to automatic fault diagnosis for mixed digital/analogue circuits*", IEE Colloquium on mixed analogue and digital circuit test, London, November 1990.

[BEN82] Bennetts R. G. : "*Introduction to Digital Board Testing*", Edward Arnold Ltd., 1982, ISBN 0-8449-1385-0.

[BEN87] Ben-Bassat M., Ben-Arie D. : "*A Case Study With AI-TEST : An Expert System for Electronic Troubleshooting*", Proc. IEEE Int. Automatic Testing Conf., 1987, pp. 363-9.

[BIE81] Biernacki R., Bandler J.: "*Multiple-Fault location of analogue circuits*", IEEE transactions on circuits and systems, vol. CAS-28, No.5, May 1981. pages 361-367.

[BUC89] Buckroyd A. : "*Computer Integrated Testing*", BSP Professional Books, 1989, ISBN 0-632-02042-3.

[BUS86] Bust W. W., Darst C. R., Krysl G. G. : "*ABNER - A Burn-In Monitor and Error Reporting System for PBX Systems Test*", Proc. Int. Test Conf., 1986, No. 2.2, pp. 65-73.

[CHE88] Chen G. -D., Parng T. -M. : "*A Database Management System for a VLSI Design System*", Proc. Design Automation Conf., June 1988, pp. 257-263.

[DAV84] Davis R. : "*Diagnostic Reasoning Based on Structure and Behaviour*", Artificial Intelligence, vol. 24, pp. 347-410, 1984.

[DEK84] De-Kleer J. : "*How Circuits Work*", Artificial Intelligence, vol. 24, pp. 205-280, 1984.

[DEK87] De-Kleer J., Williams B. C. : "*Diagnosing Multiple Faults*", Artificial Intelligence, vol. 32, pp. 97-130, 1987.

[ECK88] Ecklund D. J., Tong F. : "*A Context Mechanism to Control Sharing in a Design Database*", Proc. Design Automation Conf., June 1988, pp. 344-351.

[ELL90] Elliott W., Schneider M. : "*The Learning Aspects of The Fault Finder Expert System*", Proc. 20Th Int. Symp. Multiple-Valued Logic, pp. 378-385, May 1990.

[EVA91] Evans D. : "*Applying Intelligence to Board Diagnostics*", TEST Interfacing Design and Test, pp. 8-11, May 1991, ISSN 0143-2397.

[GUY87] Guyler I. A., Parry P. S., Bayliss J. S. : "*Circuit Testing Using Artificial Intelligence Techniques*", Proc. Third Int. Expert System Conf., pp. 323-31, June 1987.

[HAV89] Havlicsek L. : "*Integrating Diagnostic Knowledge*", IEEE AES Magazine, pp. 54-59, November 1989.

[HOC79] Hochwald W., Bastian J.D. : "*A DC Approach for Analog Fault Dictionary Determination*", IEEE Trans. Circuits & Systems vol. Cas 26, no. 7, Jul. 1979.

[JAC90] Jackson P.: "*Introduction to Expert Systems*", Addison-Wesley, 1990, ISBN 0-201-17578-9.

[KAT89] Kato J., Shimono T., Kawai M. : "*Fault Diagnosis Based on Post-Test Fault Dictionary Generation*", IEEE International Test Conference Proceedings, 1989, paper P.7.

[KEN91] Kennedy-Davies H.: "*AI-based diagnostics for GSM development*", TEST Interfacing Design and Test, pp. 16-19, May 1991, ISSN 0143-2397.

[KOD85] Kodratoff Y.: "*Introduction to Machine Learning*", Chapter eight, Pitman, 1985.

[KRI86] Kriz J., Sugaya H.:"*Knowledge-Based Testing and Diagnosis of Analogue Circuit Boards*", 16th Annual International Symposium on Fault-Tolerent Computing Systems (cat. no. 86CH2335-8), Vienna, Austria, 1-4 July 1986, p.378-83.

[KRO89] Krol J. : "*CIRCOR - An Expert System for Fault Correction of Digital NMOS Circuits*", Proc. European Conf. Circuit Theory and Design, 1989, pp. 674-676.

[LEA88] Lea S. M., Brown N., Katz T., Collins P. : "*Expert System for the Functional Test Program Generation of Digital Electronic Circuit Boards*", Proc. Int. Test Conf., 1988, No. 14.1, pp. 209-220.

[LEE79] Lee J., Bedrosian S. D. : "*Fault Isolation for Analogue Electronic Systems Using the Fuzzy Concept*", IEEE transactions on circuits and systems, vol. CAS-26, NO.7, July 1979, pp. 518-522.

[LIN85] Lin P. M. , Elcherif Y. S. : "*Analogue Circuits Fault Dictionary - New Approaches and Implementation*", International Journal of Circuit Theory and Applications, vol 17, 149-172, 1985.

[LUG93] Luger G., Stubblefield W.: "*Artificial Intelligence: Structures and Strategies for Complex Problem Solving*", The Benjamin/Cummings Publishing Company Inc., 1993, ISBN 0-8053-4780-1.

[MAN86] Mancl D., Sullivan M. : "*A Solution to Test Data Acquisition and Management*", Proc. Int. Test Conf., 1986, No. 2.1, pp. 60-64.

[MAN92] Weyerer M., Goldemund G.: "*Testability of Electronic circuits*", Prentice Hall, 1992, ISBN 0-13-911801-2.

[MAS89] Massara R. E. : "*Design and Test Techniques for VLSI and WSI Circuits*", Peter Peregrinus Ltd., 1989, ISBN 0-86341-165-7.

[MCK89] McKeon A., Wakeling A. : "*Fault Diagnosis in Analogue Circuits Using AI Techniques*", Int. Test Conf., No. 5.2, 1989, pp. 118-123.

[MCK91] McKeon A. : "*The Detection and Location of Faults in Analogue Circuits*", Ph.D. Thesis, January 1991, , Dept. Elec. Eng., Imperial College of Science and Technology, University of London.

[MIC75] Micalski R.: "*Variable-valued Logic and its Applications to Pattern Recognition and Machine Learning*", In Rine , pp. 506-34 , 1975.

[MIC81] Micalski R., Chilausky R.: "*Knowledge Acquisition by Encoding Expert Rules Versus Computer Induction from Examples: A Case Study Involving Soybean Pathology*", In Mamdani and Gaines, pp.247-71, 1981.

[MIC83] Michalski R.S., Carbonell J.G., Mitchell T.M.: "*Machine Learning: an AI Approach*", Tioga Publishing Company, Palo Alto, 1983.

[MIT77] Mitchell T. M.:"*Version Spaces: A candidate elimination approach to rule learning*", Proceedings IJCAI-77, 1977.

[MIT78] Mitchell T. M.:"*Version Spaces: An Approach to Concept Learning*". PhD thesis, 1978, Stanford University, Stanford, CA.

[ODR85] Odryna P., Strojwas A.J. : "*PROD : A VLSI Fault Diagnosis System*", IEEE Design & Test, pp. 27-35, December 1985.

[PAH82] Pahwa A., Rohrer A. : "*Band Faults: Efficient Approximation to Fault Bands for the Simulation before Fault Diagnosis of Linear Circuits*", IEEE Trans. Circuits & Systems vol. Cas 29, pp 82-88, 1982.

[PFL89] Pflueger K.W. : "*Hybrid Diagnostics Strategy for an Expert System Controlled Automatic Test System (EXATS)*", IEEE AES Magazine, pp. 25-30, October 1989.

[PRA90] Prasad V. C., Pinjala S.N. : "*A Fast Algorithm for the Generation of Fault Dictionary of Linear Analogue Circuits Using Adjoint Network Approach*", Proc. Int. Symp. Circuits and Systems, pp. 37-40, 1990.

[PUC92] Pucknell D., Eshraghian: "*Basic VLSI Design: Systems and Circuits*", Prentice Hall, 1992, ISBN 0-7248-0105-7.

[QUI86] Quinlan J.: "*Induction of Decision Trees*", Machine Learning 1 (1): 81-106, 1986.

[QUT289] Al-Qutayri M.: "*Development of testing procedures for mixed mode (Analogue/Digital) integrated circuits - Progress Report No.2*", ASCOT progress reports, ALVY Project, CAD012, University of Bath, 20th February 1989.

[QUT88] Al-Qutayri M.: "*Development of testing procedures for mixed mode (Analogue/Digital) integrated circuits - Progress Report No.1*", ASCOT progress reports, ALVY Project, CAD012, University of Bath, 24th November 1988.

[QUT90] Al-Qutayri M.A., Shepherd P.R. : "*On Testing of Mixed Mode Integrated Circuits*", Journal of Semicustom ICs Vol. 7, No. 4, 1990.

[RAP83] Rapisarda L., Decarlo R. : "*Analogue Multifrequency fault diagnosis*", IEEE transactions on circuits and systems, vol. CAS-30, NO.4, April 1983. pages 223-234.

[RAT86] Ratford V., Keating P. : "*Integrating Guided Probe and Fault Dictionary: An Enhanced Diagnostic Approach*", IEEE International Test Conference Proceedings, 1986, paper 10.3.

[RIC89] Rich S. H., Venkatasubramanian V. : "*Causality-Based Failure-Driven Learning in Diagnostic Expert Systems*", AIChE Journal, vol. 35, No. 6, pp. 943-950, June 1989.

[RIC91] Rich E., Knight K.: "*Artificial Intelligence*", McGraw-Hill Inc, second edition 1991, ISBN 0-07-052263-4.

[ROGC89] Rogel-Favila B., Cheung P. : "*Combinational and Sequential Circuit Fault Diagnosis Using AI Techniques*", Int. Test Conf., 1989, No P.16.

[ROGD89] Rogel-Favila B., Cheung P. : "*Deep Reasoning Approach to sequential circuit Fault Diagnosis*", Proc. European Conf. Circuit Theory and Design, 1989, pp. 665-669.

[SCH79] Schreiber H. : "*Fault Dictionary Based Upon Stimulus Design*", IEEE Trans. Circuits & Systems vol. Cas 26, no. 8, Jul. 1979.

[SHA48] Shannon C.: "*A Mathematical Theory of Communication*", Bell System Technical Journal, 1948.

[TAN88] Tanaka H., Kawai M., Sugasaki I., Hakuba T. : "*System Level Fault Dictionary Generation*", IEEE International Test Conference Proceedings, 1988, paper 41.2.

[TAN90] Tanimoto S. L.:"*The elements of artificial intelligence using common lisp*", Computer science press 1990, ISBN 0-7167-8230-8.

[TAY90] Tayeh M. M., Diantonio S. A. : "*Troubleshooting Hybrid Circuits Using Artificial Intelligence Technology*", Hybrid circuit Technology (CIMFLEX TECH. CORP., Pittsburgh, PA, USA), vol. 7, pp. 12-19, Jan 1990.

[TUR90] Turino J. : "*Design to Test*", Van Nostrand Reinhold, 1990, ISBN 0-442-00170-3.

[VIS84] Visvanathan V., Szeto E.,Tits A. : "*A Robust Simulation before Test Technique for DC Analog Fault Diagnosis*", Proc. Int. Symp. Circuits and Systems, pp. 689-692, 1984.

[WAW89] Wawryn K., Zinka W. : "*A Prototype Expert System for Fault Diagnosis in Electronic Device*", Proc. European Conf. Circuit Theory and Design, 1989, pp. 667-680.

[WEY84] Wey C.L.: "*On the implementation of an analogue ATPG: the nonlinear case*", IEEE International Symposium on Circuits and Systems. Proceedings (Cat. No. 84CH1993-5), Montreal, Que., Canada, 7-10 May 1984 (New York, USA:IEEE 1984), p.213-16 vol.1.

[WEY85] Wey C.L., Saeks R.: "*On the implementation of an analogue ATPG: the linear case*", IEEE transactions on instrumentation and measurement, vol. IM-34,No.3, September 1985. pages 442-449.

[WIL85] Wilkinson A. J. : "*MIND: An Inside Look at an Expert System for Electronic Diagnosis*", IEEE Design & Test, August 1985, pp. 69-77.

[WIL86] Wilkins B. R. : "*Testing Digital Circuits*", Van Nostrand Reinhold (UK) Co. Ltd, 1986, ISBN 0-442-31748-4.

[WIN75] Winston P.: "*Learning Structural Descriptions from Examples*", In The Psychology of Computer Vision, ed. P. Winston, New York: McGraw-Hill, 1975.

[WIN86] Winkel M. : "*Using a Relational Database to Develop a Statistical Quality Control System for ATE*", Proc. Int. Test Conf., 1986, No. 2.3, pp. 74-79.

[WU82] Wu C., Nakajima K., Wey C., Saeks R.: "*Analogue fault diagnosis with failure bounds*", IEEE transactions on circuits and systems, vol. CAS-29, No. 5, May 1982. pages 227-284.