MACHINE LEARNING OF FAULT CHARACTERISTICS FROM ROCKET ENGINE SIMULATION DATA

Min Ke
Moonis Ali

Center for Advanced Space Propulsion
The University of Tennessee Space Institute
Tullahoma, TN 37388

Abstract

Transformation of data into knowledge through conceptual induction has been the focus of our research described in this paper. We have developed a Machine Learning System (MLS) to analyze the rocket engine simulation data. MLS can provide to its users fault analysis, characteristics, and conceptual descriptions of faults, and the relationships of attributes and sensors. All the results are critically important in identifying faults.

I. Introduction

An important component of intelligent diagnostic systems is the knowledge which human experts employ in analyzing and diagnosing faults. However, this knowledge employed is very limited in the sense that it is based on a very small number of observed situations. Since human experts have not seen all possible instances of all faults, they cannot describe fault characteristics sufficiently well to make diagnostic decisions. We have developed a Machine Learning System (MLS) for analyzing the SSME simulator data to generate characteristics about engine faults. In MLS inductive heuristics and domain knowledge are employed to guide the inductive process. With two phases of abstractions as well as a knowledge management system, MLS can be applied to a wide spectrum of domain tasks. MLS has been tested with SSME simulator data. MLS consists of two levels of abstractions. Section II presents the general algorithm, Section III describes the basic abstraction, Section IV describes the advanced abstraction, and Section V discusses the tests and results.

II. The General Algorithm

MLS consists of two levels of abstraction: the basic abstraction which generates the characteristic descriptions, discriminant descriptions and aggregational descriptions for a concept such as an engine fault; and the advanced abstraction which groups similar concepts into a concept hierarchy to form a higher level of concepts. MLS incorporates concept instance data incrementally.

In MLS the paradigm for inductive inference can be formulated as follows:

Before induction we have:

- A hierarchy of nodes with leave nodes representing basic concepts, internal nodes representing clusters, and descriptions of concepts at each node;
- Knowledge bases which include knowledge about attributes, components, relationships and basic concepts, and also include deductive rules, generalization rules, transformation rules and aggregation rules;
- A new instance description of a basic concept represented in MLS's representation language.

After induction we get:

- An extended hierarchy of concepts such that either a new basic concept node is added or an old basic concept's descriptions are modified to cover the new instance; the structure of the hierarchy and the cluster node's descriptions are modified to incorporate the new instance.

Let C be a basic concept, and let e be a new instance of C. A raw description is a description of raw data in our representation language. In the knowledge base, appropriate transformation rules, aggregation rules, deductive rules, generalization rules and inductive heuristics are provided. It is assumed that e is a set of simple expressions (i.e., atoms). The following is the outline of the general algorithm of MLS:

1) Read in e, raw description of an instance of concept C.
2) Apply simple transformation and attribute-level aggregation on e, generating result e1.
3) Apply deductive transformation and component-level aggregation on e1, generating result e2. New attributes and relationships are generated in this step.
4) Generalize concept C's characteristic descriptions to cover e2.

This research was supported by NASA Grant Nos. NAGW-1195, and NAG-1-513 and Rocketdyne Contract No. R04QBZ90-032709.

5) Generalize concept C's discriminant descriptions and specialize (or eliminate) discriminant descriptions of other concepts.

6) Modify the concept-level aggregation descriptions.

7) Modify the generalization hierarchy above the basic concept level. The most frequent operation is to modify (adding, generalizing, or changing the weight of) the descriptions of the concept class on the higher levels. Other operations include creating a new class and deleting an unqualified class.

The algorithm described here is only one process of a single instance. As an incremental method, the above algorithm can be repeatedly applied to many incoming instances.

III. Methods and Algorithms for the Basic Abstraction

MLS is an incremental learning system, so it is efficient to incorporate new instances. MLS has a capability of rich logic representation. Multiple-valued nominal attributes and inexact value matching are MLS's capabilities not shared by most other systems.

Each time an instance of a specified concept is incorporated into the concept hierarchy, the object of the concept and the object of the instance are sent to the procedure that incrementally modifies the characteristic description of the concept. The algorithm for construction of characteristic descriptions can be described as follows:

1. Check whether the basic concept is a new concept. If it is a new concept, the instance description is taken as the characteristic description of the concept and the concept hierarchy should be extended to incorporate the new basic concept. If it is not a new concept, then perform the following steps.
2. For each expression in the instance object, try to find the matching expression in the concept object.
   If not found, put the expression into the Alternative List AL.
   If found, generalize the two matching expressions, use the resultant expression as one expression of the characteristic descriptions, increase the count of instances that imply this expression, and compute the worth of the new expression.
3. Apply the Add-Alternative Rule to the unmatched expressions of the concept characteristic descriptions and expressions in AL. The resultant expressions are put into the concept characteristic description. Also some unqualified expressions are eliminated from the characteristic description of the basic concept.
4. Call the bottom-up modification procedure in the advanced abstraction. Use only confident features of the concept to modify the concept hierarchy. If the consistency factor of an expression is greater than the consistency factor threshold, then the expression is considered a confident feature. Since all characteristic features are complete (completeness factor is 1.0), only the consistency factor needs to be considered.

The Algorithm for Constructing Discriminant Descriptions is given below:

The inputs to this algorithm are an object of concept C1 and an object of C1's instance.

For each expression EXP in the instance description perform the following steps:

1. Obtain all matching expressions from the uniqueness table UT.
2. If there is no matching expression from UT, then EXP is a unique feature.

   Add EXP to the discriminant description of C1.

   Add an entry of EXP in UT. Exit.
3. If there are matching expressions, then try to find inconsistency. Let ML be the list of matching expression entries.
   For each entry in ML, check for inconsistency (Loop A)
   1) If the expression EXP is a discriminant feature of another concept C2, then an inconsistency situation is found.

   Delete EXP from the discriminant description of C2.

   Set the entry uniqueness flag off.

   Exit Loop A.
   2) If EXP is covered by a discriminant feature of another concept C3, then an inconsistency situation is found.

   Specialize the discriminant feature of C3.

   Modify the entry in UT.

   Exit Loop A.
   3) If EXP is covered by an inconsistent feature (an inconsistent feature is a feature that is already identified as nondiscriminant), then EXP is inconsistent. Exit Loop A.
4. If EXP is not found to be inconsistent in step 3, then check for partial inconsistency.
   For each entry in ML, if the expression in the entry is partially covered by EXP, then specialize both expressions;

modify the discriminant descriptions of the concept specified in the entry;

modify the entry.

5. If EXP is matching a discriminant feature EXP1 of C1 (in which case EXP is said to be compatible), then

perform the least generalization on EXP and EXP1;

modify C1's discriminant description

modify the entry in UT.

6. If EXP is neither inconsistent nor compatible, then

add EXP to C1's discriminant description; create an entry for EXP in UT.

IV. Methods and Algorithms of the Advanced Abstraction

The advanced abstraction is an integration of two incremental processes: 1) modification of the cluster hierarchy, and 2) modification of cluster descriptions. The results of the advanced abstraction are a clustering of basic concepts and conceptual descriptions of the clusters. The combination of incremental processes with an expressive logic representation language makes MLS a unique system in the conceptual cluster area. In this section, we illustrate how the matching factor is computed; describe how to measure the quality of clustering; discuss the operations on the concept hierarchy as well as the algorithms of hierarchy extension and modification; discuss the cluster parameters; and analyze the time cost of the advanced abstraction.

Features in a cluster description will be constantly modified during the incremental inductive process. A feature has a *confidence count* which may be increased or decreased. The confidence count of a feature determines whether the feature is confident or not. Only the set of confident features is used as the description of a cluster.

When the first instance of a new basic concept is incorporated into the concept hierarchy, a top-down extension process is performed. The procedure to perform top-down extension of the concept hierarchy is a recursive one. The first call uses the root node as one of the parameters. The root is a universal cluster node which incorporates all instances input to the inductive system. The description of the root is a generalization of all the incorporated instances.

A description of the procedure, nameed extend-hierarchy, is given below where *currentnode* is the current cluster, and *newconcept* is the new basic concept to be incorporated:

extend-hierarchy (currentnode newconcept)

(1) Modify the current cluster:
   a. Increase the size (by 1) of the current cluster.
   b. For each expression E in the cluster description perform the following steps:
      i. If there is no matching expression of E in *newconcept*'s confident features, then see whether the expression confidence count of E is still greater than the feature retaining threshold FR-TH. Delete E if the expression confidence count of E is less than FR-TH. Retain E if the expression confidence count of E is not less than FR-TH.
      ii. If there is a confident expression E1 in *newconcept* that matches E and E1 is covered by E, then increase the expression confidence count of E by 1, and compute the worth of the expression.
      iii. If the confident feature E1 is not covered by E, but the value matching factor of E1 and E is greater than the feature matching threshold FM-TH, then generalize E to incorporate E1 and increase the expression confidence count of E by 1, and compute the worth of the expression.
      iv. If E1 is not covered by E and value matching factor of E1 and E is less than FM-TH but greater than the feature conflict threshold FC-TH, then do nothing. If the value matching factor is less than FC-TH, then decrease the expression confidence count of E by 1, and see whether the expression confidence count of E is less than FR-TH. Delete E if the count is less than FR-TH. Retain E if the count is not less than FR-TH.
      v. Add all unmatched expressions of *newconcept's* confident features to *currentnode*'s description.

(2) In the current cluster find each subcluster S that is close to *newconcept* and whose description does not violate constraints of *newconcept*. Closeness is measured by the matching factor between S and *newconcept*. A closeness threshold C-TH is used to determine whether a subcluster is close enough to *newconcept*.

(3) Find each basic concept directly under the current cluster that is close enough to *newconcpt*.

(4) If there are no close subclusters and basic concepts, then put *newconcept* directly under the current cluster.

(5) Find the best object from those close subclusters and basic concepts according to the quality measure of clustering. Here the quality measure is modified to include basic concepts. Since a basic concept does not have a subcluster or subconcept, the cluster matching factor is replaced by the matching factor of the basic concept and *newconcept*.

(6) If the best node is a basic concept, then combine the best node with *newconcept* to form a new subcluster of the current cluster: Create a new cluster. Generalize expressions of the descriptions of the two concepts. Two expressions are generalized only if their value matching factor is greater than FM-TH (0.7). The new expression's confidence count is the sum of the two counts of the generalized expressions. The new expression's worth is the average worth of the two generalized expressions.

(7) If the best node is a subcluster, then recursively call:
extend-hierarchy (bestnode newconcept)

When incorporating a new instance to an existing basic concept, the description of the basic concept may be generalized to cover the new instance. This modification may cause further modifications on the predecessors of the basic concept.

For each expression EXP in the instance's description, if it is not covered by a matching expression of the basic concept, the following procedure is performed:

Let P be the parent cluster of the basic concept C, OLDEXP be the expression in C that matches EXP, NEWEXP be EXP or the generalization of EXP and OLDEXP.

modify-hierarchy(C, OLDEXP, NEWEXP)

Get C's parent P.

Try to find EXP1, the expression in P that matches OLDEXP.

If not found (EXP1 is empty), then add NEWEXP to P's description.

If EXP1 is not empty, NEWEXP is EXP, OLDEXP covers NEWEXP, and the value matching factor of OLDEXP and EXP1 is greater than FM-TH, then increase the confidence count of EXP1 (Now NEWEXP is taken as the supporting feature of EXP1).

If EXP1 is not empty, NEWEXP covers OLDEXP, and the value matching factor of NEWEXP and EXP1 is greater than FM-TH, then generalize EXP1 to incorporate NEWEXP and increase the confidence count of EXP1.

If EXP1 is not empty, NEWEXP covers OLDEXP, and the value matching factor of EXP1 and NEWEXP is less than FC-TH, then NEWEXP is considered to be contradict to EXP1.

- The confidence count IC of EXP1 is decreased by 1.
- Test IC to see whether it is less than FR-TH.
- If IC is less than FR-TH, EXP1 is deleted from P's description; check the number of expressions in P's description; if the number is less than the cluster retaining threshold CR-Th, then delete P from the concept hierarchy and reassign all P's leaves and subclusters under P's parent (note: the root can never be deleted).

If P has a parent, then recursively call:
modify-hierarchy(P, OLDEXP, NEWEXP).

After modifying the concept hierarchy, try to combine C with one of its sibling concept nodes and create a new cluster, since after the modifying of the concept description, the concept may become close to another basic concept under the same cluster node.

V. Analysis and Results

Engine test analysis is one of several application areas of inductive learning, where conceptual descriptions about different faults can be automatically generated from a large number of fault instances, and similar faults can be classified into clusters. The inductive results, including high level characteristic descriptions and discriminant descriptions of faults and the concept hierarchy with descriptions of clusters, can be used to aid the fault test analysis and be used for fault diagnosis.

The Space Shuttle Main Engine (SSME) is one of the most complex reusable liquid-fuel (oxygen and hydrogen) rocket engines. Each time a test on SSME is performed, a huge amount of data is collected from many sensors. Many highly-trained engineers are required to perform a thorough investigation of the tests. Two difficulties are presented for the improvement of test analysis: (1) As more tests are performed and more thorough investigations are required, more experienced engineers are needed; (2) more senior staff with many year's experience are leaving. To overcome these difficulties, a computer conceptual induction technique is used to aid the engineers in analyzing the test data. In addition to its efficiency in forming concepts and generating concept descriptions, a computer inductive system can accumulate knowledge from both the data of many tests and the expertise of the engineering staff.

After SSME simulator data is input into MLS, a concept hierarchy is built by the inductive system. The concept nodes on the hierarchy represent various engine faults. The cluster nodes on the hierarchy represent higher level concepts, each of which describes a group of similar engine faults. Descriptions of engine faults or fault groups are also generated by the inductive system and stored in the nodes on the hierarchy. Features about any attributes, sensors or relationships can be easily accessed by a user. The expertise of the engineering staff can be incorporated into the system as concept constraints, deductive rules and various inductive biases.

SSME simulator generates the raw data about sensors for a fault. The raw data is simply a list of time-value pairs of each sensor. All values are in real

number form. Usually this kind of data is used to plot diagrams (Figure 5.1) for representing sensor behavior, and then human experts analyze the diagrams to find the characteristics of each fault. This human analysis process is usually time-consuming when the number of diagrams is large, and is complicated when the features of faults involve the relationships between sensors. Since a human expert describes the features of a fault by a set of attributes (which are shown in Table 5.1 and Table 5.3), MLS will automatically generate those human-oriented descriptions from the raw data.

Before sending the raw data to the induction system, preprocessing is performed which smooths the curves of sensors, divides the curves of sensors into segments and denotes each segment as an event. The collection of sensor descriptions constitutes the description of an instance of an engine fault which, in turn, is used for the induction process. MLS emphasizes a significant event for each sensor, since most characteristic features of a fault exist in the significant event. The preprocessor extracts basic attribute features from the raw data (as shown in Table 5.1) and deduction process generates more attribute features to describe a fault. The derived attributes are shown in Table 5.3.

Although MLS can perform induction on a domain without all the related domain knowledge, domain knowledge makes the induction more time-efficient and produces better results. A discussion of the domain knowledge needed in the SSME application area is given below.

A basic concept is an abstraction of a class of real world entities which share common features. A real world entity is a thing (such as an animal or a computer) or a situation (such as a disease, a machine fault, or a state of a process). In the SSME fault test analysis, each type of engine fault is a basic concept. MLS assumes that every real world entity belongs to only one basic concept. In the SSME domain we assume each fault instance represents only one fault. MLS takes in the instances of the basic concepts to incrementally generalize the descriptions about the basic concepts and to classify them into clusters. The basic concepts should be the main focus of an application domain if the purpose is to find the features of the basic concepts. In the SSME fault test analysis domain, the purpose is to find features for each fault and possible classification of faults. That is why engine faults are taken as basic concepts. In some application areas where the purpose is clustering, we can use basic concepts to represent every real world entity. In this case, MLS does not perform the generalization in the basic abstraction; the main task performed is in advanced abstraction.

The matching threshold for grouping basic concepts into classes is related to the number of levels in the hierarchy. In the SSME domain, faults can be grouped into several classes such as injector faults, control faults, duct faults, manifold faults, valve faults, high pressure oxidizer turbopump faults, and high pressure fuel turbopump faults. The possible number of levels of interesting high level concepts is one or two. By this kind of domain knowledge and purpose of clustering we choose the maximal level number to be three in MLS.

A structural concept has components whose features and relationships collectively constitute the description of a concept. For example, in a block world domain each block can be a component of a basic concept. In a cancerous cell analysis domain cell bodies are components of the basic concepts – cells. In a rocket engine fault analysis domain, an engine fault is described by features of temperature, pressure, flow, speed, etc. These parameters are measured by many sensors. Therefore, sensors are the components.

In a multi-concept inductive system components usually have different relevancies with different concepts. For example, a sensor may have distinct features for an engine fault and show nothing about other faults. A large number of sensors exist in the rocket engine, but only a few of them are related to a specific fault. To pay equal attention to all sensors for every fault is inefficient. Therefore, a component in MLS can be assigned different relevancies for different concepts. In building MLS the assignment of sensor-fault relevancies depends on domain knowledge such as functional relationships and structural relationships of the engine parts as well as locations of sensors and faults. Examples of component-concept relevancies in MLS are shown in Table 5.2. In this table, s10, s22, etc. are sensors; CCV, MOV, MFV, OPOV, and FPOV are five types of engine faults. In MLS sensors can be in one of three types with respect to each engine fault: critical sensors — which show strong evidence of and are closely relate to the fault; irrelevant sensors — which do not show any changes when the fault occurs; and unspecified sensors — which may show some change with the fault and whose relationship to the fault is unknown. As indicated in Table 5.2, critical sensors are assigned a high relevancy value (10.0); irrelevant sensors are given a low relevancy value (2.0); and unspecified sensors are given a value between those two values (7.0 is assigned to a temperature sensor and 8.0 is assigned to a pressure sensor). From the domain knowledge we know that pressure sensors are usually more important in identifying a fault than temperature sensors. Thus, pressure sensors are given higher relevancy values than temperature sensor.

Relationships of component features play an

important part in the description of concepts. Components may have positional relationships, temporal relationships, or some relationships governed by domain theory. MLS allow a system developer to indicate interesting component pairs. Then, deductive rules, which derive component-relationship descriptions from component features, are automatically generated. Examples of these kinds of rules in MLS are shown in Figure 5.2. Known relationships between component features can be used as concept constraints.

Attributes are the basic vocabulary to describe basic concepts. After the identification of basic concepts the system developer needs to find out what attributes should be used in modelling the domain problem. The choice of attributes is based on the *availability* and *utility*. Availability tells what attributes can be abstracted directly from the input raw data. Usually too many attributes can be abstracted from the raw data, but only a small portion is relevant and useful to the concept description. The utility of an attribute, which tells what attributes are usually used to describe a basic concept, is determined by experts with domain knowledge. Attributes determined by domain experts may not be available, so the constructive rules should be formulated so as to derive the unavailable attributes from the available attributes.

In MLS attributes are of different importance in describing a concept. Domain knowledge can be used to assign different worth values to different attributes. For example, in the SSME domain the attribute `direction` of change is more important than the attribute `rate` of change because a sensor's direction of change is usually the same for the same fault while the rate of change can be different for different severities and durations of the fault. In many cases, relationships between attributes are more important than individual attributes. MLS represents every attribute by an object and supports the knowledge acquisition facilities to help the system developer to define attributes. The "curve-pattern" attribute has a hierarchical domain which is represented by a list representation of tree. The slot "correspondence" is the transformation rule which transforms the input values into the symbolic values (for example, the value 'rf' stands for a two-event curve pattern with the first event as 'rising' and the second event as 'falling').

Inductive rules include generalization rules and transformation rules. Generalization rules are supplied by the inductive system. Transformation rules are domain related. Domain knowledge is needed to determine how to divide a real number value-domain into categories, and what symbol represents a value range. For example, the value of attribute `temperature` can be categorized into { `high, very-high, medium, low, very-low` } . For different domains the categories may cover different value ranges. There are no universal rules of transformation; the only criteria are that categories of values should correspond to the categories of concept instances, and that symbols need to reflect the value ranges in real world applications.

Since useful attributes may not be available directly from the raw data, deductive rules are used to derive them by applying various domain knowledge such as domain theory, physical laws, operational principles and domain experience. From domain knowledge in the flight engine test, we know that the attributes "starting time", "ending time", "changing rate", and "magnitude" have little value to characterize an engine fault, because they all change with the severity or duration of an engine fault. Different faults may have the same changing rate, and faults of the same type may have different changing rates. We found some relationships have more importance in characterizing engine faults. For example, a temperature sensor and a pressure sensor at the same location of the engine have certain relationships for a specific fault. New attributes START-TIME-DIFF (difference of the starting time) and END-TIME-DIFF (difference of the ending time) are used to represent the temporal relationships; RATE-RATIO (ratio of the changing rate) and MAGNITUDE-RATIO (ratio of the magnitude) are used to represent the quantitative relationships. The derived attributes in MLS are shown in Table 5.3. Logical relationships, like the concurrency of changing trend, is represented by the logical connective AND. In the domain of SSME test analysis, human experts recognize certain features and relationships for different faults. This kind of knowledge can be used as concept constraints. For example, (assuming an open loop situation) an increasing pressure of a valve inlet will cause an increasing pressure of the valve outlet. This rule is applicable to all valve blockage faults. In MLS this rule is expressed as:

IF (direction (s23) = ?x)
THEN (direction (s27) = ?x)

and

IF (direction (s23) = ?x)
THEN (direction (s28) = ?x)

where s23 is the pressure at the outlet of the high pressure oxidizer pump booster which is also the inlet to FPOV (fuel preburner oxidizer valve) and OPOV (oxidizer preburner oxidizer valve), s27 is the pressure of the outlet of OPOV, and s28 is the pressure of the outlet of FPOV. Examples of the deductive rules in MLS are shown in Figure 5.3.

The SSME simulation data of 61 instances about five valve-blockage faults is used to run the indutive system. Attributes have different worth values and sensors have different relevancy values based on whether a sensor is a critical sensor to a fault, an

irrelevant sensor or an unspecified sensor.

The concept hierarchy of the induction is shown in Figure 5.4. We can see that the induction gives quite good clustering. There are no faults grouped with different types of faults on the second level. On the first level of the hierarchy, the clusters show strong regularity. Cluster104 corresponds to the MFV blockage fault, cluster81 corresponds to the CCV blockage fault, cluster78 corresponds to the MOV blockage fault, and cluster73 corresponds to the OPV and FPOV blockage faults. In the hierarchy we can see that the OPV and FPOV faults have similar sensor behavior. Examples of the MSL output are shown in Table 5.4. and Table 5.5. In Table 5.4. a part of the characteristic description of Main Fuel Valve (MFV) blockage fault is given, where S122, S8, S9 etc, shown in the first column of the table, are sensor labels. The rest of the columns in the table illustrate the association of attributes with the corresponding values for various sensors. The association of an attribute with a value of a sensor is called a feature. The columns two to four illustrate atomic features which involve only one attribute. The other three columns illustrate the compound features which are conjunctions of two atomic features. There are different forms of attribute values in the Table. For example, sensor S122's attribute direction has a single-form of value POS; sensor pair S38/S41's attribute Rate-ratio has a range-form of value 4..5; sensor pair S9/S10's attribute Magnitude-ratio has a or-form of value 5V1. Similarly, in Table 5.5.a part of the discriminant description of MFV is given, which consists of features possessed by MFV's instances but not by any instances of other faults.

VI. Conclusion

We have developed an inductive machine learning system, MLS, for the acquisition of knowledge about SSME faults. Given fault data from an engine simulator as input, MLS will generate characteristic, discriminant and aggregational descriptions about each engine fault. MLS also generates a concept hierarehy which groups related faults into clusters. The descriptions about each cluster are higher level descriptions of fault groups. The output from MLS can be used for assisting engineers in analyzing engine tests and for engine fault diagnosis.

We have tested MLS with 61 fault instances from the SSME Simulator. Five valve blockage faults are included in those instances. MLS can correctly classify the faults with a high rate of success. Human oriented descriptions about faults and fault groups are generated.

Domain knowledge plays an important role in MLS. More knowledge enables the learning system to become more efficient and more accurate. For its future development, more domain knowledge about the engine needs to be added. Further research efforts are also needed to combine AI techniques with traditional statistical data analysis techniques.

Acknowledgement

This work was performed within the Center for Advanced Space Propulsion (CASP) and was supported in part by NASA Grant NAGW-1195, NAG-1-513 and Rocketdyne Contract No. R04QBZ90-032709. The Center for Advanced Space Propulsion is part of The University of Tennessee–Calspan Center for Aerospace Research, a not-for-profit organization located at UTSI. The authors would like to thank A.M. Norman for his assistance during the course of this effort.

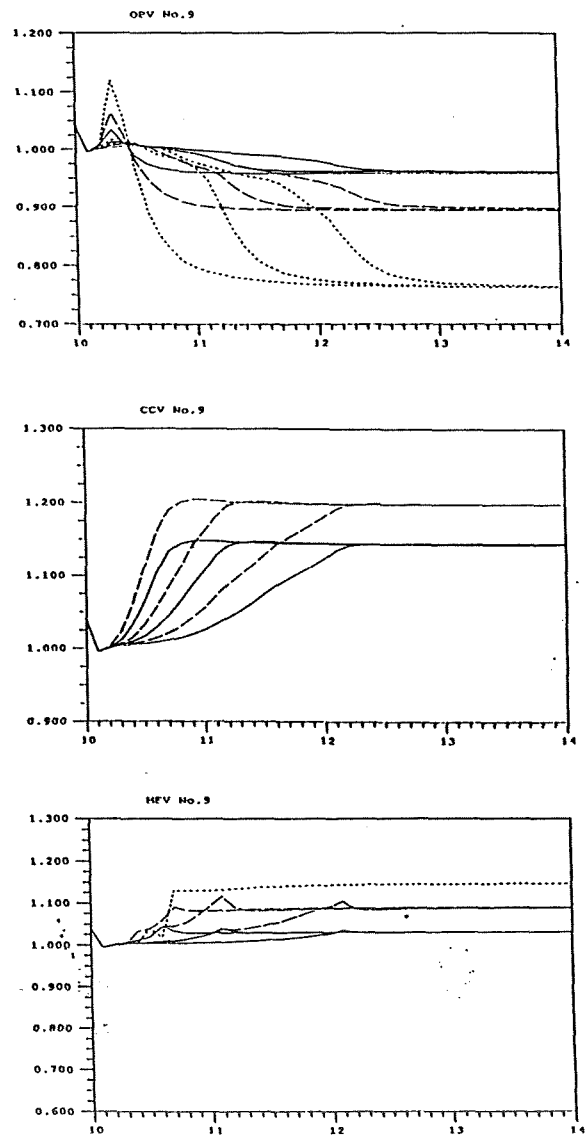

Figure 5.1 Sensor Behavior.

```
(d-rule4 if ((direction (?comp1) = pos)
             (direction (?comp2) = pos))
         then add ((increase-together (?comp1 ?comp2))))

(d-rule5 if ((direction (?comp1) = neg)
             (direction (?comp2) = neg))
         then add ((decrease-together (?comp1 ?comp2))))

(d-rule6 if ((direction (?comp1) = pos)
             (direction (?comp2) = neg))
         then add (( opposite-trend (?comp1 ?comp2))))

(d-rule7 if ((start-time (?comp1) = ?st1)
             (start-time (?comp2) = ?st2)
             (= ?st1 ?st2))
         then add ((change-concurrently (?comp1 ?comp2))))
```

Figure 5.2 Deductive Rules for Generating Component-Component Relationships.

```
(d-rule1 if ((start-time (?comp) = ?x)
             (end-time (?comp) = ?y))
         then add ((duration (?comp) = (- ?y ?x))))

(d-rule12 if ((start-time (?s1) = ?v1)
              (start-time (?s2) = ?v2)
              (<= ?v1 ?v2))
          then add ((activated-before (?s1 ?s2))))

(par4 if ((magnitude (s23) = ?m1)
          (magnitude (s27) = ?m2))
      then add
           ((magnitude-ratio (s23/s27) = (/ ?m1 ?m2))))

(d-rule11 if ((start-time (?comp1) = ?v))
          actions ((add-minmax 'first)))

(dd0 if ((direction (s9) = ?r1)
         (direction (s36) = ?r2))
     then add ((and (direction (s9) = ?r1)
                    (direction (s36) = ?r2))))

(rr0 if ((rate (s9) = ?r1)
         (rate (s36) = ?r2))
     then add ((rate-ratio (s9/s36) = (/ ?r1 ?r2))))
```

Figure 5.3 Deductive Rules in ETID.

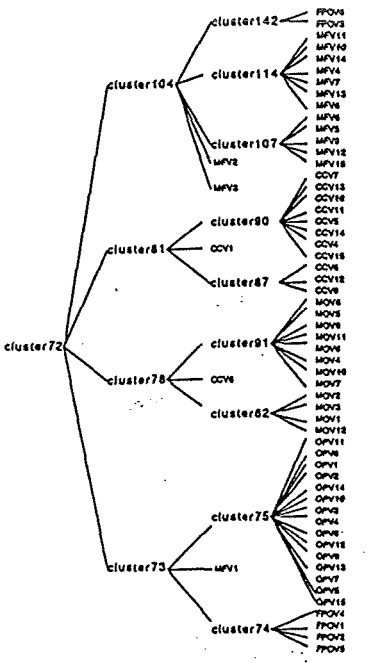

Figure 5.4 Hierarchy of Induction with Differentiating Attribute Worth and Component Relevancy.

ATTRIBUTES	EXPLANATIONS
START-TIME	starting time of an event
END-TIME	ending time of an event
RATE	average changing rate of an event
DIRECTION	indicate changing trend of an event
MAGNITUDE	the absolute amount of change of an event
AVG-VALUE	the average value of an event
VALUE-TO-NORMAL	indicate whether the average value of an event is above, same of or below the normal value
CURVE-PATTERN	indicate the general pattern of the curve
STABLE-LEVEL	the value of the sensor after it became stable
STABLE-LEVEL-TO-NORMAL	indicate whether the stable value of a sensor is above, same of or below the normal value
NUM-OF-EVENT	the number of events for a sensor
DURATION	the difference between the START-TIME and END-TIME

Table 5.1 Attributes Directly Extracted from Raw Data.

SENSORS	CCV	MOV	MFV	OPOV	FPOV
S10	10.0	8.0	10.0	8.0	8.0
S22	2.0	10.0	8.0	8.0	8.0
S30	8.0	8.0	8.0	10.0	8.0
S55	2.0	7.0	2.0	7.0	7.0
S121	7.0	7.0	2.0	7.0	7.0

Table 5.2 Sensor-Fault Relevancies.

ATTRIBUTES	EXPLANATIONS
INCREASE-TOGETHER	both events are rising together
DECREASE-TOGETHER	both events are falling together
OPPOSITE-TREND	two events have the opposite trend
CHANGE-CONCURRENTLY	two events have the same starting time
TEMPORAL-REL-TYPE1	two events have the same starting time and ending time
FIRST-START-TIME	the event has the earliest starting time
ACTIVATED-BEFORE	one event activated before the other
LARGEST-DURATION	the event has the largest duration
RATE-RATIO	the ratio of two events' rate
MAGNITUDE-RATIO	the ratio of two events' magnitude
AVG-VALUE-RATIO	the ratio of two events' AVG-VALUE
STABLE-VALUE-RATIO	the ratio of two events' stable level
START-TIME-DIFF	the difference of two events' starting time
END-TIME-DIFF	the difference of two events' ending time

Table 5.3 Derived Attributes.

sensor	direct-ion	curve-pattern	rate-ratio	stable-level-to-normal & value-to-normal	direction & direction	magnitude-ratio & stable-level-ratio
S122	POS	Rising		higher, greater		
S38/S41			4..5			
S8	POS	FF		higher, greater		
S9, S10					POS, POS	
S9/S10						5 V1, 5

Table 5.4 Characteristic Description of MFV.

sensor	direct-ion	curve-pattern	rate-ratio	stable-level-to-normal & value-to-normal	direction & direction	magnitude-ratio & stable-level-ratio
S12	NEG	FSFVFR		lower, less		
S33	NEG	Falling		lower, less		
S38/S41			4..5			
S9/S38						6, 1
S9/S10						1, 5

Table 5.5 Discriminant Description of MFV.