Transactions on Information and Communications Technologies vol 6, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

Generation of fault trees from simulated incipient fault case data

M.G. Madden & P.J. Nolan
Department of Mechanical Engineering, University College, Galway, Ireland

Abstract

Fault tree analysis is widely used in industry in fault diagnosis. The diagnosis of incipient or 'soft' faults is considerably more difficult than of 'hard' faults, which is the situation considered normally. A detailed fault tree model reflecting signal variations over wide range is required for diagnosing such soft faults.

This paper describes the investigation of a machine learning method for the automatic generation of fault trees for incipient faults. Features based on the FFT (Fast Fourier Transform) of the time response simulations are used are used to provide a training set of examples comprising records of fault types, severity and feature list. The algorithm presented, called IFT, is derived from the ID3 algorithm for the induction of decision trees. A significant aspect of this approach is that it does not require any detailed knowledge or analysis of the application system. All that is needed is a 'black-box' model of the system; i.e. knowledge of what faults arise from measurable quantities taking on particular values.

The proposed procedure is illustrated using detailed simulation results for a servomechanism typically found in machine tool applications and the results to date indicate the feasibility of the approach.

1 Introduction

Fault tree analysis (FTA) and fault tree synthesis (FTS) evolved primarily within the US aerospace and nuclear industries and have been extensively used in systems safety analysis for over 30 years. Fault tree analysis can be valuable as a design tool; using it to identify and eliminate potential sources of accident in a system can help can help prevent costly design changes and retrofits (Barlow and Lambert [2]). The technique can also be used as a diagnostic aid; in the event of system failure, it can predict what the most likely causes of the failure are by evaluating all combinations of basic events (e.g. component failures) which can lead to a top event (a particular fault). This is the target application of the work presented here.

Fault tree synthesis involves the construction of fault trees, which can be a very time-consuming task and requires considerable engineering expertise. Initially, FTS was performed manually and even today manual construction is not uncommon in industry. The procedure has been formalised by Haasl [6] into a 'structuring process' based on a functional model of the plant using schematics, piping diagrams, process flow sheets and so on. With the wider availability of computers, a number of authors have attempted to automate the procedure. Systems exist for electric networks (Fussell [5]), chemical plants (Powers, Tompkins & Lapp, [7]) and nuclear systems (Cummings [4]).

Rather than taking the above 'top-down' approach, the 'mini-fault tree' approach of Taylor, [8] uses mini-fault tree models for components. The advantage of this and other component based models (e.g. that of Lapp and Powers [9]) is that models can be reused in different studies. Over the years, a number of refinements in the technique have been proposed; a major review can be found in Bossche [3].

The approach presented here is an alternative to the usual approach of synthesizing fault trees from mini-fault or component models, in that example cases are used to deduce fault trees and thus detailed process knowledge is not required.

2 Induction of Fault Trees

Fault trees represent how basic observable system conditions (at the leaf nodes of the tree) combine to result in an undesired event (the root event of the tree). Thus the task of generating a fault tree becomes that of identifying which system conditions combine in what ways to result in undesired events. The approach proposed here is to provide examples of the system conditions which were observed for different undesired events, as well as examples of system conditions when the system is behaving normally, and to generalise from these examples to find higher-level rules about how system conditions relate to undesired events. Thus, the overall problem can be seen as one of inductive learning from examples.

An algorithm called IFT has been developed to induce fault trees from example data. It is based on Quinlan's ID3 algorithm for induction of decision trees [10]. A decision tree is basically a way of representing classification rules. If we consider classifying systems on the basis of whether they do or do not exhibit symptoms of a particular fault, then the decision tree has the same information content as the fault tree. This observation is the genesis of the IFT algorithm.

When selecting an attribute (step 3 of the algorithm), any of the set of attributes may be chosen. The choice of attribute will determine the complexity of the resulting tree, since some attributes will partition the data more cleanly into positive and negative examples than others. In the spirit of Occam's Razor principle, it is desirable to find the most compact fault tree possible — this fault tree would be expected to encapsulate best the important discriminating features of a fault and should have the greatest predictive power. In order to determine which is the optimal attribute to select, the information gain heuristic proposed by Quinlan [10] in the context of decision trees is used.

The IFT algorithm is:

Input: a set of classified examples. Each example consists of a vector of numeric attributes and a true/false flag indicating whether or not the vector of values is associated with a particular fault.

Begin with the undesired event as the top node of the tree.

1. *If all examples are true, stop.*
2. *If all examples are false, remove the parent node and stop.*
3. *Otherwise, the parent node is not sufficient in itself to classify the data and must be refined. This is done by replacing the parent node by an AND node with the parent node on one branch.*
4. *To develop the other branch, select an attribute (A_N).*

   Find a threshold point (T) which is the weighted mean of the values taken on by the attribute in the example set. This defines two basic events for the fault tree: $A_N < T$ and $A_N \geq T$. It also partitions the data into two subsets.

   Create an OR node linking the results of applying the algorithm recursively to the two data subsets defined by the two events.
5. *After generating the tree, remove degenerate (single-input) AND and OR nodes.*

3 Simulated System

The sample system chosen to investigate the proposed method, which is typical of servomechanisms found in the machine tool industry, is shown in Figure 1. It comprises a constant-field armature-controlled d.c. motor driving an inertial load through a gearbox and shaft having finite flexibility.

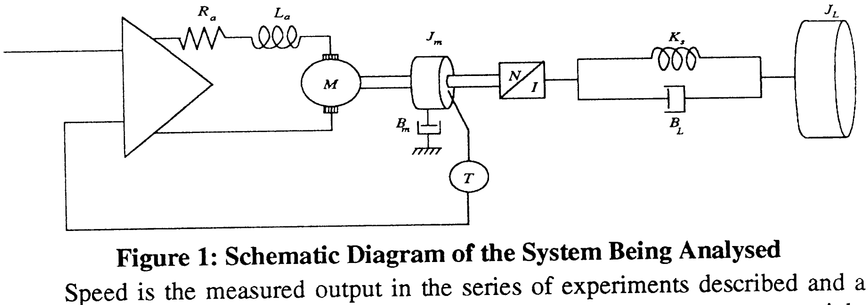

Figure 1: Schematic Diagram of the System Being Analysed

Speed is the measured output in the series of experiments described and a disturbance noise in assumed. Other signals, for example armature current, might be particularly useful, either as an alternative to or in addition to the speed signal, in diagnosing faults. The simulation model is based on deriving a set of first order nonlinear differential equations for the system. This in turn is simulated using a standard general purpose simulation language e.g. ACSL [1]. Assumed incipient failures (soft faults) are represented by appropriate parameter settings based on random variation over selected ranges. Random variation within each fault severity level band was assumed. Furthermore, *all* system parameters were assumed to vary randomly within a selected band (typically 2 - 5%) of their nominal values. The time response was determined for a step input in command reference. Typical results are shown in Figure 2.

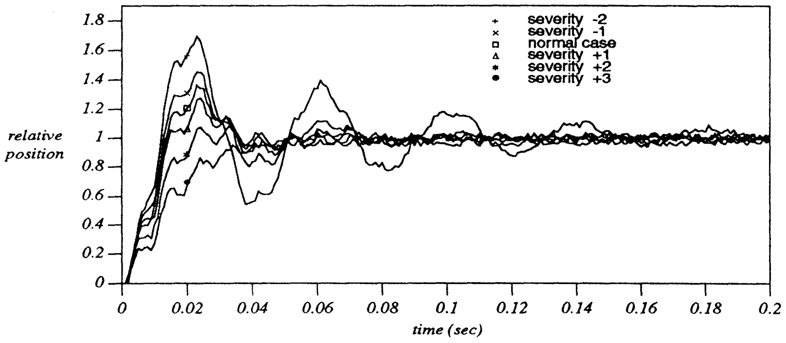

Figure 2: Time Response for Different Armature Resistance Fault Levels

About one thousand simulation runs were carried out in total. Other incipient failures included changing motor, load and shaft damping, changing amplifier performance and malfunction of the tachometer feedback loop. This gave rise to a total of 20 different *system states:* the desired state in which the system is operating normally and 19 undesired states consisting of the different severity levels of the different faults. A noise component was added to make the experiments as realistic as possible.

4 Feature Extraction

In order to employ the simulation results as a training set in the induction algorithm, it is necessary to extract some features or attributes which can be used. There are various ways in which the time response can be characterised. One approach would be to use standard control system metrics (based on time or frequency response) such as natural frequency, damping ratio, peak value, time to peak, percentage overshoot, rise time, settling time, bandwidth and so on. An alternative approach would be to consider some kind of data fit to the model (e.g. polynomial, exponential and so on) and subsequently to use the derived coefficients as the features. A third approach would be to use the FFT (Fast Fourier Transform) which characterises the system in terms of the frequency response evaluated at discrete frequency values.

The FFT approach was selected for this work. The procedure was first to "detrend" the time response by subtracting the average value. Subsequently, a windowing function (in our case a Hanning or cosine bell window) is used to force the time response to be zero at the beginning and end of the time sample.

5 Test Results

The performance of the algorithm has been analysed using the standard approach of splitting the available data at random into two sets, one for training and one for testing. Each item in the training data consists of a vector of raw FFT values and a label identifying which system state is associated with the item. Then in testing, the fault tree diagnosis procedure is supplied with the vector of FFT values and attempted to determine the system state from it. In the four series of experiments carried out, the simulation data used consisted of FFT vectors for six system states: the normal operational state and the five severity levels for faults associated with changing armature resistance.

The first series of experiments were 'yes/no' experiments; the challenge for the diagnosis procedure was to determine for each FFT vector presented to it whether that vector was or was not associated with a given fault. As an example, the training data was used to construct a fault tree for the system state in which there is a *moderate increase in armature resistance*. A sample fault tree which was generated by the algorithm is shown in Figure 3. Notice that the algorithm constructs a binary fault tree; this can be simplified further by collapsing nodes of the same type in series to produce multiple-input AND and OR nodes. After a fault tree had been constructed, the diagnosis procedure used this tree to attempt to identify whether data from the testing set was or was not associated with that fault state. This procedure was carried out for different sizes of training sets. (For the tree of Figure 3, the data used in generation comprised 30% of the available data.)

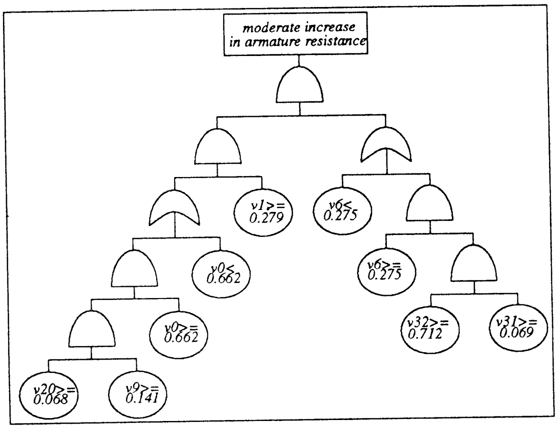

Figure 3: Fault Tree for Moderate Increase in Armature Resistance

The results (not listed) showed promising diagnostic performance; after training on just 10% of available data, the diagnosis procedure was able to correctly identify 83.5% of the data on which it had not been trained, and its performance increased to being able to correctly identify 97% of the unseen data after being trained on 90%. (The procedure was, of course, able to correctly identify 100% of the data it had actually been trained on.)

In order to investigate the performance of the diagnosis procedure in more detail, a second series of experiments was carried out. These used the same data set and methodology as the first experiments did, but were more comprehensive in examining the performance of the fault trees in diagnosis. Results are presented in Figure 4. Each of the dashed lines in Figure4 represents a learning curve for 'yes/

no' identification of a single system state, averaged over two runs in which different training sets were selected at random from the total data set. For reference, the average of all of the dashed lines is plotted in the figure as a solid line.

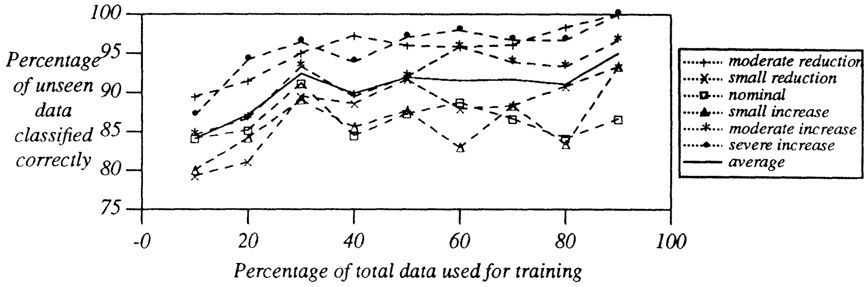

Figure 4: Averaged Results for 'Yes/No' Testing for Several Fault States

It may be seen from these results that there is an overall improvement in the diagnostic ability of the fault trees when the set of examples used in generating the tree is increased. It may also be noticed that the diagnosis is most accurate for the most extreme levels of the fault type under consideration: for the 'moderate reduction' and 'severe increase' cases, a tree generated from 90% of the data can correctly classify all of the remaining data, whereas the success rate of a tree generated from 90% of the data for the 'nominal' case is only 87%. This seems intuitively obvious, as we would expect more extreme levels of the fault type to be easier to diagnose. Some evidence of overtraining may also be noted in Figure 4, in that some of the curves show performance on unseen data actually weakening when a larger training set is used; in particular, for the 'nominal' case performance on unseen data is better when trained on 30% of the data than when trained on 90% of the data.

Note however that the 30% tree classifies 93.8% of the *total* data set correctly, whereas the 90% tree classifies 98.7%, so in one sense its performance is better — while it does not classify as high a proportion of unseen data correctly, it has dealt with a much higher volume of data during its generation. Examining this effect more closely, the performance of the fault trees in classifying the total data set, rather than just the set of unseen data, is shown in Figure 5.

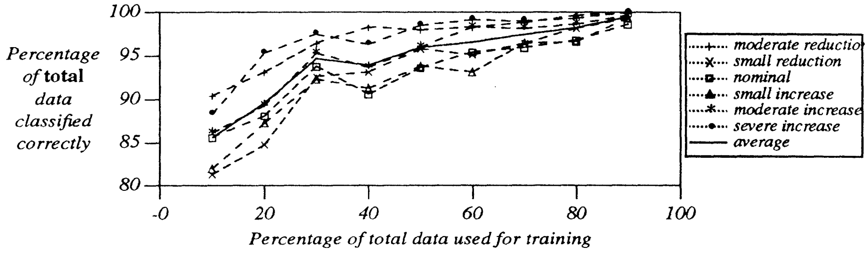

Figure 5: Results of 'Yes/No' Testing on Training and Testing Data Sets

Having obtained quite good results for the 'yes/no' experiments, a third series of experiments was carried out. These were categorisation experiments; the challenge for the diagnostic system was to attempt to name the fault associated with a given FFT vector. The group of trees trained for the different faults can be applied in turn to a FFT vector to come up with a list of potential diagnoses for a fault. A category test can give one of the following results:

- A single category, which is one of 'normal' or 'undesired state X'.
- A list of categories, if it fails to uniquely identify the system state category.
- 'Unknown', indicating it cannot find any category of system state for the FFT vector.

The results (not plotted) of this series of experiments showed performance rising from being able to categorise 45% of unseen data correctly when trained on 10% of the data set, increasing to 77% of unseen data when trained on 90% of the data set. These results may not at first glance appear to be as good as those of the 'yes/no' experiments discussed above — for example, the trees in the 'yes/no' experiments identified on average 83% of unseen data when trained on 10% of the data. However, the challenge in this series of experiments is a more difficult one. Since the 'yes/no' experiments gave binary results, a diagnostic procedure acting totally at random might classify 50% of data presented to it correctly, whereas here there are 6 possible results, so a diagnostic procedure acting at random would only classify 16.67% of data correctly.

In order to examine where the errors arise in categorisation, vectors for each of the six system states being considered were drawn from the data set and presented to the diagnostic procedure for classification. The classifications returned by the diagnostic procedure (after training on 40% of the data) are shown in Figure 6, where the 'Unknown' category represents vectors that the procedure failed to classify as belonging to any of the six states known to it. The figure shows that, apart from vectors the diagnostic procedure failed to classify, in general the mistakes it made tended to result from confusing data for adjacent system states. It also highlights something that was mentioned previously with reference to Figure 4: as one would expect, the diagnostic procedure tends to be more successful in identifying the states associated with extreme fault levels — those labelled '-2' and '3' in Figure 6 — than other system states, particularly the state associated with normal operation of the system.

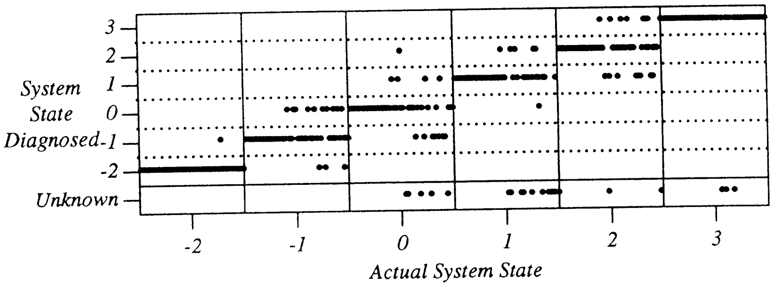

Figure 6: Sample of System States Predicted in Category Testing

6 Discussion and Conclusions

This paper has presented a practical approach to the problem of generating detailed fault trees for incipient faults in dynamic systems based on time response measurements, using an algorithm to induce fault trees from data. While there are other machine learning techniques for induction and classification which could be used — for example, neural network approaches — the advantage of the approach adopted here is that the fault trees produced provide a clear engineering representation of the reasoning employed by the system in identifying fault states. These fault trees are suitable for inclusion in a rule-based diagnosis system. The approach uses a training set derived by extracting features derived from a FFT of the time response. Detailed simulations were used to produce the time responses for the fault cases.

The operation of the proposed methodology is independent of whether simulation or actual test data is used. Even when real test is available, it is expected that detailed simulation results will be necessary to compliment the training set with examples of additional fault situations.

The results to date are very encouraging especially when one considers that we are attempting to predict 'soft' or incipient faults where the response of the faulted system and unfaulted system are close.

References

1. ACSL. *Advanced Continuous Simulation Language*, Mitchell & Gauthier Associates, Concord, Massachusetts, 1987.
2. Barlow, R.E. and Lambert, H.E. Introduction to Fault Tree Analysis, *Reliability And Fault Tree Analysis,* eds. R.E. Barlow and J.B. Fussell, 1975.
3. Bossche, A. *Fault Tree Analysis and Synthesis*, Ph.D. Thesis, Dept. of Electrical Engineering, Technical University of Delft, 1988.
4. Cummings, G.E. Application of the Fault Tree Technique to a Nuclear Reactor Containment System, *Reliability And Fault Tree Analysis,* eds. R.E. Barlow and J.B. Fussell, 1975.
5. Fussell, J.B. Computer Aided Fault Tree Construction for Electrical Systems, *Reliability And Fault Tree Analysis,* eds. R.E. Barlow and J.B. Fussell, 1975.
6. Haasl, D. F. Advanced Concepts in Fault Tree Analysis, *Proceedings of System Safety Symposium,* Seattle, Washington, 1965.
7. Powers, G.J., Tompkins, F.C. & Lapp, S.A. A Safety Simulation Language for Chemical Processes: A Procedure for Fault Tree Synthesis, *Reliability And Fault Tree Analysis,* eds. R.E. Barlow and J.B. Fussell, 1975.
8. Taylor, J.M. An Algorithm for Fault Tree Construction, *IEEE Trans. Reliability,* Vol. R-31, pp. 137-146, 1982.
9. Lapp, S.A. and Powers, G.J. "Computer-Aided Synthesis of Fault Trees", *IEEE Trans. Reliability,* Vol. R-26, pp. 2-12, 1977.
10. Quinlan, J.R. Induction of Decision Trees, *Machine Learning,* Vol. 1, pp 81-106, 1986.