Efficient Basic Event Ordering Schemes for Fault Tree Analysis

L.M. Bartlett and J.D. Andrews
Mathematical Sciences Department
Loughborough University
Loughborough
Leicestershire
LE11 3TU

Summary

Fault Tree Analysis is a commonly used means of assessing the system reliability performance in terms of its component's reliability characteristics. More recently, significant advances have been made in methodologies to analyse the fault tree diagram. The most successful of these developments has been the Binary Decision Diagram (BDD) approach. The Binary Decision Diagram approach has been shown to improve both the efficiency of determining the minimal cut sets of the fault tree and also the accuracy of the calculation procedure used to determine the top event parameters.

To utilize the Binary Decision Diagram approach the fault tree structure is first converted to the BDD format. Implementing the conversion of the tree is relatively straight forward but requires the basic events of the tree to be placed in an ordering. The ordering scheme chosen is critical to the size of the BDD produced, and hence the advantages of this technique. Alternative ordering schemes have been investigated and no one scheme is appropriate for every tree structure.

The work presented in this paper takes a machine learning approach based on Genetic Algorithms to select the most appropriate ordering scheme. Features which describe a fault tree structure have been identified and these provide the inputs to the machine learning algorithm. A set of possible ordering schemes has been selected based on previous heuristic work. The objective of the work detailed in the paper is to predict the most efficient of the possible ordering alternatives from parameters which describe a fault tree structure.

1. Introduction

Over the past five years an alternative technique, to the Kinetic Tree Theory (Vesely[1]), known as the Binary Decision Diagram (BDD) method has been developed[2-7] to analyse the fault tree. This method has proved to be more accurate and efficient than the conventional approaches. In calculating the system or top event parameters it does not need to first evaluate all the minimal cut sets, nor does it require the use of approximations, the exact calculations can be performed.

To take advantage of these features, the fault tree constructed to represent the system failure mode must first be converted to a BDD. To accomplish this the basic events in the fault tree are placed in an order. A good ordering of the basic events can result in a very efficient analysis, a poor ordering can lead to problems.

Several research papers have been published which investigate different ordering strategies for the BDD[8-9]. As yet no rule based approach to identify a scheme which yields an efficient ordering for each tree has been produced. Lack of this efficient ordering for any tree structure is probably the reason that no commercially available code has been produced which is based on this method.

This paper presents a method of identifying a 'good' ordering scheme based on a machine learning approach, in conjunction with a genetic algorithm.

2. Binary Decision Diagrams

A BDD is a directed acyclic graph, as shown in figure 1. All paths through the BDD start at the root vertex and terminate in one of two states, either a 1 state which corresponds to a system failure, or a 0 state which corresponds to a system success. A BDD is composed of terminal and non-terminal vertices, which are connected by branches. Non-terminal vertices correspond to the basic events of the fault tree.

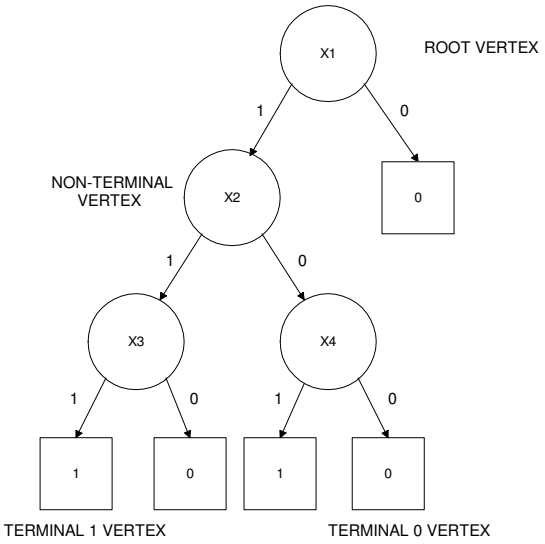

Figure 1: A Binary Decision Diagram.

All the left branches leaving a vertex are the 1 branches and all the right branches the 0 branches. Every path starts from the root vertex, and proceeds down through the diagram to the terminal vertices. Only the vertices that lie on a 1 branch on the way to a terminal 1 vertex are included in the path. All the paths terminating in a 1 state give the cut sets of the fault tree. For example, the cut sets for the BDD shown in figure 1 are:

1) X1,X2,X3 2) X1,X4.

2.1. Variable Ordering

In constructing the BDD, the ordering of the basic events is crucial to the size of the resulting diagram. Using an inefficient ordering scheme will produce a non-minimal BDD structure. Alternative ordering schemes will produce BDD's of different sizes, the smaller the BDD the more optimal the diagram. To illustrate this fact, consider the

simple fault tree shown in figure 2. The tree has four basic events, where X2 is repeated.

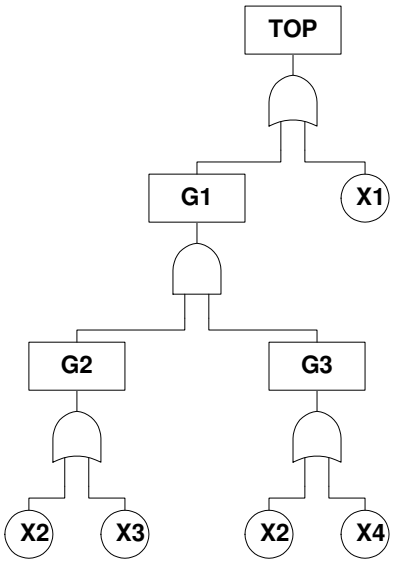

Figure 2 : A simple fault tree.

If the basic event ordering permutation of X1<X2<X3<X4 is taken, the resulting BDD is shown in figure 3. This structure consists of only four nodes, it is a minimal structure and hence produces only minimal cut sets.

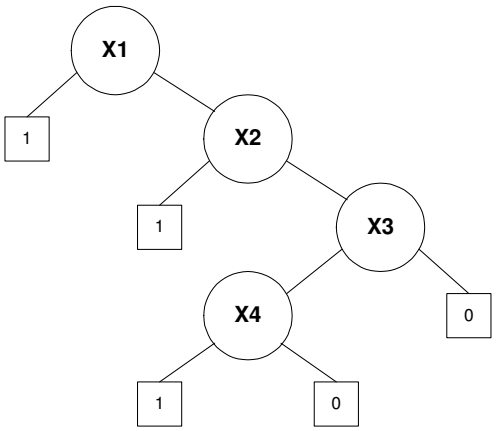

Figure 3 : Resulting BDD from ordering X1<X2<X3<X4.

However, if the alternative ordering permutation of X4<X3<X2<X1 is taken the resulting BDD consists of seven nodes, it is non-minimal and yields non-minimal cut sets (shown in figure 4). From this result it can be shown that for larger fault tree structures the resulting BDD would be much larger, and in the worst case of using a poor ordering permutation, the diagram may be unsolvable.

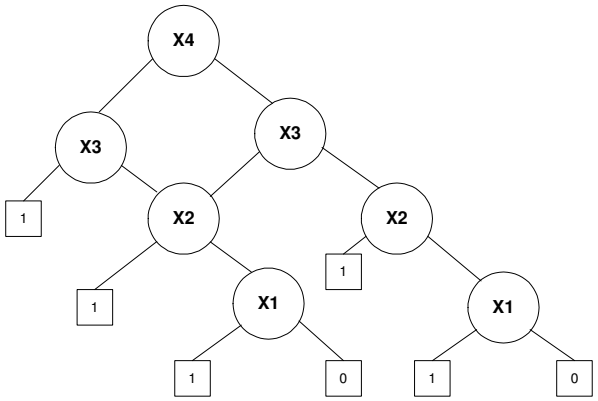

Figure 4 : Resulting BDD from ordering X4<X3<X2<X1.

The objective would be to produce an ordering scheme which achieves the 'best' BDD. Numerous studies have investigated the effects of variable ordering schemes on the BDD size and it has been shown[9] that there is no universal scheme which will always guarantee the 'best' BDD formation, and the most appropriate scheme must be selected depending on the characteristics of the fault tree.

3. Pattern Recognition Approaches

One approach to the problem of variable ordering is to consider pattern recognition techniques. There are many alternatives. These include classifier systems[10], ID3[11], neural networks[12], Bayesian methods and fuzzy logic[13]. There is no way to determine which of these would be the most effective to apply to the ordering problem. Therefore, classifier systems have been initially selected and investigated in this paper.

4. Machine Learning Systems - The Classifier Model

Classifier systems are a kind of rule-based machine learning system, with general mechanisms for processing rules in parallel, for testing the effectiveness of existing rules and for adaptive generation of new rules[21]. A classifier system, depicted schematically in figure 5, consists of three main components:

1) Rule and message system.
2) Apportionment of credit system.
3) Rule/message generation system.

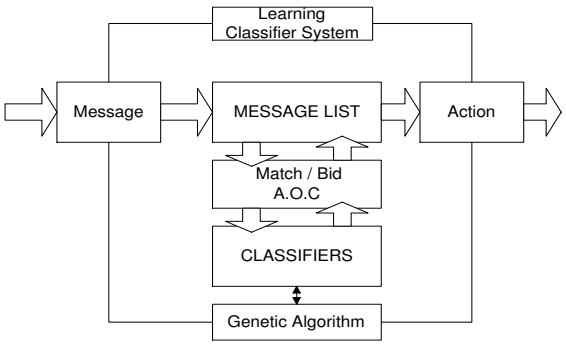

Figure 5: A Classifier System.

The environment posts an input message which is sent to the internal message list. The messages activate classifiers, matching classifiers bid to place their messages on the message list. In the case of more than one match the winning classifier is decided by a bidding process in the apportionment of credit system. Once a winning classifier has been found, this then posts its message to the message list, and the process is repeated several times until an action is then produced. The input corresponds to a fault tree structure and the action specifies the best way of ordering the basic events.

4.1 The Rule and Message System.

The rule and message system forms the computational backbone of the machine learner. A **classifier** consists of two main parts: (1) a message part; and (2) a condition part. It has simple syntax:

$$\text{classifier} \Rightarrow \text{condition} : \text{message}$$

A **message** within the classifier system is a means of information exchange, and is simply a finite length string over some finite alphabet, in terms of a binary alphabet, we have:

$$\text{message} \Rightarrow \{0,1\}^n$$

Thus, the message is defined as a concatenation of 0's and 1's of length n. The **condition** is a simple pattern recognition device where a wild card character (#) is added to the underlying alphabet.

$$\text{condition} \Rightarrow \{0,1,\#\}^n$$

Thus, a condition matches a message if at every position a '0' in the condition matches a '0' in the message, a '1' matches a '1', and a '#' matches either.

Both the condition and the message parts of a classifier contain coded characteristics of the problem and coded outputs of the problem. An example condition or message part of a classifier is:

0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1
coded characteristics | action or output
which describe problem | of the problem

Once a classifier matches a previous message from the message list, whether it posts its message is determined by the outcome of the bidding process determined in the apportionment of credit system.

4.2 Apportionment of Credit Algorithm (A.O.C)

The A.O.C is the competition phase of the classifier system, implemented in terms of a bidding process. This bidding process is referred to as the bucket brigade algorithm. A classifier, when matched, makes a bid proportional to a product of its strength and the bid coefficient.

Winning classifiers must then pay their bid to other classifiers for matches rendered. The matched and activated classifier divides its bid, to those classifiers responsible for posting the messages that matched the bidding classifiers condition. This division of payoff among contributing classifiers helps to ensure the formation of an appropriate

sized subpopulation of rules[10].

To illustrate the workings of the A.O.C system we consider five classifiers, see table 1. Assuming initial strength values of 500 for all five classifiers, we post the initial input (environmental) message 0101, where {01} represents the characteristics of the problem and the {01} represents the action/output of the problem. We assume a bid coefficient of 0.1 and take the bid as the product of this bid coefficient, and strength. In the initial step (t=0), classifier 1 and classifier 4 match the input and bid 50 units, since they both have the same bids the tie is broken by random selection, the winning classifier sends its message during the next time step. Classifier 4 pays its bid to the party responsible for its activation; in this case, the input (environment) strength is increased by 50 units as the environmental message was responsible for activating classifier 4. In subsequent time steps, activated classifiers make their payment to previously active classifiers. Finally, at time step 4, a reward comes into the system and is paid to the last active classifier, classifier 3 where the action corresponds to the input action (i.e. the last two digits are the same).

		---------- t = 0 ----------				---------- t = 1 ----------			
	Classifier	Strength	Message	M	Bid	Strength	Message	M	Bid
1	01## : 0000	500		e	50	450			
2	00#0 : 0011	500				500			
3	11## : 1001	500				500			
4	0#01 : 0001	500		e	50	450	0001*	4	45
5	##01 : 1100	500				500		4	50
	Environment	0	0101			50			

		---------- t = 3 ----------				------- t = 4 --------		
	Classifier	Strength	Message	M	Bid	Strength	Message	Payoff	Strength
1	01## : 0000	450				450			450
2	00#0 : 0011	500				500			500
3	11## : 1001	500		5	50	450	1001	1000	1450
4	0#01 : 0001	455				455			455
5	##01 : 1100	450	1100			500			500
	Environment	50				50			50

*when two bids are equal, tie broken by random selection.

Table 1: Apportionment of Credit System.

5. Rule and Message Generation System

Injecting new possibly better rules into the system requires the use of a rule / message generation system, in this instance a genetic algorithm.

5.1 Genetic Algorithms

Genetic algorithms (G.A's) are search algorithms based on the mechanics of natural selection and natural genetics[1] . They originated from the studies of Holland and colleagues at the University of Michigan[15]. A genetic algorithm is composed of three operators: Reproduction, Crossover and Mutation.

Reproduction is a process in which individual strings are copied according to their fitness values. The reproduction operator is implemented by creating the equivalent of a biased roulette wheel in the computer, where each current string in the population has a roulette wheel slot sized in proportion to its fitness. The effect of roulette wheel parent selection is to return a randomly selected parent. This parental selection technique has the advantage that it directly promotes reproduction of the fittest population members by biasing each members chances of selection in accordance

with its evaluation[16].

Next crossover may proceed in two steps. First, members of the newly reproduced strings are paired at random. Second, each pair of strings undergoes crossover as follows: an integer position k along the string is selected at random between 1 and the string length less one [1,t-1]. Two new strings are created by swapping all characters between positions k+1 and t inclusively. For example, consider strings A1 and A2:

A1 = 0 1 1 0 | 1 0 1
A2 = 1 1 0 0 | 0 1 0

Suppose in choosing a random number between 1 and 6, we obtain a k=4 (as indicated by the separator symbol |). The resulting crossover yields two new strings:

A1' = 0 1 1 0 0 1 0
A2' = 1 1 0 0 1 0 1

Mutation, the final G.A operator, is the occasional (with small probability) random alteration of the value of a string position. It is an insurance policy against premature loss of important notions.

6. Rule and Message Classification

The first portion of the classifiers and their messages represents the fault tree characteristics, hence it is essential to code characteristics defining the fault tree structure in some form. Several characteristics can be used to represent the significant features of the trees. Figure 6 indicates a simple fault tree structure with common distinguishing elements labelled.

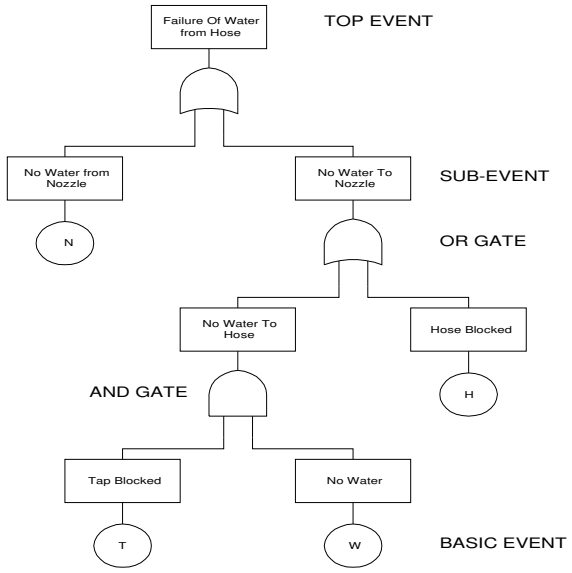

Figure 6: A simple fault tree with labelled elements.

By examining the combinations of these factors a fault tree structure can be described. The list of possible characteristics is endless. Some possibilities include: number of gates; number of events; number of repeated events; number of levels, etc. It is clearly impractical to code all possible characteristics, hence a select few need to be extracted which represent the most significant tree features with relevance to the BDD

construction. In this study eleven characteristics were chosen as an initial starting point to investigate the potential of the classifier system. The characteristics are applied once the fault tree has been represented as an alternating sequence of AND and OR gates.

7. Variable Ordering Schemes

As previously mentioned the ordering placed on the basic events of a fault tree will determine the size of the resulting BDD[17-19], and hence the number of cut sets. It is beneficial to achieve an ordering which is optimal in terms of the resulting size of the BDD.

It is clear that numerous methods of variable ordering are available In previous research[20] 6 different ordering schemes which lead to a reduction on the size of the resulting BDD for different tree structures, have been identified. These are: top-down, left-right approach; modified top-down, left-right approach; depth first approach; modified depth first approach; priority depth first approach; and modified priority depth first approach.

The top-down, left-right approach is the most common, and is produced by listing the variables in a top-down , left-right manner from the original fault tree structure. The depth first approach involves breaking the whole tree structure into smaller trees (subtrees) and looking at the optimal ordering of these subtrees. The depth first ordering scheme gives each subtree a top-down, left-right ordering, working from the first gate inputs of the top event. The priority depth first approach takes the depth first approach one step further and considers subtrees with only basic event inputs first. All the modified versions of each approach consider repeated events of each gate with basic events inputs first. If the gate has more than one repeated event as an input then the most repeated event is placed first, if they occur the same number of times then the events are taken in gate list order to break the tie. The differences in these scheme options are highlighted in Andrews and Bartlett[22].

This short list of schemes have been adopted to investigate the potential of the classifier approach.

8. Breakdown of Input / Output variable coding

Each classifier message and condition comprises of 34 bits, which can be broken down to two main sections, namely: the characteristics coding and the scheme coding. The characteristics coding comprises 33 bits and is broken down to:

4 bits for percentage of and gates;
5 bits for percentage of different events repeated;
5 bits for percentage of total events repeated;
1 bit for top gate type;
2 bits for number of outputs from top gate;
2 bits for number of levels of tree;
3 bits for number of basic events
3 bits for maximum number of gates in any level;
3 bits for number of gates with gate outputs only;
3 bits for number of gates with event outputs only;

and 2 bits for highest multiple of a repeated event.

The scheme coding comprises of 1 bit of either a 0 or a 1, where a 1 represents that scheme as a 'good' schemes and a 0 as a bad scheme. Results to date have shown that testing each scheme separately produces improved accuracy. This involves training a classification scheme for each of the different ordering options. To run the prediction program a tree is tested against each scheme classification system, starting at scheme 1, and producing an output of scheme 1 or not. Trees which produce a negative result are then further processed against the other ordering options until one scheme is chosen. In the event that no numbering option is selected scheme 1 can be used as the default.

9. Training and Test Data

Fault tree structures used in the training and test phases were obtained from industry and also by random production using a computer program. Each tree structure was analyzed for the chosen characteristics, and these characteristics were converted to the appropriate binary representation.

Each tree was analyzed prior to training for the best ordering scheme for the most efficient BDD representation. The best scheme was identified by the minimum number of nodes in the BDD structure.

To evaluate the performance of the learning classifier system a test set of data was produced with different tree structures and known best ordering schemes. The output scheme coding bits for prediction purposes are set to wildcard characters. The performance is evaluated by comparing the number of correct scheme outputs predicted.

10. Results

Results for the number of correct scheme predictions for the test data have been encouraging. For the smaller fault tree structures the results have been more accurate. The significant characteristics that have been considered in this study may for the larger tree structures require more research. More tests need to be undertaken to provide further evaluations.

11. Conclusions

Initial predictions using the classifier approach have indicated promising results in determining the best of alternative ordering schemes to yield an efficient BDD representation, and hence an efficient analysis.

With slight alterations to the set of factors chosen to adequately represent the complexity of the fault tree structure, and also by increasing the size of the training data set, it is predicted that the results for larger trees will be as convincing as that currently shown for the smaller sized fault trees.

References

1. Vesely W.E., “A Time Dependent Methodology for Fault Tree Evaluation”, Nuclear Eng and des., 13, 1970, pp337-360.
2. Bryant R.E., “New Algorithms for Fault Tree Analysis”, Reliability Engineering and System Safety, vol 40,1993, pp203-211.
3. Sinnamon R.M. and Andrews J.D., “ New Approaches to Evaluating Fault Trees”, Proceedings of ESREL 95 conference, June 1995.
4. Sinnamon R.M. and Andrews J.D., “Fault Tree Analysis and Binary Decision Diagrams”, Proceedings RAMS 96, Las Vegas, Jan 96.
5. Rauzy A., “A Brief Introduction to Binary Decision Diagrams”, European Journal of Automation”, Vol 30, No 8, 1996.
6. Sinnamon R.M. and Andrews J.D., “Quantitative Fault Tree Analysis using Binary Decision Diagrams”, European Journal of Automation”, Vol 30, No 8, 1996.
7. Dugan J.B. and Doyle S.A., “Incorporating Imperfect Coverage into Binary Decision Diagrams”, European Journal of Automation”, Vol 30, No 8, 1996.
8. Boussiou M., ”An Ordering Heuristic for building Binary Decision Diagrams from Fault Trees”, Proceedings RAMS 96, Las Vegas, Jan 96.
9. Sinnamon R.M., “Fault Trees and Binary Decision Diagrams”, PhD Thesis, Loughborough University, 1997.
10. Goldberg D.F., “Genetic Algorithms in Optimization and Machine Learning”, Addison Wesely 1997.
11. Quinlan J.R., “Discovering rules from large collections of examples: a case study”, Expert Systems in the Macro Electronic Age (D. Michie Ed), Edinburgh University Press.
12. Bishop C.M. “Neural Networks for Pattern Recognition”, Clarendon, 1995.
13. Eberhart, Simpson and Dobbins, “Computational Intelligence PC Tools”, AP, 1996.
14. Davis,R. and King, J., “An overview of Production Systems.” In E.W. Elcock and D. Michie (Eds.), Machine Intelligence 8 (pp300-332), 1976. New York: Wiley.
15. DeJong, K.A., “Genetic Algorithms: A 10 year perspective,” Proceedings of an International Conference on Genetic Algorithms and Their Applications, 1985, pp169-177.
16. Davis, L., "Handbook of Genetic Algorithms"New York: Van Nostrand Reinhold, 1991.
17. Bryant, R.E., “Graph-Based algorithms for Boolean function,” IEEE Trans. Computers, vol C-35, 1986, No. 8, pp667-691.
18. Akers, S.B., “Binary decision diagrams,” IEEE Trans. Computers, vol C-27, 1978, No. 6, pp509-561.
19. Rauzy, A., “New algorithms for fault tree analysis,” Reliability Engineering and System Safety, vol 40, 1993, pp203-211.
20. Sinnamon, R. and Andrew, J.D., “Improved Efficiency in Qualitative Fault tree analysis”, Advances in Reliability Technology Symposium, Manchester, 1996.
21. Michalewicz, Z., "Genetic Algorithms + Data Structures = Evolution Programs", Springer, 1992.
22. Andrews, J.D., and Bartlett, L.M., "Efficient Basic Event Orderings For Binary Decision Diagrams", Proceedings of the Annual Reliability and Maintainability Symposium, 1998.