Genetic algorithms in optimal safety system design

R L Pattison and **J D Andrews***
Department of Mathematical Sciences, Loughborough University, Leicestershire, UK

Abstract: This paper describes a design optimization scheme for systems that require a high likelihood of functioning on demand. For safety systems whose failure could result in loss of life it is imperative that the best use of the available resources is made and that a system which is optimal and not just adequate is produced.

To demonstrate the practicalities of the method it has been applied to a high-integrity protection system. Analyses of individual system designs are carried out using the latest advances in the fault tree analysis technique utilizing the binary decision diagram approach.

A genetic algorithm (GA) is used to perform the optimization resulting in the final design specification. Techniques are introduced to penalize the fitness of infeasible designs and to incorporate these values into the GA.

The latter part of the paper considers the effect of varying parameters, which affect the action of the GA. A parameter combination is suggested which may achieve the most effective exploration of the search space and, thus, result in the best system design.

Having implemented the GA it became apparent that areas of the algorithm could be improved. The latter part of the paper investigates suggested improvements to particular processes of the scheme.

Keywords: optimization, fault tree analysis, binary decision diagrams, genetic algorithms, design

1 INTRODUCTION

Failure of a safety system for a potentially hazardous industrial system or process may have severe consequences, possibly injuring members of the workforce or public and occasionally resulting in loss of life. It is, therefore, imperative that such systems have a high likelihood of functioning on demand.

One measure of system performance is the probability that the system will fail to operate when necessary. Typically the design of a safety system follows the traditional design process of preliminary design, analysis, appraisal and redesign. If, following analysis, the initial design does not meet some predetermined acceptability target for system unavailability, deficiencies in the design are removed and the analysis and appraisal stages are repeated. Once the predicted system unavailability of a design reaches the acceptable criteria, the design process stops and the system is adopted. For a system whose failure could result in fatality, it could be considered that a merely adequate level for system unavailability is not sufficient. The aim should be to produce the optimal performance attainable within the constraints imposed on resources.

It is highly unlikely that the design parameters can be manually selected such that the optimal system performance can be achieved within the available resources. An approach, by which optimal performance can be obtained using the fault tree analysis method to determine the availability of each system design, was described in a paper in 1994 [**1**]. An alternative methodology was presented in a later paper [**2**], which incorporated the latest advances in the fault tree analysis technique, using binary decision diagrams [**3**–**7**] and utilized a genetic algorithm (GA) [**8**, **9**] to perform the optimization. The research presented in this paper extends the approach in reference [**2**] by investigating the effects of modifying the GA process and the parameter values used. The GA process is thus more accurate and effective.

2 SAFETY DESIGN CONSIDERATIONS

Safety systems are designed to operate when certain conditions occur and act to prevent or mitigate their

The MS was received on 10 June 1998 and was accepted after revision for publication on 16 June 1999.
**Corresponding author: Department of Mathematical Sciences, Loughborough University, Loughborough, Leicestershire LE11 3TU, UK.*

development into a hazardous situation. Where possible, a safety system should not be designed such that single component failure can prevent the system from functioning. To ensure this, redundancy or diversity can be incorporated into the system. Redundancy duplicates elements within a system while diversity involves the addition of a totally different means of achieving the same function.

Component selection is a second design option. Each component selected for the design is chosen from a group of possible alternatives. The design engineer must decide how to trade off the specific characteristics of each component to give the most effective overall system performance.

The time interval between preventative maintenance activities is a further consideration. This is generally assigned on an *ad hoc* basis after the design has been fixed. Significant gains are to be made by considering the maintenance frequency at the design stage.

The choice of design is not, however, unrestricted. Practical considerations place limits on resources, which prevent a completely free choice of system design, rendering some design variations infeasible.

3 THE DESIGN OPTIMIZATION PROBLEM

The objective of the design optimization problem is to minimize system unavailability by manipulating the design variables such that limitations placed on them by constraints are not violated. Commonly with mathematical optimization problems, such as linear programming, dynamic programming and sequential unconstrained optimization [**10**], there will be an explicit objective function which defines how the characteristic to be minimized is related to the variables.

In this problem an explicit objective function cannot be formulated. The system performance is assessed using a fault tree specifically representing the design in question.

The nature of the design variables also adds difficulty to the problem. Design variables that represent the levels of duplication for fully or partially redundant systems and the number of weeks between maintenance activity are all integer. Selecting component types is governed by Boolean variables, i.e. selection or non-selection. A numerical scheme is, therefore, required that produces integer values for these variables since it will not be appropriate to utilize a method where real numbers are rounded to the nearest whole number.

Constraints involved in this problem fall into the category of either explicit or implicit constraints. The cost and maintenance downtime can be represented by an explicit function of the design parameters. However, the number of spurious trips can only be calculated via a full analysis of the system, which will again employ the fault tree analysis technique.

4 GENETIC ALGORITHMS

GAs are a robust class of optimization techniques that use principles mimicking those of natural selection and genetics. Each system design is coded as a string of parameter values. Each string is analogous to a chromosome in nature. The method then works with a population of strings.

The structure of the GA is that each string is assigned a measure of its fitness in the environment. Selection (or reproduction as it is also known) then exploits this fitness information. The greater the fitness value the higher is the string's chance of being selected to enter the next generation.

The whole process is influenced by the action of the genetic operators, typically cross-over and mutation. These perturb the parameter information on each string and allow for greater exploration about the search space.

The basic method of selection allocates offspring to the next generation via a biased roulette wheel. Each string is assigned a certain percentage of the roulette wheel depending on the size of their fitness value in relation to the other strings in the population.

Cross-over involves crossing information between two solution strings, already selected to enter the next generation, from some randomly determined cross-over point. Mutation is the alteration of a specific parameter value on the solution string. Both operators enable exploration of different system designs.

5 SYSTEM ANALYSIS

5.1 Use of fault trees

As no explicit objective function exists, fault trees are used to quantify the system unavailability of each potential design. It is, however, a time-consuming impractical task to construct a fault tree for each design variation.

To resolve this difficulty, house events can be used to enable the construction of a single fault tree capable of representing causes of the system failure mode for each possible system design. House events in the fault tree, which are either TRUE or FALSE, are utilized to switch on or off different branches to model the changes in the causes of failure for each design alternative.

Consider, for example, the choice of a valve type, from the possible alternative valves V1,V2 or V3. The structure of the tree is as shown in Fig. 1. If valve type

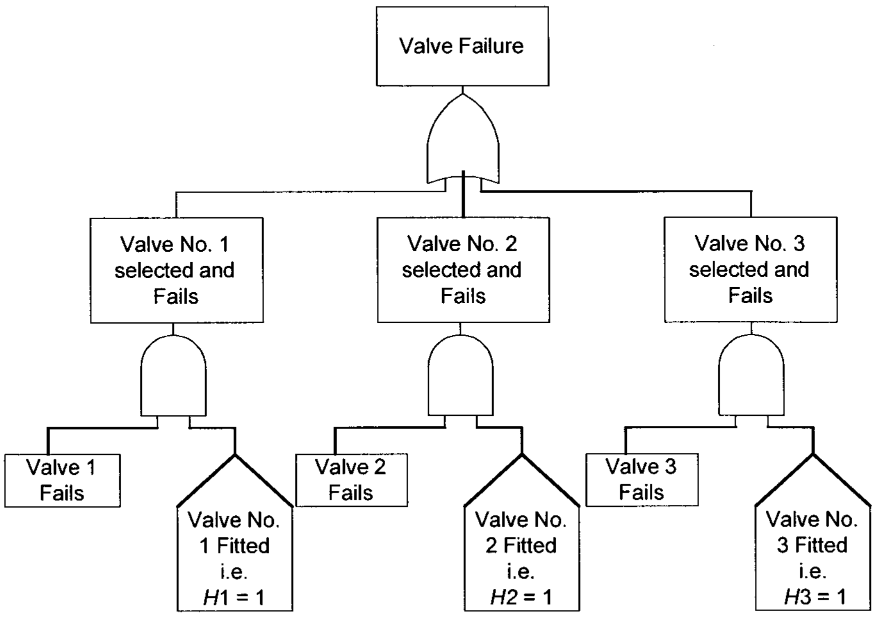
Valve Failure
Valve No. 1 selected and Fails
Valve No. 2 selected and Fails
Valve No. 3 selected and Fails
Valve 1 Fails
Valve No. 1 Fitted i.e. H1 = 1
Valve 2 Fails
Valve No. 2 Fitted i.e. H2 = 1
Valve 3 Fails
Valve No. 3 Fitted i.e. H3 = 1

Fig. 1 Fault tree structure for component selection

1 is selected the house event H1, corresponding to the selection of this valve is set to TRUE. House events H2 and H3, corresponding to the selection of V2 and V3, are conversely set to FALSE. A contribution to the top event arises from the left-most branch only. The two right-most branches are in effect switched off. Levels of redundancy are handled in a similar manner.

The spurious trip frequency for each design is an implicit constraint that requires the use of a fault trees analysis to assess its value. House events are again used to construct a fault tree capable of representing each potential design for this failure mode.

5.2 Use of binary decision diagrams

In order to improve efficiency the binary decision diagram (BDD) method is used to solve the resulting fault tree. A BDD is a directed acyclic graph composed of terminal and non-terminal vertices, which are connected by branches. Terminal vertices have the value 0 or 1 and non-terminal vertices correspond to the basic events of the fault tree. Each vertex has a 0 branch which represents basic event non-occurrence (works) and a 1 branch which represents basic event occurrence (fails). Thus, all paths through the BDD terminate in one of two states, either a 1 state, which corresponds to system failure, or a 0 state, which corresponds to a system success. The BDD represents the same logical function as the fault tree from which it is developed. As an example consider the BDD illustrated in Fig. 2.

Analysis of a BDD has proven to be more efficient than the quantification of the fault tree structure itself. This is because evaluation of the minimal cut sets for use in the quantification is not required. In addition the BDD produces more accurate results.

The fault tree structures for each system failure mode are converted to their equivalent BDDs. A full description of the conversion process can be found in references [**3**] and [**4**]. For the purpose of BDD construction, where house events are encountered in the fault tree, they are treated as basic events. Using this process the fault tree for the component selection design variables, shown in Fig. 1, can be represented by the BDD in Fig. 2.

The quantities q appearing on the 1 and 0 branches developed from each node in Fig. 2 represent the

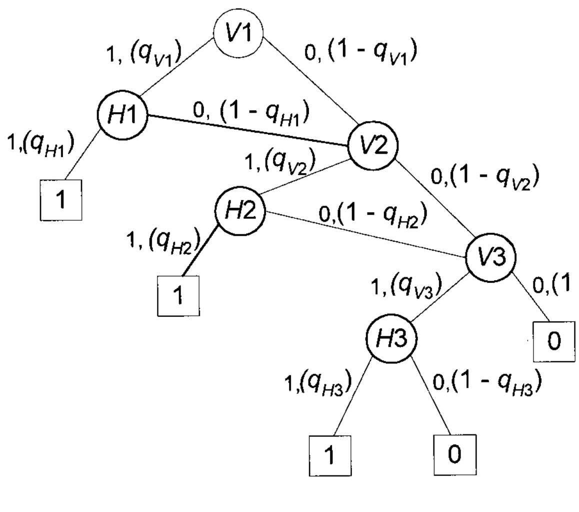

Fig. 2 Component selection BDD

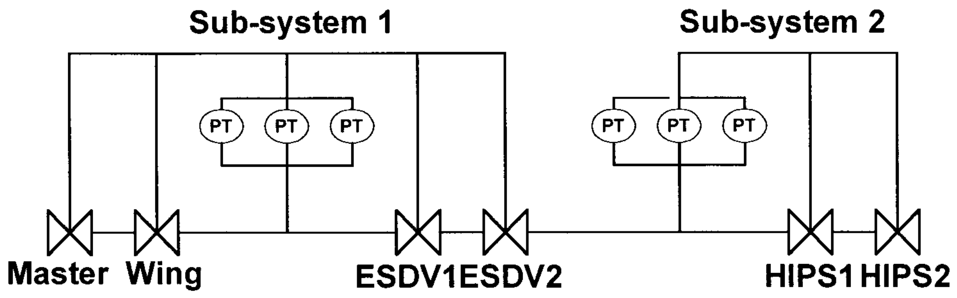

Fig. 3 High-integrity protection system

probability of going down each path. The house events are turned on or off by setting their probability to 1 or 0 respectively. Consider, for example, the design where valve 1 has been selected for the fault tree shown in Fig. 1. This is presented by $H1 = 1$, $H2 = 0$, $H3 = 0$ for the house events and hence the corresponding probabilities $q_{H1} = 1$, $q_{H2} = 0$ and $q_{H3} = 0$ are set on the equivalent BDD. The only path to a terminal 1 node leaves V1 and H1 on their 1 branches which has probability q_{V1} as required.

Quantification of the BDD can be carried out in the GA source code. The probability values assigned to each house event, determined by the design in question, are available within the package and hence are automatically assigned to the BDD. In terms of practicality this is the major advantage of the BDD.

6 EXAMPLE

As an example, the technique has been applied to the simple high-pressure protection system taken from reference [2]. The basic features of the high-pressure protection system are shown in Fig. 2. Its function is to prevent a high-pressure surge passing through the system. In this way protection is provided for processing equipment whose pressure rating would be exceeded. The high pressure originates from a production well of an offshore platform not normally manned and the pieces of equipment to be protected are vessels located downstream on the processing platform.

The first level of protection is to be the emergency shut-down (ESD) subsystem. Pressure in the pipeline is monitored using pressure transmitters (PTs). When the pipeline pressure exceeds the permitted value, then the ESD system acts to close the wing and master valves on the well together with any ESD valves that have been fitted.

To provide an additional level of protection, a second level of redundancy can be incorporated by inclusion of a high-integrity protection system (HIPS) (Fig. 3). This works in a similar manner to the ESD system but is completely independent in operation.

Even with a relatively simple system such as this there are a vast number of options for the designer to consider. In the example it is required to determine values for the design variables given in Table 1. Limitations have been placed on the design as follows:

1. The total system cost must be less than 1000 units. Hardware costs are given in Table 2.
2. The average time each year that the system resides in the down state owing to preventative maintenance must be less than 130 h. The times taken to service each component at each maintenance test are also shown in Table 2.
3. The number of times that a spurious system shutdown occurs would be unacceptable if it was more than once per year.

7 GENETIC ALGORITHM IMPLEMENTATION

SGA_C is a C-language translation and extension of the original Pascal simple genetic algorithm (SGA) code presented by Goldman [8]. This package was used as a framework to build the GA software for the safety protection system optimization. Significant changes and extensions were necessary to carry out the modelling of the HIPS optimization.

Table 1 Design variables required

Designer option	Design variable
How many ESD valves are required (0, 1, 2)?	E
How many HIPS valves are required (0, 1, 2)?	H
How many pressure transmitters are required for each subsystem (0, 1, 2, 3, 4)?	N_1, N_2
How many transmitters are required to trip?	K_1, K_2
Which of two possible ESD/HIPS valves should be selected?	V
Which of two possible pressure transmitters should be selected?	P
Maintenance test interval (MTI) for each subsystem (1 week–2 years)?	θ_1, θ_2

Table 2 Component data

	Dormant failures			
Component	Failure rate	Mean repair time	Cost	Test time
Wing valve	1.14×10^5	36.0	100	12
Master valve	1.14×10^5	36.0	100	12
HIPS1	5.44×10^6	36.0	250	15
HIPS2	1×10^5	36.0	200	10
ESDV1	5.44×10^6	36.0	250	15
ESDV2	1×10^5	36.0	200	10
Solenoid valve	5×10^6	36.0	20	5
Relay contacts	0.23×10^6	36.0	1	2
PT1	1.5×10^6	36.0	20	1
PT2	7×10^6	36.0	10	2
Computer logic	1×10^5	36.0	20	1

7.1 Coding and initializing the population

To specify a safety system design a value is assigned to each of the ten design parameters. These values are then expressed in binary form and placed contiguously to form a string of binary digits. Each parameter must be allowed a particular length of the string, i.e. a particular number of bits, in order to accommodate its largest possible value in binary form. For example θ_1, the parameter governing the MTI for subsystem 1 requires 7 bits to accommodate its maximum time span of 104 weeks. In total each string representing all design variables is 32 bits in length and can be interpreted as a set of concatenated integers coded in binary form.

The restricted range of values assigned to each parameter does not in each case correspond to the representative binary range on the solution string. For this reason a specialized procedure is used to code, to initialize and, in subsequent generations, to check the feasibility of each string [**2**].

7.2 Evaluating string fitness

Constraints are incorporated into the optimization by penalizing the fitness when they are violated by the design. The fitness of each string consists of four parts:

(a) probability of system failure, unavailability, Q_{SYS},
(b) penalty for exceeding the total cost constraint,
(c) penalty for exceeding the total maintenance down-time constraint and
(d) penalty for exceeding the spurious trip constraint.

The result is a sole fitness value for each design, referred to as the penalized system unavailability of the design.

Calculating the penalized system unavailability involves the derivation of the penalty formula for excess cost, maintenance down-time (MDT) and spurious trip occurrences. If a particular design exceeds any of the stated limits, the respective penalty is added to the system unavailability of the design in question. The forms of penalty function used are described in Section 10.2:

$$Q'_{\mathrm{SYS}} = Q_{\mathrm{SYS}} + C_{\mathrm{PEN}} + \mathrm{MDT}_{\mathrm{PEN}} + \mathrm{ST}_{\mathrm{PEN}} \tag{1}$$

where

Q'_{SYS} = penalized probability of system unavailability

Q_{SYS} = unpenalized probability of system unavailability

C_{PEN} = penalty exerted due to excess cost

$\mathrm{MDT}_{\mathrm{PEN}}$ = penalty due to excess MDT

$\mathrm{ST}_{\mathrm{PEN}}$ = penalty due to excess spurious trips

7.3 Reproduction probabilities

The fitness value, or penalized system unavailability, is evaluated for each string. For the purpose of selection in the GA, each string is assigned a reproduction probability which is directly related to its fitness value.

In the safety system optimization problem the smaller the fitness value, the fitter is the string, and hence the greater should be its chance of reproduction. For cases such as these a possible approach is to let the reproduction probability be one minus the fitness value. However, using a string's availability produces all reproduction probabilities of a similar value, thus detracting from the fitness information available to the GA. A more specific method is required which retains the accuracy of each string's fitness value during conversion to its corresponding reproduction probability. Section 10.3 considers this conversion method further and investigates a more effective alternative.

8 RESULTS

The program was used with a population of ten strings. A maximum of 50 generations was allowed together with a mutation rate of 0.01 and cross-over rate of 0.7. In total, therefore, 500 system evaluations were performed in determining the best design. The running time of the program was of the order of minutes. The fittest string from the entire process arose in generation 15. The characteristics of this design are specified in Table 3.

Convergence to a fit design through the GA is not necessarily smooth. A particularly fit string may be produced in an early generation as a result of its random nature; else the structure of the GA may enable uniform convergence to a fit string over later generations.

Table 3 Characteristics of the best design

Subsystem 1	
Number of ESD valves	0
Number of PTs	3
Number of PTs to trip system	2
MTI	23
Subsystem 2	
Number of HIPS valves	2
Number of PTs	3
Number of PTs to trip system	2
MTI	57
Valve type	2
PT type	1
MDT	123 h
Cost	842 units
Spurious trip	0.455
System unavailability	0.001 12

9 THE GA PARAMETERS

The GA requires the following selection parameters to be set:

(a) population size,
(b) cross-over rate,
(c) mutation rate and
(d) number of generations.

The values entered for these parameters have a marked effect on the action of the GA. Using the optimization software previously described, an analysis was carried out to investigate the effect of changing these parameter values. A limited set of values for each parameter was chosen as in Table 4.

In total, 36 runs of the optimization were carried out, thus ensuring that each possible combination of the values above was analysed. The penalized system unavailability of the best overall string per run was then investigated for each parameter set.

To obtain an indication of the effect of setting each parameter to a particular value the best penalized system unavailability obtained for all the other parameter values was summed and averaged. A summary of the best results is given in Tables 5, 6 and 7, for the mutation rate, cross-over rate and population size respectively.

9.1 Discussion of the quantitative results

As might be expected, larger populations lead to a better performance. When the population size doubles from 10 to 20 strings the fitness value improves by 20 per cent. An additional 18 per cent improvement is incurred when the initial population is further increased to 50 individuals.

Table 4 Chosen sets of parameter values

Mutation rate	0.1	0.01	0.001	
Cross-over rate	0.5	0.6	0.7	0.8
Population size	10	20	50	

Table 5 A summary of the quantitative results for the mutation rates M_1, M_2 and M_3

	Mutation rate		
	M_1	M_2	M_3
Sum of fitness values	0.015 031	0.017 101	0.018 992
Average fitness value	1.25×10^{-3}	1.42×10^{-3}	1.58×10^{-3}

Table 6 A summary of the quantitative results for the cross-over rates C_1, C_2, C_3 and C_4

	Crossover rate			
	C_1	C_2	C_3	C_4
Sum of fitness values	0.012 53	0.013 641	0.013	0.011 953
Average fitness value	1.044×10^{-3}	1.37×10^{-3}	1.08×10^{-3}	9.96×10^{-4}

Table 7 A summary of the quantitative results for the population sizes P_1, P_2 and P_3

	Population size		
	P_1	P_2	P_3
Sum of fitness values	0.020 93	0.016 642	0.013 552
Average fitness value	1.74×10^{-3}	1.39×10^{-3}	1.13×10^{-3}

The mutation rate parameter implies that the largest rate, 0.1, leads to the generation of a fitter string. Strings produced in each run of the program with the largest mutation rate were on average 20 per cent fitter than program runs with the lowest rate. The cross-over parameter again produced a slight bias towards the highest value.

These results support a more random search. A population of ten strings may not incorporate enough diversity from the onset and for this reason a high degree of mutation moves to areas in the search space, which would otherwise not be explored. Too high a mutation and cross-over rate, however, indicates that the inherent properties of the GA have been lost and the structured random search has degenerated to a purely unstructured enumerative technique. A balance between the diversity and processing time was made to obtain the best results and it is suggested that a high population size is selected together with a cross-over rate of 0.7 and a mutation rate of 0.01.

10 MODIFYING THE SGA

Having implemented the GA, areas of improvement became apparent. The following sections consider these areas in more detail.

10.1 Utilization of the MDT resource

The MTI parameters for each subsystem directly affect the unavailability of a design. They also contribute to the calculation of a system's MDT, which exerts a penalty on the system unavailability if 130 h is exceeded.

Fully utilizing the 130 h MDT resource and splitting the available effort between the two subsystems to the best advantage will result in the most optimal system unavailability. It was noted that very few designs produced in the GA approached this limit and, hence, are not fully utilizing their available resources.

The MDT is governed by the values assigned to the MTI parameters for subsystems 1 and 2. These MTI parameters span a much greater range of values than the other parameters and hence occupy a greater proportion of the string. The test parameters have a much larger area of the search space to explore. This knowledge supports the evidence from a quantitative analysis, implying a need for greater variation within this area of the string. Three methods to improve the exploration of the range of values allowed by the MTI parameters have been explored, as discussed in Subsections 10.1.1 to 10.1.3.

10.1.1 MDT modification method 1

In method 1 the GA executes in the normal manner and a best overall string is deduced. This best system design is then sent to an additional routine. Within this routine a loop checks the system unavailability for each feasible combination of test intervals for subsystems 1 and 2. Any infeasible combination with an MDT exceeding 130 h is ignored. The combination of MTIs 1 and 2 resulting in the most optimal system performance is retained.

10.1.2 MDT modification method 2

Method 2 introduces greater search to the MTI area of the string, without affecting the rest of the design. This method uses the inversion operator.

Inversion is a reversal of a segment of the string. Consider, for example, the string S, given by

$$S = 1\ 0\ \underline{0\ 0\ 1\ 1\ 1}\ 1\ 1\ 0\ 1\ 1\ 1\ 0$$

If cut points a and b are generated, e.g. $a = 2$ and $b = 7$,

$$S' = 1\ 0\ \underline{1\ 1\ 1\ 0\ 0}\ 1\ 1\ 0\ 1\ 1\ 1\ 0$$

The string S' is the original string with the bits 2 to 7 inverted.

Cross-over and mutation are carried out in the latter part of each string pair only, i.e. between bits 15 and 32. Consequently, following the action of the two genetic operators the MTI area of the string is unaffected. Each new string is then further processed. The string is first decoded to obtain the MTI values for each subsystem. These are stored. The MTI area of each string is then inverted between two randomly generated points a and b, where $0 \leqslant a < b \leqslant 14$. Following inversion the modified MTI values are established and stored. Four potential MTI parameter combinations now exist for the two subsystems. The resulting combinations are considered in one of two ways.

Method 2a evaluates the MDT associated with each MTI combination using the MDT formula. The pair of values that result in the MDT closest to but not greater than 130 h are retained. If all resulting MDT values are greater than 130 h, the test intervals associated with the lowest are kept. The penalty formula is enforced as in the original GA.

Method 2b evaluates the system unavailability in addition to the MDT for each MTI combination. Should any pair of test intervals exceed 130 h MDT the respective penalty is added to the system unavailability for that combination. The test intervals resulting in the lowest penalized system unavailability are retained.

10.1.3 MDT modification method 3

Method 3 is a mixture of methods 1 and 2. As in method 2, cross-over and mutation are carried out in the latter part of each string only. Each string is then processed in the MTI area. Each string enters a loop, as with method 1, that checks the system unavailability of the design for every feasible combination of test parameters in the range. The pair of test parameters resulting in the best system performance for each system is retained. The population of strings, therefore, enters the next generation with each design fully utilizing all the available MDT resource.

10.1.4 Results of the GA using the MDT modification methods

The GA with MDT modification methods 1, 2a and 2b and 3 was tested. Each method improved the utilization of MDT resources, thus improving the performance of the system. Method 1 does not affect the GA process directly. The MDT is considered after as opposed to during the design stage, as is typical of most engineering disciplines. It is common engineering practice to ensure that the resulting design achieves its most optimal performance.

Method 3 ensures that all resources are used at all times, eliminates the need for an MDT penalty formula and results in the best system performance. This method, however, relies on a very large number of system unavailability evaluations and may be considered impractical for larger systems.

Method 2 achieves the desired intention to introduce greater variety into the test interval parameter area of the string. Full use of the MDT resource is not guaranteed but the general case results in convergence towards the limit. The advantage of this method is that it is incorporated within the design process and does not require an excessive number of extra system unavailability evaluations.

Method 3 achieves the best results for the HIPS system. It is probable that method 2b would be more practical for larger safety systems. This explores extra safety system evaluations but not to excess.

10.2 Derivation of the penalty formula

If the performance of a design is significantly improved owing to a comparatively small excess in one or more of the constraints, the design in question deserves further consideration. An excessive abuse of the limits with only a small degree of performance improvement conversely implies that the design be discarded. It is essential that an appropriate penalty be exerted to the system unavailability when constraints are violated which retain these features.

A spurious trip results in loss of production of the industrial system, and hence loss of profit. For this reason a spurious trip is expressed in terms of excess cost and the cost penalty formula used. The penalty formula under consideration is, therefore, that regarding cost.

The cost penalty in the original SGA is derived from the formula

$$\text{Cost penalty} = \left(\frac{\text{excess cost}}{100}\right)^{5/4} \times 0.002 \qquad (2)$$

A base level in system performance is assumed. An unavailability of 0.02 for a system is considered reasonably fit. Should the cost of a design exceed 1000 units, the excess cost percentage should be reflected in the system unavailability as a corresponding percentage improvement about the base level. Thus, a design which costs 1100 units should show an improvement of at least 0.002. This relationship is linear. Small excesses in cost may be tolerated but, as the extra cost becomes much larger, its feasibility significantly decreases [**2**]. For this reason an exponential relationship is preferred, as given in equation (2).

The multiplying factor of 0.002, 10 per cent of the base level performance, is the area of concern. This value is a fixed percentage of a fixed system unavailability. Owing to the set form of the multiplying factor the penalty formula does not take into account the system unavailability of the particular design being penalized.

To illustrate this, consider the following example:

1. Design A costs 1150 units and has an unpenalized system unavailability of 0.015. Apply the cost penalty formula

$$\left(\frac{150}{100}\right)^{5/4} \times 0.002 = 0.0033 \qquad (3)$$

   The penalized fitness value is 0.018, a fitness decrement of approximately 18 per cent.
2. Design B costs 1150 units and has an unpenalized system unavailability of 0.002. Applying the cost penalty formula also gives a penalty of 0.0033. The penalized fitness value is then 0.0053, a fitness decrement of approximately 62 per cent.

The comparative penalty for the fitter string is much greater. The penalty should take the fitness value of the system to be penalized into consideration.

An alternative penalty formula is introduced which takes into account both the cost violation and the system unavailability of the design being penalized. This is achieved using a multiplying factor which, rather than being fixed, varies according to the system unavailability of the design.

Consider a particular design with cost C. The cost C exceeds 1000 units. The percentage excess of the system's cost is calculated, X^C say. The multiplying factor is derived by calculating X^C per cent of the system unavailability of the design under consideration. In the designs A and B previously stated, both designs exceed 1000 units by 15 per cent, i.e. $X^C = 15$ per cent. The system unavailability of design A is 0.015, of which 15 per cent is 0.0025. The multiplying factor to be used in the penalty formula for this design is, therefore, 0.0025. Hence, apply the penalty formula

$$\text{Cost penalty} = \left(\frac{150}{100}\right)^{5/4} \times 0.0025 = 0.0041 \qquad (4)$$

The penalized fitness value is 0.0191, a 22 per cent decrement. The system unavailability of design B is 0.002, of which 15 per cent is 0.0003. Hence, apply the cost penalty formula

$$\text{Cost penalty} = \left(\frac{150}{100}\right)^{5/4} \times 0.0003 = 0.000\,49 \qquad (5)$$

The penalized fitness value is 0.0025, a 20 per cent decrement.

Altering the cost penalty formula enables a more detailed exploration around the border of the search space. The lack of final system designs from the original GA that portray a slight excess in either the cost or trip constraints implies that the penalty exerted is too great.

10.3 Converting the fitness value to a roulette wheel percentage

Each design receives a measure of its fitness. This is the design string's penalized system unavailability. This value is not in an appropriate form to be directly used in the selection process of the GA since, the smaller the fitness, the better is the design.

A specialized conversion method is required. It is imperative that the conversion method gives rise to roulette wheel percentages in accordance with the fitness value of each string. A system whose performance is twice as good as another should have twice the percentage allocation.

The original conversion method allocates each string to one of three categories according to its fitness value. The ranges of values covered by each category are given in Table 8.

Each string in category 1 is automatically given zero per cent. This category consists of poor system designs. If they do not feature on the roulette wheel, they will be eliminated from the succeeding generation.

Category 2 contains relatively unfit designs. It is, however, important to retain a little diversity in the population. Each string is allocated some portion of a total of 5 per cent of the roulette wheel.

The strings that fall into category 3 are of ultimate interest. To enhance their fitness values each string is subtracted from the upper category limit, 0.1. A particular amount of the remaining 95 per cent of the roulette wheel is then allocated to each string, depending on how much their fitness value exceeds the 0.1 limit.

Problems occur when a very high, or a very low, proportion of strings fall into a particular category. The percentage allocated to each category is fixed and, therefore, independent of the number of strings that it contains.

An alternative method is required which is able to cope with very diverse populations and simultaneously to show sensitivity to a highly fit set of strings. Initially nine categories are depicted which cover the area of the fitness domain of importance, i.e. below 0.2, and each category is assigned a particular weight, as shown in Table 9. As the category becomes fitter, its weight increases in size.

The fundamental steps of the modified method are then as follows:

1. Firstly each category is designated a particular percentage of the roulette wheel depending on:

Table 8 Range of values of categories of fitness values

Lower limit		Category		Upper limit
0	$\leqslant$	3	$\leqslant$	0.1
0.1	$<$	2	$\leqslant$	0.2
0.2	$<$	1	$\leqslant$	1.0

Table 9 Nine categories plus weights

Fitness domain					
Upper limit		Category		Lower limit	Weight
0.2	$>$	1	$\geqslant$	0.1	1^2
0.1	$>$	2	$\geqslant$	0.05	2^2
0.05	$>$	3	$\geqslant$	0.01	3^2
0.01	$>$	4	$\geqslant$	0.005	4^2
0.005	$>$	5	$\geqslant$	0.004	5^2
0.004	$>$	6	$\geqslant$	0.003	6^2
0.003	$>$	7	$\geqslant$	0.002	7^2
0.002	$>$	8	$\geqslant$	0.001	8^2
0.001	$>$	9	$\geqslant$	0	9^2

   (a) the number of strings, n, of the total population N in the category and
   (b) the weight assigned to the category.
2. The percentage allocated to each category is then distributed appropriately between the strings within. The method used must ensure that a system in a fitter category is given a greater percentage than a poorer design in a less fit category.

The following sections describe steps 1 and 2 in greater detail.

10.3.1 Establish the percentage allocation for each category

Each string in the population is considered in turn. It is first established into which category the string falls. As in the original method the fitness value is enhanced by subtracting its value from the upper limit of the least-fit category, namely 0.2. The enhanced fitness value is then multiplied by the weight associated with the category into which the string falls. The 'weighted' values of the strings in each separate category are summed. The result is that each category is designated a particular value directly related to the number of strings that it contains and its weight. Using these designated values the relative percentage of each category is calculated.

10.3.2 Distribute the percentage of each category between the strings that it contains

Having established the percentage allocated to each category, the average percentage allocated to each string within each category can be evaluated. As the categories become fitter, the average percentage allocated to each string should increase.

The important aspect here is to ensure that strings in fitter categories are given a larger percentage than those in less-fit categories. For each category, the average percentage allocated to each string in the previous non-empty category is ascertained. As regards category 9, this is always zero.

The total percentage appointed to each category is split into two parts: a 'used' and an 'excess' percentage. Each string in a category, X say, must be given a percentage at least that of the average allocated to each string in the most previous non-empty category, as previously determined. If category X contains n from N strings in the population, the 'used' percentage is

$$\text{cat}_X\text{used} = n \times \text{prevav}_X$$

where

cat_Xused = 'used percentage of category X

prevav_X = average percentage allocated to each string in the most previous non-empty category

The 'excess' percentage for a category is, therefore, its total minus its 'used' percentage. Each string in a category is first given the average percentage allocation of each string in the most previous non-empty category. An additional portion of the categories' 'excess' percentage is then given to each string.

10.3.3 Results comparing both conversion methods

The GA was tested with ten program runs, over 50 generations using the modified conversion method. The overall system performance was improved using the modified method. Most importantly the new approach has a better ability to converge on a very fit design.

10.3.4 Discussion of the conversion method results

The original method is not able to cope adequately with highly fit populations of system designs. The fitness values of the strings are not accurately represented, and hence the information used by the GA is not in accordance with the actual fitness information of the population.

The modified method is able to differentiate between the strings in the fitter population, while simultaneously retaining the ability to handle a varied population. Essential information is not lost in the conversion process. The best design obtained using the modified conversion method had the characteristics shown in Table 10.

11 SUMMARY AND DISCUSSION

The modified cost penalty and the modified conversion method together with MDT modification method 3 (described in Section 10) is established as the preferred GA method. It ensures that all MDT resources are used at all times and are distributed between both subsystems to the best advantage. As such, potentially fit designs are not eliminated owing to inferior MTI values. This method, however, requires a large number of system unavailability evaluations and may be considered impractical for larger systems (in which case method 2a, plus application of method 1 to the resulting design, should be used). Analysing the system unavailability of only those test interval combinations within a specific range of the MDT limit effectively reduces computer effort. Adapting the exhaustive approach to calculate the MDT values only, thus incurring no extra demand on system unavailability evaluations, is an alternative method with much less demand on processing time. This distribution of maintenance values between each subsystem may not, in this case, be optimal.

The modified GA demonstrates the ability to find and explore the fittest areas of the search space. This is achieved via full usage of available MDT resources and a thorough exploration of the boundary of the domain. It is able to differentiate between highly fit strings as the algorithm progresses and retention of the best design over latter generations is achieved.

The modified GA has been applied to a firewater deluge system (FDS) on an offshore platform. The FDS is a larger, more complex system than the HIPS, which has in excess of 4.4×10^{10} design variations. The fault tree used to quantify this system has more than 450 gates and 420 basic events and requires conversion to 17 BDDs.

Analysis of the FDS was carried out using the modified GA. The average fitness of the initial population was 0.207. The population average fitness reduced to 0.0157 when convergence was achieved. The most optimal design arose in generation 48 with a fitness value of 0.013 26. The running time of the program was of the order of 1 h on a workstation.

Table 10 Characteristics of the best design obtained using the modified conversion method

Subsystem 1	
Number of ESD valves	0
Number of PTs	4
Number of PTs to trip system	1
MTI	38
Subsystem 2	
Number of HIPS valves	2
Number of PTs	4
Number of PTs to trip system	2
MTI	30
Valve type	2
PT type	1
MDT	120 h
Cost	882 units
Spurious trip	0.978
System unavailability	0.000 94

Design projects require many months to progress from the initial conceptual design through to detailed design. Whether the use of computer time to aid this process takes a matter of minutes or a number of days is, therefore, not the issue and such differences are considered unimportant.

12 CONCLUSIONS

1. An automated robust design optimization process has been developed in which the adequacy of the system performance is assessed using fault tree analysis. A single fault tree represents the causes of failure for each possible design alternative of a safety system. Implicit and explicit constraint forms have been incorporated to place limits on the design specification. The solution of this type of problem is made possible by use of the BDDs to solve the fault tree.
2. The GA has been shown to produce good results for system design optimization. It is a robust method, which can potentially be applied to a wide range of systems.
3. The practicality of the overall design optimization process has been demonstrated by successful applications to a high-pressure protection system.
4. Investigation of the results of the relatively small problem of the HIPS has highlighted possible difficulties in the GA's exploration of the design space. Problems can occur with the MTI variables and their relationship with the constraint limiting the MDT of the system. This parameter also dominated, i.e. required a larger portion of the design string than, the other variables which can result in an inadequate exploration of the parameter's search space.
5. The simple GA has been modified to overcome the difficulties in its application to the high-pressure protection system to give a very effective design tool.

REFERENCES

1 **Andrews, J. D.** Optimal safety system design using fault tree analysis. *Proc. Instn Mech. Engrs*, *Part E*, *Journal of Process Mechanical Engineering*, 1994, **208**(E2), 123–131.

2 **Andrews, J. D.** and **Pattison, R. L.** Optimal safety-system performance. In Proceedings of the 1997 Reliability and Maintainability Symposium, Philadelphia, Pennsylvania, January 1997, pp. 76–83.

3 **Rauzy, A.** New algorithms for fault tree analysis. *Reliability Engng Safety System*, 1993, **40**, 203–211.

4 **Sinnamon, R. M.** and **Andrews, J. D.** Fault tree analysis and binary decision diagrams. In Proceedings of 1996 Reliability and Maintainability Symposium, Las Vegas, California, January 1996, pp. 215–222.

5 **Sinnamon, R. M.** and **Andrews, J. D.** New approaches to evaluating fault trees. In Proceedings of European Safety and Reliability Conference (ESREL 95), June 1995, pp. 241–254.

6 **Sinnamon, R. M.** and **Andrews, J. D.** Improved efficiency in quantitative fault tree analysis. *Qual. Reliability Engng Int.*, April 1997, **13**, 293–298.

7 **Sinnamon, R. M.** and **Andrews, J. D.** Improved accuracy in quantitative fault tree analysis. *Qual. Reliability Engng Int.*, April 1997, **13**, 285–292.

8 **Goldberg, D. F.** *Genetic Algorithms in Search Optimization and Machine Learning*, 1989 (Addison-Wesley, Reading, Massachusetts).

9 **Painton, L.** and **Campbell, J.** Genetic algorithms in optimization of system reliability. *IEEE Trans. Reliability*, June 1995, **44**(2), 172–178.

10 **Tillman, F.A., Hwang, C.** and **Kuo, W.** *Optimization of Systems Reliability*, 1980 (Marcel Dekker, New York).