System failure minimisation using automated design optimisation

System Failure Minimisation Using Automated Design Optimisation

Dovile Astapenko and Lisa M. Bartlett
Aeronautical and Automotive Engineering , Loughborough University, Loughborough, UK

Abstract

Safety systems are designed to prevent the occurrence and future development of hazardous situations. Consequences of the failure of a safety system varies from minor inconvenience and cost to personal injury, significant economic loss and death. The operation of a safety system can be improved by either introducing better performing components or by increasing the number of redundant components. At the same time, such design alterations can influence how available resources are utilized. The focus of this paper is to introduce a generic optimisation method for constructing an optimal design case for any safety system, with the aim of maximising its likelihood of functioning on demand and at the same time ensuring optimal usage of available resources. The analysed optimisation problem is represented as the constrained single objective problem. The implemented optimisation method employs Fault Tree Analysis (FTA) to represent system failure causes and Binary Decision Diagrams (BDDs) to quantify its failure probability. A Single Objective Genetic Algorithm (SOGA) has been chosen as the optimisation technique. The methodology is illustrated with the optimisation of a High Integrity Protection System (HIPS) design. The constraints imposed are on system dormant failure probability, cost and maintenance down time. Results of the application, with the generic implications of the analysis, are discussed.

1. Introduction

Safety systems are designed to prevent the occurrence of certain conditions and their future development into a hazardous situation. Consequences of the failure of a safety system varies from minor inconvenience and cost to personal injury, significant economic loss and death. Safety systems have a specific functioning principle, i.e. such systems work on demand. A high likelihood of functioning on demand for a safety system can be ensured by altering its design. In such case redundancy techniques can be introduced or certain components maybe replaced with that of better reliability characteristics [1]. However design alterations and therefore level of system reliability improvement is usually subject to a number of design requirements such as cost, weight [2] or maintenance downtime and other requirements of the system performance. Thus the problem is to construct such system design that would improve system reliability within constraints imposed on certain design characteristics. However in many cases to achieve this by manually enumerating all possible design options and analysing system performance is impossible.

In this paper the general system design optimisation algorithm (GSDOA) to determine an optimal design configuration for a safety system is introduced. The approach combines an optimisation technique with both qualitative and quantitative system analysis methods. Fault tree analysis provides schematic description of possible combinations of system conditions that could lead to its failure [3]. Thus it was employed to represent a particular design configuration through listing system failure causes for this system design. A binary decision diagram (BDD) is a directed acyclic graph representing a Boolean Function. The quantitative analysis of fault trees can be performed by transformation into BDDs [4]. The BDD based approach is considered to be a computationally more efficient method. Thus in the design optimisation algorithm the BDD method has been implemented to quantify system failure. Genetic Algorithm (GA) was chosen as the optimisation technique to perform the optimisation part of the approach. GA is a stochastic global search method which is based on the mechanics of natural genetic variation and natural selection [5]. It is a population based technique where encoded parameter sets are used rather than the parameter sets themselves. GA uses the fitness function itself, does not require derivative or other auxiliary quantities and can easily handle constrained optimisation problems [6].

The approach is designed to be applicable to any safety system. In this paper as an application example the summary of work carried out on the High Integrity Protection System (HIPS) is provided. The overall aim of the optimisation is to achieve the HIPS system design which is optimal in three criteria: unavailability, cost, and maintenance down time.

2. System Analysis

One of the main objectives when designing any safety system is to ensure its high likelihood of availability on demand. Required system availability can be achieved be introducing certain system design structure. This section discusses cases when initial system design is defined and a number of alterations are introduced in order to improve system availability. Quantitative system analysis due to design changes is also discussed. System design alterations and analysis is a preparative part of the system design optimisation process.

2.1 Design Considerations

A number of alterations regarding the structure and operation of a safety system influence system availability. Design alterations that can affect system reliability are redundancies introduced at component or subsystem levels and also different component-type selections. System operation can be influenced by time taken to maintain the system and how often maintenance actions are taken. However when making alteration a number of system characteristics have to be taken into account, such as system cost, weight and etc.

In order to prevent system failure caused by failure of one component redundancy can be incorporated into the system structure. Introducing redundancy system components are duplicated. As a result contribution of failure of one-type component towards failure of the system occurs if and only if all redundant components fail. It is also possible to introduce so called *k-out-of-n* redundancy. Here *n* defines number of redundant components and *k* is a number of working components that is needed for successful system operation. In this case a system will be subject to failure if $n-k+1 \leq n$ components fail. If *k* is equal to *1* then it is equivalent to a simple redundancy case. Both redundancy types can be implemented at component or sub-system level. Thus the problem is what redundancy level, i.e. number of redundant components, to choose in order to improve system reliability up to a certain level since available resources are usually very limited.

It is also possible to improve system reliability by replacing a component with another component selected from a group of possible alternatives. Each possible candidate can have different characteristics such as failure rate, cost, weight or time taken to perform its maintenance. A problem in making a decision about candidate suitability appears when the choice between different characteristics of the components needs to be made. For example, a choice needs to be made between a more reliable component which is expensive and a less reliable component but which is twice as cheap as the previous one.

After system design has been finalised system maintenance can also be considered as an option contributing towards system reliability improvement [7]. The time taken to maintain the system can be altered to meat safety requirements. When a system design is defined system maintenance frequency depends only on time intervals between preventive maintenance activities of the components. Thus time intervals between preventive maintenance of components can also be considered during system design alteration process.

2.2 Analysis of System Designs

A fault tree provides a schematic description of possible causes of a specific system failure. Each event of the fault tree defines a dynamic change of state of a system element. Thus if a system design is altered and new components are introduced resulting fault tree for a new design system will also include new events representing failures of the new system components. This justifies the use of fault tree analysis to identify different system designs and find system reliability for the specified designs.

However, on the other hand, it is time-consuming to construct and then analyse a fault tree for each possible design case. The problem can be resolved by using a fault tree representing all possible design alterations. First this idea was suggested by Andrews & Pattison [8]. They analysed a specific system case which was discussed in more detail in [9]. In this

paper the problem of constructing a fault tree comprising all possible designs is extended. Rules which define changes in a fault tree according to alterations of system design introduced are presented. Employing these rules it is possible to represent all possible design variations in one fault tree for any safety system analysed.

2.2.1 Fault Tree Modification Patterns

Consider different design alteration cases, for example, a replacement of a chosen component with a certain number of redundant components. In this case the number of new components can vary from 1 to *n*. Additionally the *k*-out-of-*n* redundancy can be chosen were *k* is not defined and can be equal to any number form interval (1, *n*). A different component type selection and its replacement with another chosen initial design component also results in a system design change. Letter *t* identifies a number of possible component types. However It is also possible to introduce component type selection option for new redundant components. Thus the choices of system design alterations associated with a replacement of one component can be defined with maximum values of three variables: *n*, *k* and *t*. These variables are called design variables since they define changes being introduced to the initial system design.

A fault tree representing all possible design variations includes house events. The house events are employed to switch on or off different branches of the fault tree to model causes of system failure for each design alternative. All house events in the fault tree are linked up in groups. Each group is associated with either introduced redundancy or component type selection. Links between house events in one group are set to alter a part of the fault tree by switching certain branches on and off so that only one possible design alternative is modelled.

House events together with structural fault tree changes associated with system design alterations are incorporated in the initial design fault tree using fault tree modification patterns. A fault tree modification pattern (FTMP) defines a fault tree structure representing all possible design variations after a replacement of one component. The fault tree structure incorporates new basic events associated with a group of together linked house events. Groups of house events corresponding to different FTMPs are independent from each other.

Each modification pattern is defined by three parameters: the maximum possible number of redundant components (*n*), redundancy type (*k*) and the number of possible different component types (*t*). There are five FTMP patterns corresponding to all possible component replacement cases. Each replacement is possible at both component and subsystem level. A FTMP is identified according to values of variables *n*, *k* and *t*:

1. Pattern 1*: $n > 1$, $k = 1$, $t = 1$*;
2. Pattern 2: $n > 1$, $k \leq n$ ($k \neq 1$), $t = 1$;

3. Pattern 3: *n > 1, k = 1, t > 1*;
4. Pattern 4: *n > 1, k ≤ n (k≠1), t > 1;*
5. Pattern 5: *n = 1, k = 1, t > 1;*

Consider a system with a fault tree in Figure 1. As an example Pattern 1 is used which represents the case when one system element is replaced with several redundant elements. The pump is chosen to be replaced and up to *n* redundant pumps can be used. The resulting fault tree is shown in Figure 2 a). If a specific case is analysed and, for example, when *n* = 3 the fault tree for this system is given in Figure 2 b).Here the fault tree contains a group of 3 house events. A number of pumps in the system is defined by turning on a relevant branch, i.e. by setting a relevant house event to TRUE while the rest of the house events are set to FALSE. For instance, if the value of the house event PFT2 is set to TRUE and house events PFT1 and PFT3 are set to FALSE then the fault tree represents as system with two redundant pumps fitted in.

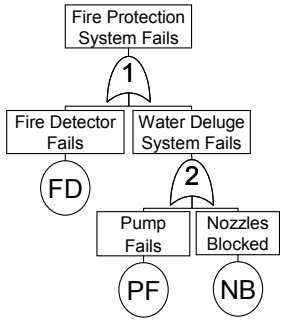

Figure 1. Initial Design System Fault Tree

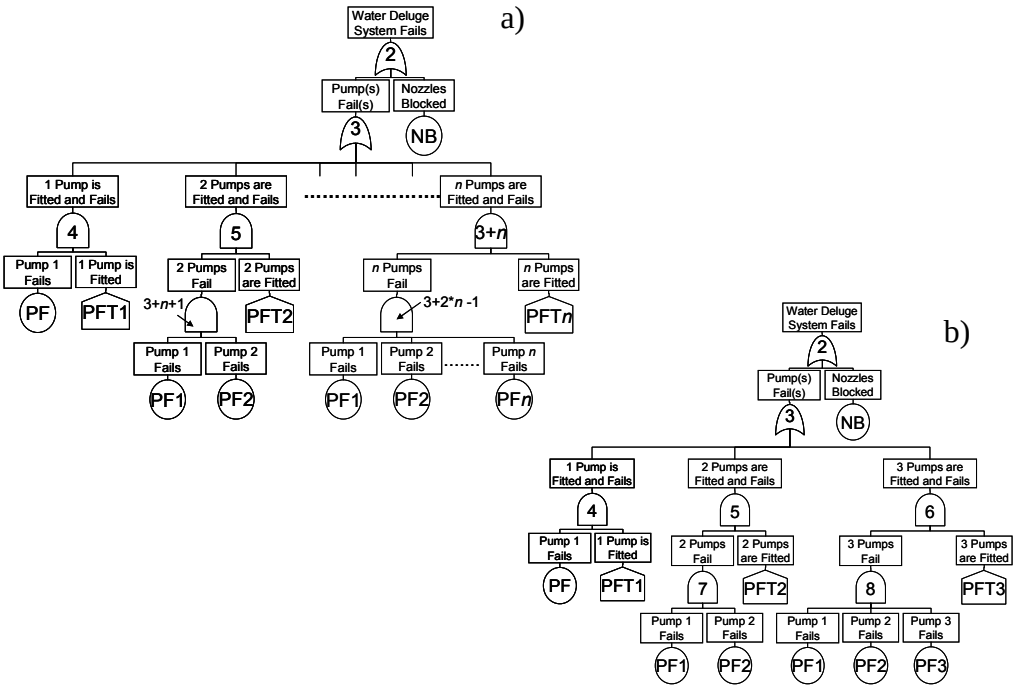

Figure 2. Altered Design System Fault Tree

In a similar manner other modification patterns are implemented and then certain house events can be set to True of False to model different design cases.

To summarise the fault tree representing all possible system designs is constructed from the initial design fault tree and altered according to given design variables using appropriate FTMPs. It means any system can be chosen and the design alterations specific only to that system can be implemented. Therefore it can be stated that implementation of FTMPs is the first step towards constructing a general optimisation algorithm applicable to any safety system.

2.2.2 System Unavailability Evaluation

As sated at the beginning of this section, to analyse different design cases one fault tree representing all potential designs can be built. However, when performing quantitative system analysis for different designs each design needs to be analysed individually. Fault tree trimming operation is introduced for this purpose. It is implemented in two steps. First a particular design case is modelled by setting house events to either TRUE or FALSE sate. Then the fault tree undergoes trimming operation. Branches with house events set to a FALSE state are cut off resulting in a fault tree for the analysed system design case.

Although trimming may slow down the optimisation process, it allows greatly reducing the size of the tree being analysed which results in a much faster calculation. It is especially useful when a larger number of design variables is used and / or their maximum possible values are large and when alterations are made at sub-system level.

At the following stage to find system unavailability for each generated design its fault tree is converted to BDD. Thus a BDD based method was chosen for quantitative analysis instead of using a fault tree approach. A BDD is a directed acyclic graph that represents a Boolean function. All paths through the BDD star at the root vertex and terminate in one of two states: a 1 state or a 0 sate. When applied in reliability analysis the 1 state corresponds to system failure, i.e. occurrence of the top event. Therefore paths terminating in a 1 sate form a cut set of the fault tree. Conversion of the fault tree to BDD format enables to find exact system unavailability in a computationally more efficient way. The BDD approach detailed by R. Remenyte-Prescott [10] was employed in the algorithm.

3. System Design Optimisation Algorithm

3.1 Optimisation problem introduction

A system design optimisation problem is analysed as a general single objective minimisation problem. The problem is stated as a minimisation of a system failure probability:

$$\min Q(\mathbf{X})_{\text{sys}} \tag{1}$$

where **X** is *m*–dimensional vector **X** = {x_1, $x2$, …, x_n}. An element x_i is a failure probability value of a basic event *i* i.e. system component *i*. Vector dimension *m* is equal to a number of basic events in the fault tree or system components subject to failure. Thus if the number of system components varies for different design cases contents of **X** is adjusted to the changes. It follows that the optimisation objective to minimise system unreliability is equivalent to the objective to find such vector **X** that corresponds to minimum system unavailability.

In the developed approach it is possible to set a number of limitations to system characteristics. If a system design *i* meets the limitations it is considered as a feasible system design. Constraints for system cost, weight and volume can be used. To use the resources efficiently it may be useful to have minimum and maximum constraints (Eq. 2). If only maximum limit values are needed then the minimum constraint values become equal to zero.

$$\begin{aligned} &Cost_{\min} < Cost^{i}_{sys} < Cost_{\max}, \\ &Weight_{\min} < Weight^{i}_{sys} < Weight_{\max}, \\ &Volume_{\min} < Volume^{i}_{sys} < Volume_{\max}, \end{aligned} \tag{2}$$

Since safety system works on demand time taken to maintain the system influence its availability. Therefore possibility to define limits for minimum and/ or maximum maintenance down time is also implemented in the algorithm:

$$MDT_{\min} < MDT^{i}_{sys} < MDT_{\max} \tag{3}$$

3.2 Optimisation Algorithm Structure

The structure of the optimisation algorithm is shown in Figure 3. The algorithm contains three main stages. At first all possible system designs are introduced using fault tree modification patterns chosen according to a given list of design variables. The resulting fault tree then can used for quantitative system analysis. Quantitative analysis is performed to evaluate system unavailability for different design cases. On the other hand individual system designs are generated using an optimisation technique. Therefore these two processes are performed simultaneously. It means each time a set of certain design variable values is generated it is passed to evaluated reliability of the corresponding design. The obtained result is passed back to the optimisation part. The obtained information is then applied to generate another set of design variable values and the process is repeated. At the end of the optimisation process an optimal system design is generated which represents a system design with minimal failure probability.

Three major techniques are employed. System designs are presented using fault trees analysis and BDD technique is employed for system quantitative analysis. Search for the optimal system design is performed using single objective Genetic Algorithm (SOGA).

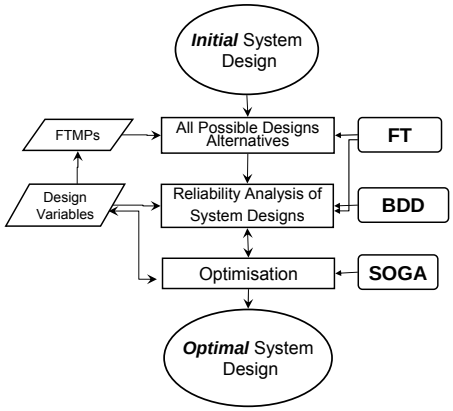

Figure 3. Structure of GSDOA

3.3 Genetic Algorithm

The implemented GA is summarised by the flowchart in Figure 4. Each stage of the algorithm is discussed in detail.

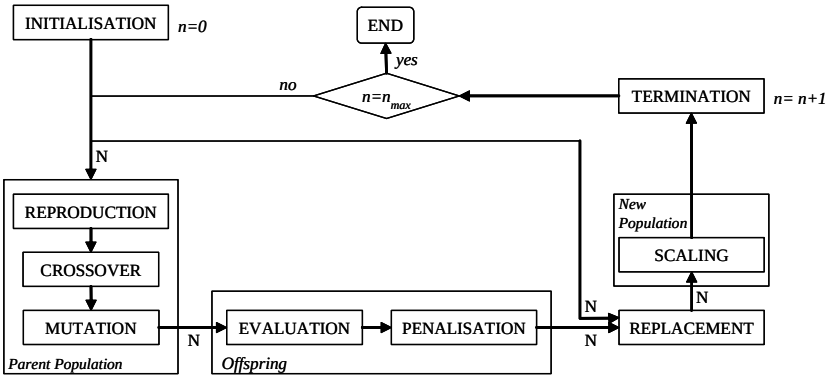

Figure 4. Structure of SOGA

3.3.1 String Encoding and Initialisation

When developing a GA a problem specific representation is needed to describe each chromosome of the population. In the introduced approach a set of design variables associated with a problem forms a chromosome. Binary numbers are used for encoding of each variable. In each string, each design variable is allocated a particular number of bits required to code a possible maximum value of the variable. It ensures that the size of a chromosome is sufficient to store any value of design variables and its size remains constant.

In the proposed algorithms a feasible initial population is generated. It is implemented in the following way. Each string is generated by random

sampling. For each decision it is checked if the amount of resources required for the generated design system does not overcome predefined limits. If all constraints for resources are satisfied the string representing the system design enters the initial population. The process is repeated until *N* chromosomes enter the initial population. Here *N* denotes the size of the population.

3.3.2 Reproduction, Crossover and Mutation

Three operators are used to create a new offspring population. At first the reproduction operator is implemented employing a biased roulette wheel. As a result *N/2* couples of parent strings enter into a mating pool. Strings of each couple are crossed over employing a one-point crossover operator. As a result two new strings are created. During the crossover process, a bit-by-bit mutation is also carried out.

3.3.3 Penalisation

A new offspring population is created as a result of reproduction, crossover and mutation. At this stage the new strings are decoded and their corresponding objective function values are evaluated. Since the algorithm is developed to solve a constrained optimisation problem a penalty application was chosen as an approach to deal with possible violations of constraints. A penalty function proposed by Coit et. al. [11] was employed in the algorithm:

$$F_p(\mathbf{x}) = (F_{all} - F_{feas}) \sum_{i=1}^{nc} \left(\frac{d_i(\mathbf{x}, B)}{NFT_i} \right)^{\kappa_i} \tag{4}$$

Here, F_{all} is the best unpenalised value of the objective function yet found, F_{feas} is the best feasible value of the objective function yet found, NFT_i denotes the near-feasibility threshold that corresponds to a given constraint i, $d_i(\mathbf{x},B)$ is the magnitude of the violation of a given constraint i for solution $\mathbf{x}$, κ_i denotes a user-specified severity parameter and nc is the total number of constraints set for the problem.

3.3.4 Replacement and Scaling

Replacement was implemented employing an algorithm described by Chambers [12]. The idea of this algorithm is to replace a parent population with an offspring population. If the best parent chromosome is fitter than the best offspring chromosome than it replaces the worst offspring chromosome. Replacement is then followed by fitness scaling procedure.

A linear fitness scaling [5] was introduced in order to improve the performance of the algorithm. Research shows that it is especially valuable when small population GA are employed. Parameters used in the linear scaling procedure are problem-independent. They depend on a population life and are found for a population in each generation.

4. Application Example and Results

The GSDOA has been applied to a simple High Integrity Protection System (HIPS) on an offshore oil production well. The main function of the HIPS is to prevent a high-pressure surge passing through it. Protection is provided for processing equipment whose pressure rating could be exceeded. Figure 2 represents the main features of the HIPS.

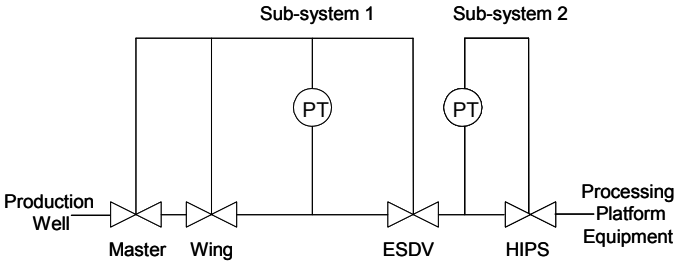

Figure 5. Structure of High Integrity Protection System

HIPS is divided into two separate subsystems. Sub-system 1 is the Emergency Shutdown (ESD) sub-system. This is the first level of protection of the HIPS. Sub-system 2 provides an additional level of protection. Inclusion of the high-integrity protection system incorporates this second level of redundancy.

The data for the optimisation problem of the HIPS included a fault tree for its initial design and data for each component such as failure rate, mean repair time, cost and testing time to perform maintenance. Time intervals between maintenance for two sub-systems were not defined thus they were included in the list of design variables. Ten main design variables were introduced (Table 1). Two limiting factors were considered. HIPS cost could not exceed 1000 units and maintenance down-time had to be less than 130 hours a year.

The initial HIPS fault tree was altered twice using Pattern 3 (n=2, k=1, t=2) regarding system changes associated with replacement of one ESD and one HIPS valves. Pattern 2 (n=4, k=4, t=2) was also applied twice due to introduced replacement choices for pressure transmitters in both sub-systems.

GAs are guided search methods and the best values for the GA parameters are case dependant. Therefore the choice of GA parameter values was based a trial-and-error approach. Different values of population size, crossover rate and mutation rate were chosen. Three population sizes were analysed: 50, 30 and 10 chromosomes. Population size had the biggest influence on optimisation duration. Mutation rates were chosen equal to 0.001, 0.005 and 0.01 and crossover rate values were equal to 0.75, 0.8 and 0.95. The best result, i.e. the smallest number of generations required to find the minimal failure probability value was equal to 73 and was obtained when using a 50 chromosome population, a crossover rate equal to 0.75 and a probability of mutation equal to 0.01.

In 1999 R. Partison [9] also implemented the simple GA specially designed for the HIPS optimization. The comparison of the best designs obtained using both approaches is in Table 2.

Associated System Component	Design Variable	Design Variable Description	Design Variable Value
ESD Valve	E	Number of ESD valves fitted	1, 2
	V_1	Valve type	Type 1 or type 2
HIPS Valve	H	Number of HIPS valves fitted	1,2
	V_2	Valve type	Type 1 or type 2
Pressure Transmitter 1	N_1	Number of pressure transmitters fitted in subsystem 1	1 - 4
	K_1	Number of pressure transmitters required to activate for subsystem 1	1 - N_1
	P_1	Pressure transmitter type	Type 1 or type 2
Pressure Transmitter 2	N_2	Number of pressure transmitters fitted in subsystem 2	1 - 4
	K_2	Number of pressure transmitters required to activate for subsystem 2	1 – N_2
	P_2	Pressure transmitter type	
n/a	θ_1	Inspection interval for subsystem 1	1 week – 2 years
n/a	θ_2	Inspection interval for subsystem 2	1 week – 2 years

Table 1. Design Variables

Associated System Component	Design Variable	Initial Design	Design Variable Values of GA [xx]	Design Variable Values of GSDOA
ESD Valve	E	1	0	2
	V_1	Type 1	n/a	Type 1
HIPS Valve	H	1	2	1
	V_2	Type 1	Type 2	Type 2
Pressure Transmitter 1	N_1	1	2	3
	K_1	1	1	2
	P_1	Type 1	Type 1	Type 1
Pressure Transmitter 2	N_2	1	3	2
	K_2	1	2	1
	P_2	Type 1	Type 1	Type 2
n/a	θ_1	n/a	29	46
n/a	θ_2	n/a	32	29

Table 2. Results Comparison

Cost of the system with the design generated using the GA in [9] was 822 units, maintenance down time was 128,43 hours and system unavailability

was 7.6×10^{-4}. System characteristics for the design obtained using GSDOA are as following; system cost 642 units, maintenance down time 83.96 hours and system unavailability is 2.98×10^{-7}.

5. Conclusions

The introduced algorithm is employed to solve a general system design optimisation problem. The objective of the optimisation process is to define a particular set of system components that would constitute an optimal system design. As a result the algorithm determines the case where the system failure probability is minimised and the utility of available resources is optimised.

A simple High Integrity Protection System (HIPS) was employed as an application example. Comparison of the optimisation results obtained when using GSDOA and a simple GA specifically developed for HIPS optimisation shows that GSDOA is capable to provide good optimisation results. Using GSDOA system design was found with unavailability equal to 2.98×10-7 and cost 642 units. These characteristics are better that the ones of the system design obtained using the simple GA.

References

1. Martorell S., Sanchez A., Carlos S. and Serradell V. Alternatives and Challenges in Optimizing Industrial Safety Using Genetic Algorithms. *Reliability Engineering & System Safety* **86** [1] 25-38 (2004).
2. Ren Y. and Dugan J.B. Design of Reliable Systems Using Static & Dynamic Fault Trees. *IEEE Transactions on Reliability*, **47** [3], 234-244 (1998).
3. Barlow R. E. and Lambert H. E. in *Reliability and Fault Tree Analysis: Theoretical and Applied Aspects of System Reliability and Safety assessment*, ed. Barlow R. E., Fussell J.B. and Singpurwalla N.D. Society for Industrial and Applied Mathematics, Philadelphia, pp 7-35 (1975).
4. Rauzy A., New algorithms for fault trees analysis. *Reliability Engineering & System Safety* **40** [3], 203-211 (1993).
5. Goldberg D. E., *Genetic algorithms in search, optimization, and machine learning*. Wokingham: Addison-Wesley. (1989).
6. Levitin G. Genetic algorithms in reliability engineering. *Reliability Engineering & System Safety,* **91** [9], 975-976 (2006).
7. Cepin M., Optimization of Safety Equipment Outages Improves Safety. *Reliability Engineering & System Safety,* **77** [1] 71-80 (2002).
8. Andrews J.D. and Pattison R.L. Optimal Safety System Performance. *Proc. Ann. Reliability & Maintainability Symp*, 76-83 (1997).
9. Pattison R.L. *Safety System Design Optimisation* (PhD Thesis). Loughborough: Loughborough University (1999).
10. Remenyte-Prescott R. *System Failure Modelling Using Binary Decision Diagrams* (PhD Thesis). Loughborough: Loughborough University (2007).

11. Coit D.W., Smith A.E. and Tate D. M. Adaptive Penalty Methods for Genetic Optimization of Constrained Combinatorial Problems. *INFORMS Journal on Computing* 8 [2] 173-182 (1996).
12. Chambers L.D. (ed.) *The practical handbook of genetic algorithms*. Boca Raton, Fla: Chapman & Hall/CRC (2001).