DETERMINATION OF PRIME IMPLICANTS BY DIFFERENTIAL EVOLUTION FOR THE DYNAMIC RELIABILITY ANALYSIS OF NON-COHERENT NUCLEAR SYSTEMS

Francesco Di Maio[1], Samuele Baronchelli[1], Matteo Vagnoli[1], Enrico Zio[1,2]

[1] *Energy Department, Politecnico di Milano*

Via La Masa 34, 20156 Milano, Italy

francesco.dimaio@polimi.it

[2] *Chair on System Science and Energetic Challenge*

Foundation EDF – Electricite de France

Ecole Centrale, Paris, and Supelec, Paris, France

ABSTRACT

We present an original computational method for the identification of prime implicants (PIs) in non-coherent structure functions of dynamic systems. This is a relevant problem for dynamic reliability analysis, when dynamic effects render inadequate the traditional methods of minimal cut-set identification. PIs identification is here transformed into an optimization problem, where we look for the minimum combination of implicants that guarantees the best coverage of all the minterms. For testing the method, an artificial case study has been implemented, regarding a system composed by five components that fail at random times with random magnitudes. The system undergoes a failure if during an accidental scenario a safety-relevant monitored signal raises above an upper threshold or decreases below a lower threshold. Truth tables of the two system end-states are used to identify all the minterms. Then, the PIs that best cover all minterms are found by Modified Binary Differential Evolution. Results and performances of the proposed method have been compared with those of a traditional analytical approach known as Quine-McCluskey algorithm and other evolutionary algorithms, such as Genetic Algorithm and Binary Differential Evolution. The capability of the method is confirmed with respect to a dynamic Steam Generator of a Nuclear Power Plant.

Keywords: Dynamic Reliability; Prime Implicants; Non-coherent Structure Functions; Modified Binary Differential Evolution (MBDE); Genetic Algorithm (GA); Binary Differential Evolution (BDE); Steam Generator (SG); Nuclear Power Plant (NPP).

1. INTRODUCTION

The reliability analysis of systems with significant hardware/software/human interactions is difficult, because the response of the system under accidental scenarios depends on the time of occurrence and on the magnitude of the events [1; 2]. Further, it turns out that the logic of these systems can give rise to non-coherent structure functions, where both failed and working states of the same components can lead the system to failure [3]; for example, if in a system made up of three components *J*, *K*, *L* it fails with components states (J, $\overline{L}$, K), with the negation sign indicating that the component is failed, whereas it is working when the components states are ($\overline{J}$, $\overline{L}$, K), then the system is non-coherent. The traditional Probabilistic Risk Assessment (PRA) modeling tools, e.g. Fault Tree and Event Tree Analysis, have difficulties in including the specific timing and magnitude of the events. On the other hand, so-called dynamic reliability methods can complement the traditional methods to accounts for the interactions among the physical parameters of the processes (temperature, pressure, speed, etc.), the human operators actions and the failures of the components [2;4-6] and to identify the system prime implicants (PIs), i.e., the event product terms that render true the structure function and that cannot be covered by more reduced implicants [7], even if the structure functions are non-coherent[1]. PIs have been introduced as dynamic equivalent of Minimal Cut Sets (MCSs) for conveying the information on the minimum combinations of failures that lead (non-coherent and/or dynamic) the system to failure and that cannot be covered any other implicant [8].

Traditionally, non-coherent structure functions have been interpreted as indication of poor system design. However, in [9] it is shown that PIs identification can help developing an effective maintenance schedule for non-coherent systems. For example, suppose that $\overline{J},\overline{K},L$ (components *J* and *K* failed and component *L* working) is a PI that causes a catastrophic system failure. This shows that, if components *J*, *K* and *L* have failed, *L* should be the last component to be repaired in order to avoid system failure. Furthermore, PIs identification allows taking additional counteracting measures

[1]For clarity sake, we recall that an implicant is a product of Boolean variables, each one associated with a system component and representing its failed (1) or safe state (0), that leads the system to failure: differently from minterms, in implicants not all the variables have to appear when these (missing) variables cannot affect the system behavior. Implicants, thus, can cover more minterms that differ in only one (or more) variable that does not influence the system failure (as well as cut sets and minterms in traditional PRA).

to prevent system failure, for example by forcing failure of component *L* when component *J* and *K* have already failed [10].

Fault tree analysis is undoubtedly an useful and efficient tool for minimal cut set identification, but not for PIs identification, since it can only deal with coherent structure functions [11]. The problem of extending the analysis to non-coherent fault trees has, then, been tackled in different ways: the simplification of non-coherent structure functions expressed in canonical forms has been raised by [7] and solved by [12], allowing a preliminary identification of PIs; the problem has also been tackled by means of graphical methods such as Karnaugh maps [13]. However, the actual implementation of these methods becomes very time-consuming when the number of variables involved in the given structure function increases. The computational efficiency has been improved resorting to various Partitioned List algorithms [14] and fast Binary Decision Diagram (BDD) algorithms [15]: in [16], a modification of a minimal cut sets algorithm known as Simple Prime Implicant Set Algorithm is proposed, although it does not always produce complete PI sets, whereas in [17] a method is proposed to convert the fault tree of a non-coherent structure function into a BDD for PIs identification, where each of the basic events of the tree is represented as a node with two branches (branch 1 and 0, corresponding to the component failure and working states respectively). This latter approach has been adapted in [18] for PI identification based on Dynamic Flowgraph Methodology (DFM).

The difficulty in developing efficient computational methods for PIs identification lays in the fact that this can be seen as an NP-hard problem of covering a set (the minterms) with elements from given subsets (the PIs) [19]: each given subset has an associated cost proportional to its dimension and the objective of the problem is to choose the smallest group of subsets whose union contains the whole set with minimal cost, as we shall see in what follows.

In this paper, we develop a new method for identifying all PIs of a non-coherent structure function resorting to the powerful evolutionary algorithm of Differential Evolution (DE) [20]. The PIs are found by solving by DE a properly defined optimization problem, for determining the exact (not approximated) solution of the Set Covering Problem (SCP) [21; 22]: in this way, none of the prime (minimal) failure scenarios (i.e., the PIs) can be neglected by the identification method.

The paper is organized as follows. In Section 2, the artificial case study used to generate the scenarios for the dynamic reliability analysis is presented. In Section 3, the model of a Steam Generator (SG) of a Nuclear Power Plant (NPP) is presented [23]. In Section 4, PIs identification is formulated as an optimization problem and tackled by resorting to the DE-based approach. In Section 5, the results of the application of the approach to the scenarios of the artificial case and of the SG are presented. Conclusions and remarks are given in Section 6.

2. THE ARTIFICIAL CASE STUDY

For ease of illustration of the method proposed, we build an artificial case study by simulating the accidental scenarios for a system made of 5 components (denoted as *A*, *B*, *C*, *D* and *E*), that can fail at random times with random magnitudes, giving rise to different scenarios whose evolutions are represented by 4 monitored signals. Multiple component failures can occur during the system life, set to T=7 [h]. For the simulation, a Monte Carlo sampling procedure for injecting faults of random magnitudes at random times is implemented. In particular, times and magnitudes of faults are obtained by a stratified sampling with respect to the possible accident scenarios [24]. The number of components that fail is sampled from a binomial distribution with parameters n=5 (equal to the number of components) and p=0.8 (so that even rare multiple fault events are included in the set of accident scenarios). The first failure time is sampled from a uniform distribution [0,1] [h], and the successive failure times are sampled by a stick-breaking strategy from the conditional distributions, uniform from the last sampled time up to 7 [h]. This sampling strategy models a wearing system, with average failure rate increasing in time.

The equations deliberately used to simulate the signal evolutions in time during the accidental scenarios are (Tab. 1):

$$y(t) = 2\alpha_1 a\left[1 + erf\left(\frac{t-\mu}{\sqrt{2}}\right)\right] + 10^{-3\omega} \tag{1}$$

$$y(t) = \alpha_2\left(c^{d^t} - c\right) + 10^{-3\omega} \tag{2}$$

$$y(t) = \alpha_3 bt + 10^{-3\omega} \tag{3}$$

where a, b, c, d, μ, ω, α_1, α_2 and α_3 are randomly sampled from the distributions listed in Tab. 2. Parameters α_1, α_2 and α_3 represent the magnitudes of the faults of the accidental scenarios. All parameters and variables have arbitrary units.

Failed Component	Signal 1	Signal 2	Signal 3	Signal 4
A	Eq. (1)	Eq. (1)	Eq. (3)	Eq. (1)
B	Eq. (1)	Eq. (2)	Eq. (3)	Eq. (1)
C	Eq. (2)	Eq. (3)	Eq. (1)	Eq. (1)
D	Eq. (2)	Eq. (3)	Eq. (2)	Eq. (1)

E	Eq. (3)	Eq. (3)	Eq. (3)	Eq. (1)

Tab.1. *Equations used to simulate the signals evolutions in time for each failed component*

Parameter	***Distribution***	***Mean value***	***Standard deviation***
a	*Gaussian*	*0.4*	*0.017*
b	*Gaussian*	*0.4*	*0.017*
c	*Gaussian*	*1.3*	*0.033*
d	*Gaussian*	*1.3*	*0.017*
α_1	*Gaussian*	*1*	*0.083*
α_2	*Gaussian*	*1.05*	*0.033*
α_3	*Gaussian*	*1*	*0.033*
μ	*Gaussian*	*2.45*	*0.083*
ω	*Gaussian*	*0*	*1*

Table 2. *Parameters distribution*

We take signal 1 as the safety-relevant parameter to be monitored against pre-defined safety thresholds: if it exceeds the upper threshold value of 2.5, the system fails in the "High" end state; if it decreases below the lower threshold value of -1.5, the system end state is "Low" [25]. In Fig. 1, the evolution of the 4 signals for 10 randomly sampled accidental scenarios are shown. Signals measurements are plotted in continuous lines; the upper and lower thresholds are in dotted and dashed lines, respectively.

Fig. 1 shows that under different scenarios, the signals can increase or decrease. This can occur in reality where, for example, if a valve of the coolant injection system of a nuclear power plant (NPP) fails to open during a loss of coolant accident (LOCA), an in-vessel temperature growth is measured, which could arrive at exceeding the upper threshold [26]; if the pressurizer safety relief valve fails to close, the water level drops below the low-level safety threshold, leading the system into the undesirable state of uncovered electric heaters [27].

Yet, it is important to underline that the procedure implemented in this work for sampling the fault events is not intended to reproduce the actual stochastic failure behavior of the components of a real system; rather, the choices and hypotheses for modeling the faults (e.g. system life, number of faults and distributions of failure times and magnitudes) have been arbitrarily made with the aim of favoring multiple failures in the sequences and capturing the dynamic influence of their order, timing and

magnitude including possible compensatory effects for which a failure later in time compensates for the impact of another earlier failure, thus highlighting non-coherent system behavior.

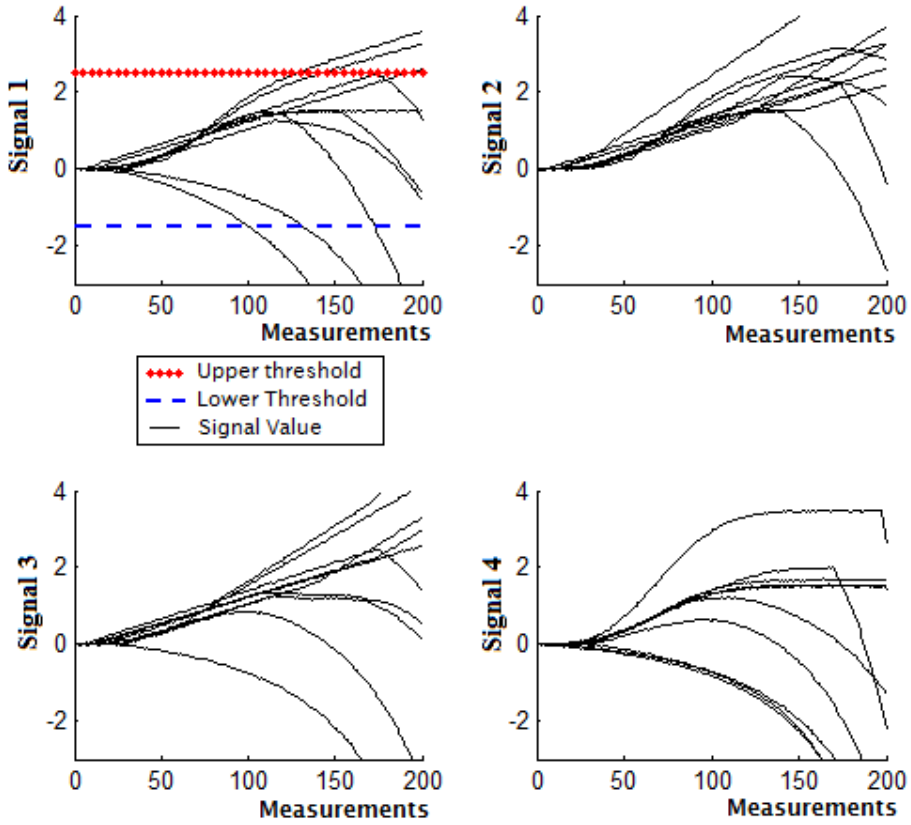

Fig 1. *Examples of the behavior of the 4 monitored signals during simulated accidental scenarios*

2.1 Non-coherence

Considering the binary (safe or faulty) states of the five components of the system, the number of possible system configurations is equal to 32. One simulation has been run for each system configuration with the hypothesis that faults are assumed to occur at the beginning of the scenarios and their magnitudes are taken equal to their mean values of Tab. 2. Tab. 3 shows the truth-table of the system, i.e., all possible system configurations, with the end state "Low" or "High" they lead to.

System Configuration	Component A	Component B	Component C	Component D	Component E	End State		
						Low	Safe	High
1	-	-	-	-	-	No	Yes	No
2	x	-	-	-	-	No	Yes	No
3	-	x	-	-	-	No	Yes	No
4	-	-	x	-	-	Yes	No	No
5	-	-	-	x	-	Yes	No	No
6	-	-	-	-	x	No	No	Yes
7	x	x	-	-	-	No	No	Yes

8	x	-	x	-	-	Yes	No	No
9	x	-	-	x	-	Yes	No	No
10	x	-	-	-	x	No	No	Yes
11	-	x	x	-	-	Yes	No	No
12	-	x	-	x	-	Yes	No	No
13	-	x	-	-	x	No	No	Yes
14	-	-	x	x	-	Yes	No	No
15	-	-	x	-	x	No	Yes	No
16	-	-	-	x	x	No	Yes	No
17	x	x	x	-	-	No	Yes	No
18	x	x	-	x	-	No	Yes	No
19	x	x	-	-	x	No	No	Yes
20	x	-	x	x	-	Yes	No	No
21	x	-	x	-	x	No	Yes	No
22	x	-	-	x	x	No	Yes	No
23	-	x	x	x	-	Yes	No	No
24	-	x	x	-	x	No	Yes	No
25	-	x	-	x	x	No	Yes	No
26	-	-	x	x	x	Yes	No	No
27	x	x	x	x	-	Yes	No	No
28	x	x	x	-	x	No	No	Yes
29	x	x	-	x	x	No	No	Yes
30	x	-	x	x	x	Yes	No	No
31	-	x	x	x	x	Yes	No	No
32	x	x	x	x	x	No	No	Yes

Table 3. *Truth-table for the 32 system configurations and the "Low", "Safe" and "High" end states. Legend: - = safe component, x = faulty component*

The analysis of the truth-table points out that the system failure logic is represented by a non-coherent structure function. In fact, as it can be shown in Fig. 2 and Fig. 3, both failed and working states of the components can contribute to the failure of the system. In particular, in Fig. 2 (left) the safety-relevant signal 1 for the system configuration 11 of Tab. 3 (components *B* and *C* failed, and components *A*, *D* and *E* working) is shown; on the other hand, in Fig. 2 (right) the same signal for system configuration 17 of Tab. 3 (components *A*, *B* and *C* failed, and components *D* and *E* working) is plotted: from 11 to 17, adding the failure of component *A* brings the system from a "Low" end state to a "Safe" end state, violating coherence requirements.

In Fig. 3 (left), the safety-relevant signal 1 for the system configuration 6 of Tab. 3 (component *E* failed, and components *A*, *B*, *C* and *D* working) is shown; on the other hand, in Fig. 3 (right) the same signal for system configuration 15 of Tab. 3 (components *C* and *E* failed, and components *A*, *B* and *D* working) is plotted: from 6 to 15, adding the failure of component *C* brings the system from a "High" end state to a "Safe" end state, violating coherence requirements.

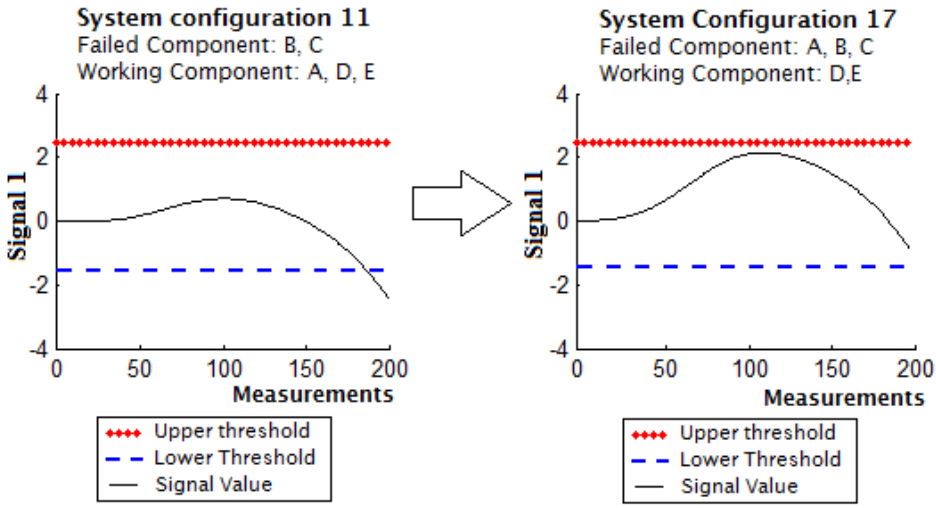

Fig. 2. *Example of non-coherence for the "Low" end state*

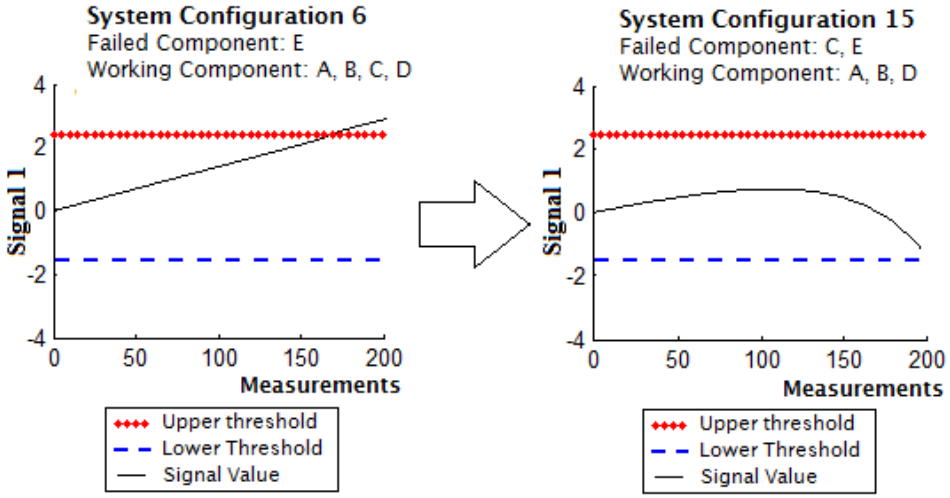

Fig. 3. *Example of non-coherence for the "High" end state*

Furthermore, when we take into account uncertainties on timing and magnitudes of components failures, the dynamic aspects render non-coherence even more evident. Fig. 4 shows the frequency of the three system end states ("High", Safe and "Low") for the 32 system configurations reported in Tab. 3, estimated from the simulation of 10,000 accidental scenarios for each system configuration with random components failure times and magnitudes. Most of the configurations do not lead unequivocally to one end state: on one side, this means that even though the configuration is the same, when the failures of the components occur at different times or with different magnitudes, the end state can be different. For example, if a failure occurs towards the end of the mission time (as opposed to the start of the period), it may not lead to system failure [24]. On the other side, Fig. 4 shows that

as a new failure occurs, the faulty end states frequencies can become smaller or, vice versa, as a faulty component is repaired, the safe end state frequencies can become smaller. For example, adding one failure from system configuration 14 (components *C* and *D* failed and components *A*, *B* and *E* working) to system configuration 26 (components *C*, *D* and *E* failed and components *A* and *B* working), or from system configuration 23 (components *B*, *C* and *D* failed and components *A* and *E* working) to system configuration 31 (components *B*, *C*, *D* and *E* failed and component *A* working), the safe end state frequencies increase, and correspondingly the "Low" and "High" end state frequencies decrease.

These examples show the need in dynamic reliability analysis to focus on the PIs of the system, rather than on the identification of its minimal cut sets, due to the evident non-coherence of the structure function.

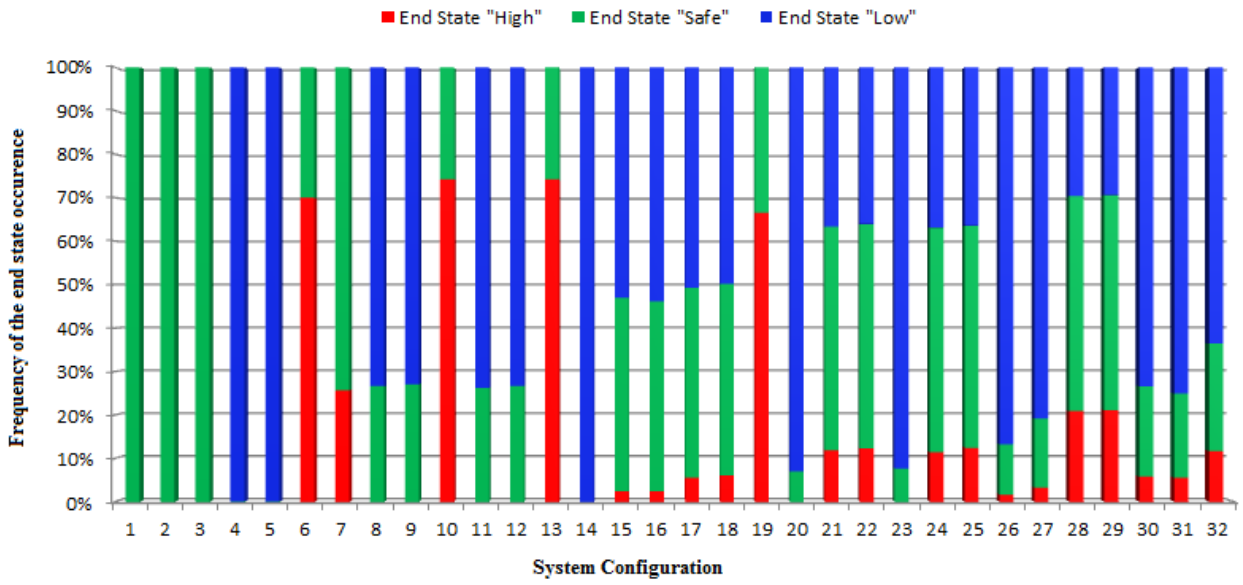

Fig. 4. *Histograms of the frequency of end states for each of the 32 system configurations listed in Tab. 3*

3. THE STEAM GENERATOR OF A NUCLEAR POWER PLANT

The U-Tube Steam Generator (UTSG) under consideration is sketched in Fig. 5. The reactor coolant enters the UTSG at the bottom, moves upward and then downward in the inverted U-tubes, transferring heat to the secondary fluid before exiting at the bottom. The secondary fluid, the feedwater (Q_e), enters the UTSG at the top of the downcomer, through the space between the tube bundle wrapper and the SG shell. The value of Q_e is regulated by a system of valves: a low flow rate valve, used when the operating power (P_o) is smaller than 15% of nominal power (P_n) and a high flow rate valve when $P_o > 0.15\ P_n$ [23].

In the secondary side of the tube bundle, water heats up, reaches saturation, starts boiling and turns into a two-phase mixture. The two-phase fluid moves up through the separator/riser section, where steam is separated from liquid water, and through the dryers, which ensure that the exiting steam (Q_v) is essentially dry. The separated water is recirculated back to the downcomer. The balance between the exiting Q_v and the incoming Q_e governs the change in the water level in the SG. Because of the two-phase nature, two types of water level measurements are considered, as shown in Fig. 5, each reflecting a different level concept: the Narrow Range Level (N_{rl}) is calculated by pressure difference between two points close to the water level and indicates the mixture level, whereas, the Wide Range Level (W_{rl}) is calculated by pressure difference between the two extremities of the SG (steam dome and bottom of the downcomer) and indicates the collapsed liquid level that is related with the mass of water in the SG.

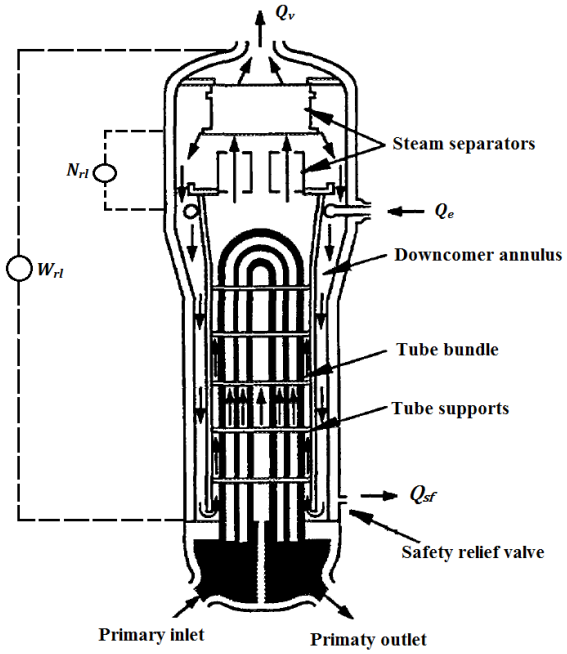

Fig 5. *Schematic of the UTSG [29]*

At low P_o, "swell and shrink" phenomena are also modeled to reproduce the dynamic behavior of the SG: when Q_v increases, the steam pressure in the steam dome decreases and the two-phase fluid in the tube bundle expands causing N_{rl} to initially swell (i.e., rise), instead of decreasing as would have been expected by the mass balance; contrarily, if Q_v decreases or Q_e increases, a shrink effect occurs [29]. A similar model has been presented in [23].

The goal of the system is to maintain the SG water level at a reference position (N_{ref}): the SG fails if the N_{rl} rises (falls) above (below) the threshold, N_{high} (N_{low}), in which case automatic reactor or turbine trips are triggered. Indeed, if the N_{rl} exceeds N_{high}, the steam separator and dryer lose their functionality and excessive moisture is carried in Q_v, degrading the turbine blades profile and the turbine efficiency; if N_{rl} decreases below N_{low}, insufficient cooling capability of the primary fluid occurs. Similarly, the W_{rl}, is relevant for the cooling capability of the primary circuit [29].

A dedicated, simulation model has been implemented in SIMULINK to simulate the dynamic response of the UTSG at different P_o values. Both feedforward and feedback digital control schemes have been adopted. The feedback controller is a PID that provides a flow rate Q_{pid} resulting from the residuals between N_{rl} and N_{ref}, whereas the feedforward controller consists in a safety relief valve that is opened if and only if N_{rl} exceeds the N_{hl}, and removes a constant flow safety flow rate (Q_{sf}). The block diagram representing the SIMULINK model of the SG is shown in Fig. 6: the controlled variable is N_{rl}, whereas the control variable is Q_e.

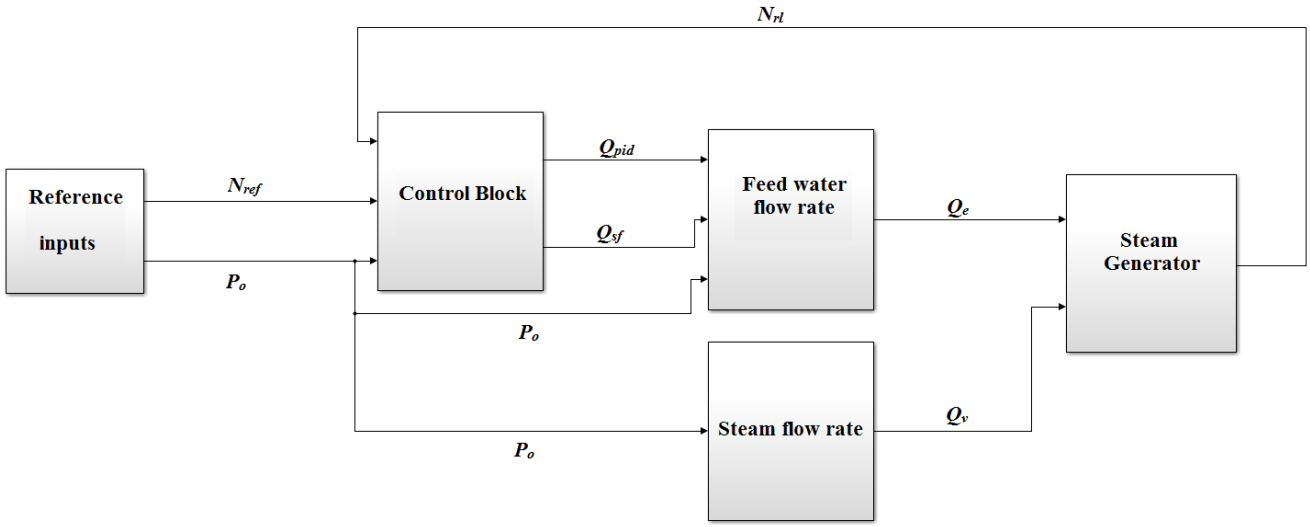

Fig 6. *Block diagram representing the SIMULINK model of the SG.*

3.1 The set of possible failures

We assume component failures to occur at the beginning of the scenario (with T_{miss} equal to 4000 (s)) [1]. We here analyze the system in constant P_o=80% P_n scenarios with respect to high level failure mode. Choices and hypotheses for modeling the failures have been arbitrarily made with the aim of generating multiple failures and the choice of a mission time (T_{miss}) equal to 4000 (s) has been made because it is a long enough interval of time to allow the complete development also of slow dynamic accident scenarios. The set of multiple component failures that can occur are:

1. The outlet steam valve (component T) can fail stuck at 85% of the nominal Q_v that should be provided at P_o.
2. The communication between the sensor that monitors N_{rl} and the PID controller (component U) can fail so that the PID is provided with the same input value of the previous time step.
3. The safety relief valve (component V) can fail stuck at a value Q_{sf} = 50.5 (kg/s).
4. The PID controller (component Z) can fail stuck providing a flow rate Q_{pid} = 12.35% of the nominal Q_e that should be provided at P_o.

Considering the binary (safe or faulty) states of the five components of the system, the number of possible system configurations (for which a simulation has been run) is equal to 16. Tab. 4 shows the truth-table of the system, i.e., all possible system configurations, with the end state "Low" or "High" they lead to.

System configuration	Failure of the outlet steam valve	Failure of the safety relief valve	Level sensor-PID controller communication interruption	Failure of the PID controller	End State		
					Low	Safe	High
1	-	-	-	-	No	Yes	No
2	X	-	-	-	No	No	Yes
3	-	X	-	-	No	Yes	No
4	-	-	X	-	No	Yes	No
5	-	-	-	X	No	No	Yes
6	X	X	-	-	No	Yes	No
7	X	-	X	-	No	No	Yes
8	X	-	-	X	No	No	Yes
9	-	X	X	-	Yes	No	No
10	-	X	-	X	No	Yes	No
11	-	-	X	X	No	No	Yes
12	X	X	X	-	No	No	Yes
13	X	X	-	X	No	No	Yes
14	X	-	X	X	No	No	Yes
15	-	X	X	X	No	Yes	No
16	X	X	X	X	No	No	Yes

Table 4. *Truth-table for the 16 system configurations and the "Low", "Safe" and "High" end states. Legend: - = safe component, x = faulty component*

The analysis of the truth-table points out that the system failure logic is represented by a non-coherent structure function. In fact, as it can be shown in Fig. 7 and Fig. 8, both failed and working states of the components can contribute to the failure of the system. In particular, in Fig. 7 (left) the N_{rl} level for system configuration 2 (steam valve failure) is shown; on the other hand, in Fig. 7 (right) the N_{rl} level for system configuration 6 (steam and safety valves failures) is plotted: adding the failure of the safety valve brings the system from a "High" end state to a "Safe" end state, violating coherence requirements.

In Fig. 8 (left) the N_{rl} level for system configuration 9 (safety valve and communication failures) is shown; on the other hand, in Fig. 8 (right) the N_{rl} level for system configuration 15 (safety valve, communication and PID failures) is plotted: adding the failure of the PID brings the system from a "Low" end state to a "Safe" end state, violating coherence requirements.

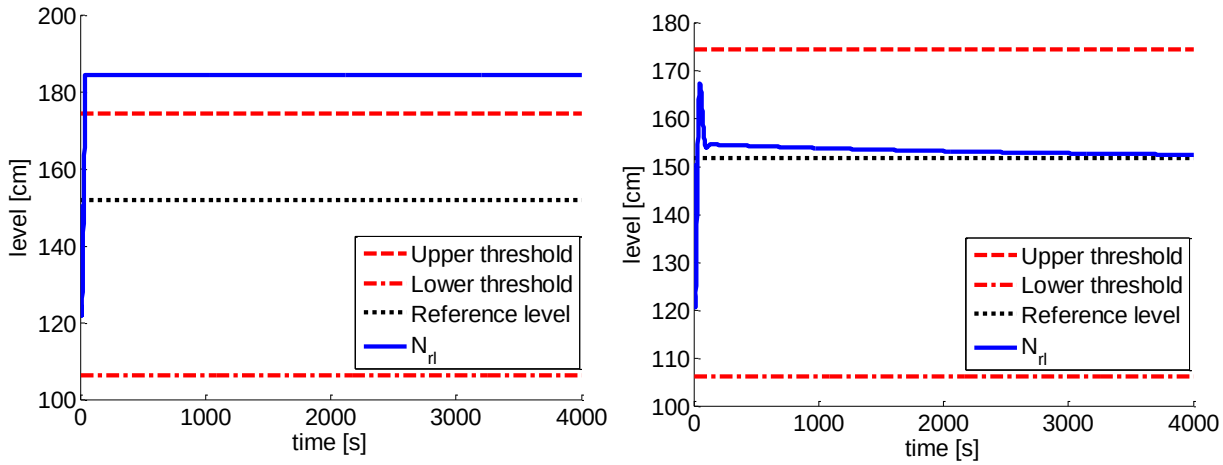

Fig 7. *Example of non-coherence for the "High" end state.*

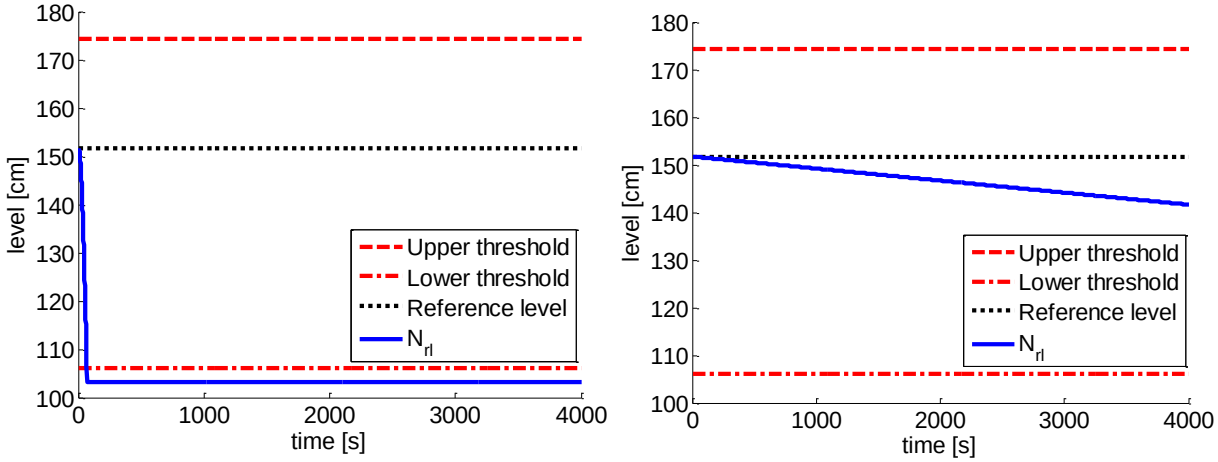

Fig 8. *Example of non-coherence for the "Low" end state.*

4. A NOVEL METHOD FOR PIs IDENTIFICATION

In this paper, the problem of PIs identification is innovatively handled resorting to the DE algorithm for solving a set covering problem (SCP) [22; 30]. Differently from [30], here we develop a DE search strategy to identify PIs and not the classical MCSs. The SCP is the problem of covering at minimal cost (that is defined depending on the context of the application) the columns of a zero-one matrix $A=[a_{ij}]$, where $i=1,2,...,R$ and $j=1,2,...,C$, by a subset of the rows. Defining $x_i=1$ if row i is in the solution, and $x_i=0$ otherwise, the SCP aims at identifying the set of x_i with the lower cost (Eq. (4))

that guarantee the coverage of each column *j* by at least one row (i.e., for each *i-th* row corresponding to the implicant chosen, there is at least one entry equal to 1 in one of the *C* columns) (Eq. (5)), viz:

$$minimize \sum_{i=1}^{R} w_i x_i \tag{4}$$

$$subject\ to \sum_{i=1}^{R} a_{ij} x_i \geq 1 \tag{5}$$

where w_i is the positive cost weight associated to the *i-th* row (which, again, depends on the specific problem). In the PIs identification, let $A=[a_{ij}]$ be an implicant chart (i.e., a matrix representing the minterms covered by each implicant, where $a_{ij}=1$ if the *i*-th implicant covers the *j*-th minterm, $a_{ij}=0$ otherwise), m_j denote the *j*-th minterm (i.e., the product of all the Boolean variables associated with a system component, representing its failed (1) or safe state (0), that leads the system to failure), x_i denote the *i*-th implicant of the structure function.

A cost vector $\bar{w} = (w_1, w_2, ..., w_R)$ assigns a positive cost w_i to each implicant *i*, e.g. cost of components in manufacturing industry [19], number of trips that can be performed by a single crew in transportation company [31]. For generality, here we define the cost w_i as the number of Boolean variables (either true or complemented) associated to the system components included in the *i*-th implicant. For this problem, the solution space is the set of all possible combinations of *1,2,...,R* implicants (hence the size of the solution space is 2^R-1, excluding the possibility where no implicant is chosen). Each solution $\hat{\bar{x}}_{opt}$ is represented by a specific combination of independent variables, or, mathematically speaking, by a *R*-dimensional vector $\bar{x} = (x_1, x_2, ..., x_R)$ (hereafter called chromosome within the differential evolution (DE) optimization method that will be adopted) that is a hypothetical solution of the optimization problem (4) and (5). A value of 1 in the *i*-th vector position x_i implies that the implicant *i* is chosen to be in the cover; a value of 0, otherwise [19].

For clarification, let us consider the system made up by three components (*J*, *K* and *L*) whose reliability block diagram is shown in Fig. 9. The $C=5$ minterms m_j that lead this system to failure are reported in Tab. 5.

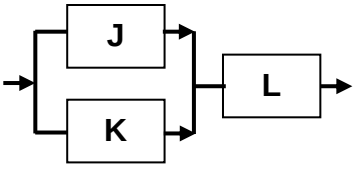

Fig. 9. *Reliability block diagram of the system*

	J	***K***	***L***
m_1	0	0	1
m_2	0	1	1
m_3	1	0	1
m_4	1	1	0
m_5	1	1	1

Table 5. *List of the faulty minterms m_j of the system of Fig. 9 (1=failed component, 0=safe component)*

The R=11 implicants (x_i in Eqs. (4) and (5)) of the system of Fig. 9, and their costs (w_i in Eq. (4)), are reported in Tab. 6. Intuitively, the PIs of the system of Fig. 9 are x_{10} and x_{11}.

	J	***K***	***L***	***Cost (w)***
x_1	0	0	1	3
x_2	0	1	1	3
x_3	1	0	1	3
x_4	1	1	0	3
x_5	1	1	1	3
x_6	0	-	1	2
x_7	-	0	1	2
x_8	1	-	1	2
x_9	-	1	1	2
x_{10}	1	1	-	2
x_{11}	-	-	1	1

Table 6. *List of the implicants x_i of the system of Fig. 9 (1=failed component, 0=safe component, -=component state does not influence the system failure)*

In Tab. 7, the implicant chart A, whose rows are a_{ij} in Eq. (5), for the system is finally shown.

	m_1	m_2	m_3	m_4	m_5
x_1	1	0	0	0	0
x_2	0	1	0	0	0
x_3	0	0	1	0	0
x_4	0	0	0	1	0
x_5	0	0	0	0	1
x_6	1	1	0	0	0
x_7	1	0	1	0	0

x_8	0	0	1	0	1
x_9	0	1	0	0	1
x_{10}	0	0	0	1	1
x_{11}	1	1	1	0	1

Table 7. *Implicant chart A for the system of Fig.9 (a_{ij}=1, minterm is covered, a_{ij}=0, minterm is uncovered)*

Within the evolutionary algorithm context, the optimal cover $\bar{x}_{opt}$ is the chromosome $\bar{x} = (0,0,0,0,0,0,0,0,0,1,1)$ which means that only x_{10} and x_{11} are chosen to be in the solution, i.e., are PIs.

For solving the above-defined SCP, we resort to Differential Evolution (DE), which belongs to the class of Evolutionary Algorithms (EAs) [32]. A main advantage of DE with respect to other EAs is the fact that the evolutionary operations used in DE are specifically built for optimization over continuous spaces and based on a floating-point representation [33-35].

DE entails three phases called mutation, crossover and selection. This is the original scheme proposed in [20]: at the *G+1*-th generation, for each gene x_i in the chromosome vector $\bar{x}_G = (x_1, x_2, ..., x_R)_G$ of the population of *NP* different chromosomes at the *G*-th generation, a noisy gene v_i of the noisy vector $\bar{v}_{G+1} = (v_1, v_2, ..., v_R)_{G+1}$, is generated by randomly adding to the *i*-th gene of the *l*-th chromosome the weighted difference between two other randomly selected *k*-th and *m*-th chromosomes from the population.

$$v_i = x_{i_{(l)}} + F(x_{i_{(k)}} - x_{i_{(m)}}) \tag{6}$$

where the weighting factor $F \in [0,2]$ is a user-defined parameter, kept constant during the optimization and $x_{i_{(l)}}$, $x_{i_{(k)}}$ and $x_{i_{(m)}}$ are the *i*-th gene values of the three randomly chosen individuals, with $l,k,m \in \{1,2,...,NP\}$.

To maintain the diversity inside the perturbed population, and shuffle old and new information, after mutation, $\bar{v}_{G+1}$ is not directly compared with $\bar{x}_G$, but it is further modified by the crossover process, in which $\bar{v}_{G+1}$ and $\bar{x}_G$ are mixed according to some rule to create the trial vector $\bar{u}_{G+1}$, which inherits from them different pieces of chromosome. The most common crossover type adopted is the binomial: $\bar{u}_{G+1}$ is built by a modified Bernoulli trial rule, gauged by the control parameter $CR \in [0,1]$, which influences the probability for $\bar{v}_{G+1}$ to be selected for the mutation process. Each gene u_i of the trial vector is equal to

$$u_i = \begin{cases} v_i & \text{if } U(0,1] \le CR \text{ or } i = irand(R) \\ x_i & \text{otherwise} \end{cases} \tag{7}$$

where *U(0,1*] denotes the uniform continuous random value in $(0,1]$ and *irand(R)* is a uniform discrete random number from the set $\{1,2,...,R\}$, where *R* is the length of the chromosome.

The trial vector obtained $\bar{u}_{G+1}$, then, enters the selection process where it is compared with (and eventually substitutes) the target vector $\bar{x}_G$ that is partially its parent according to the crossover rule. Referring to minimization, if the fitness, i.e., the cost, of $\bar{u}_{G+1}$ is less than the fitness of $\bar{x}_G$, the first will be a member of the next generation *G+1*, replacing the target vector, and the trial vector is discarded

$$\bar{x}_{G+1} = \begin{cases} \bar{u}_{G+1} & \text{if } fitness(\bar{u}_{G+1}) < fitness(\bar{x}_G) \\ \bar{x}_G & \text{otherwise} \end{cases} \tag{8}$$

In this work, we aim at comparing the performance of two different DEs, that differ in the mutation step and are called "Binary Differential Evolution" (BDE) [33] and "Modified Binary Differential Evolution" (MBDE) [34].

4.1 Binary Differential Evolution

BDE is based on a mapping operator, defined as Eq. (9), that is constructed to map the gene x_i in a discrete domain (in our case it is a binary domain) into a continuous domain by partitioning the interval [0, 1] into two equal subintervals [0,0.5) and [0.5,1], (i.e., if x_i=0 and *rand* is a random number in [0,1), then, its image belongs to the first subinterval, whereas if x_i=1 its image belongs to the second interval).

$$x_i = \begin{cases} 0.5 \cdot rand & \text{if } x_i = 0 \\ 0.5 + rand \cdot rand & \text{if } x_i = 1 \end{cases} \tag{9}$$

After variable x_i is mapped in the new domain, the mutation operator of Eq. (6) is applied. To ensure that the resulting gene generated by the mutation operator in the original DE falls into the interval [0,1], a sigmoid function is applied to obtain v_i:

$$v_i = \frac{1}{1+e^{-v_i}} \tag{10}$$

Before the crossover phase, an inverse mapping operator is used:

$$v_i = \begin{cases} 0 & \text{if } v_i \in [0,0.5) \\ 1 & \text{if } v_i \in [0.5,1] \end{cases} \tag{11}$$

Then, the procedure follows traditional DE steps of crossover and selection.

4.2 Modified Binary Differential Evolution

MBDE is based on the mutation phase of the standard DE: it entails embedding Eq. (6) into a probability estimation operator (Eq. (12)) that helps generating the mutated individuals, accounting for the information of the parent population:

$$P(x_i) = \frac{1}{1+e^{-\frac{2b\bullet\left[x_{i_{(l)}}+F\bullet\left(x_{i_{(k)}}-x_{i_{(m)}}\right)-0.5\right]}{1+2F}}} \tag{12}$$

where b is a positive real constant, usually set to the value of *6*; F is the weighting factor and $x_{i_{(l)}}, x_{i_{(k)}}$ and $x_{i_{(m)}}$ are the i-th genes of three randomly chosen individuals, as in Eq. (6) for the standard DE.

According to the probability estimation vector $P(\bar{x}) = \left[P(x_1), P(x_2), \ldots, P(x_R)\right]$, created by Eq. (12), the corresponding genes of the noisy vector $\bar{v}_{G+1}$ of the current target individual $\bar{x}_G$ are generated:

$$v_i = \begin{cases} 1 & \text{if } rand \le P(x_i) \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

The genes of the trial individual $\bar{u}_{G+1}$ can be obtained by the crossover operator through Eq. (14):

$$u_i = \begin{cases} v_i & \text{if } rand \le CR \text{ or } i = irand(R) \\ x_i & \text{otherwise} \end{cases} \tag{14}$$

Therefore, at least one bit of the trial individual is inherited from the mutant individual so that MBDE is able to avoid duplication individuals and effectively search within the neighborhood [34]. Then, the procedure follows the traditional selection step.

5. RESULTS

5.1 The artificial case study

Without loss of generality, we present our analysis on the "Low" end state. From the truth-table of Tab. 3, we can identify all the C=13 minterms that make the system fail, listed in Tab. 8. These are the 13 columns m_j, j=1, 2, ..., 13, of the implicants chart A that have to be covered by the PIs we aim at identifying. The rows x_i, i.e., the complete set of implicants of the system structure function, of the implicant chart A are listed in Tab. 9.

	A	***B***	***C***	***D***	***E***
m_1	0	0	1	0	0
m_2	0	0	0	1	0
m_3	1	0	1	0	0
m_4	1	0	0	1	0
m_5	0	1	1	0	0
m_6	0	1	0	1	0
m_7	0	0	1	1	0
m_8	1	0	1	1	0
m_9	0	1	1	1	0
m_{10}	0	0	1	1	1
m_{11}	1	1	1	1	0
m_{12}	1	0	1	1	1
m_{13}	1	1	1	1	1

Tab 8. *List of the faulty minterms m_i of the system*

	A	***B***	***C***	***D***	***E***
x_1	0	0	1	0	0
x_2	0	0	0	1	0
x_3	1	0	1	0	0
x_4	1	0	0	1	0
x_5	0	1	1	0	0
x_6	0	1	0	1	0
x_7	0	0	1	1	0
x_8	1	0	1	1	0
x_9	0	1	1	1	0
x_{10}	0	0	1	1	1
x_{11}	1	1	1	1	0
x_{12}	1	0	1	1	1
x_{13}	1	1	1	1	1
x_{14}	-	0	1	0	0
x_{15}	0	-	1	0	0
x_{16}	0	0	1	-	0
x_{17}	-	0	0	1	0
x_{18}	0	-	0	1	0
x_{19}	0	0	-	1	0
x_{20}	1	0	1	-	0
x_{21}	1	0	-	1	0
x_{22}	0	1	1	-	0

x_{23}	0	1	-	1	0
x_{24}	-	0	1	1	0
x_{25}	0	-	1	1	0
x_{26}	0	0	1	1	-
x_{27}	1	-	1	1	0
x_{28}	1	0	1	1	-
x_{29}	-	1	1	1	0
x_{30}	0	1	1	1	-
x_{31}	-	0	1	1	1
x_{32}	0	-	1	1	1
x_{33}	-	0	1	-	0
x_{34}	0	-	1	-	0
x_{35}	-	0	-	1	0
x_{36}	0	-	-	1	0
x_{37}	-	-	1	1	0
x_{38}	-	0	1	1	-
x_{39}	0	-	1	1	-

Tab 9. *List of the implicants x_i of the system*

The optimal cover $\bar{x}_{opt}$ is the one for which the cost function Eq. (4) is minimized. Different approaches can be tailored for penalizing incomplete solutions (solutions that do not cover all faulty minterms), taking into account that assigning them a very high cost (for example the cost of all implicants) do not differentiate between extremely bad solutions (those who cover only a few minterms) and almost optimal ones (those that cover almost all minterms at a very low cost) [19].

In this work, we adopted two different cost functions for this, namely "Penalty" [19] and "One complement" [36]. The "Penalty" fitness function is the sum of the costs of the chosen implicants plus, in case the chosen implicants do not cover all the faulty minterms, an extra cost of αw_i, with $\alpha=1.25$, for each i-th implicant that should be added for a complete cover. So, when the chosen implicants do not cover all the faulty minterms, the function resorts to a sequential search starting at the first implicant and including all implicants needed to cover all the minterms. With the "One complement" fitness function, the cost of the trial solution is mapped into a binary fitness function made up by two parts: the most important digits are determined as the complement to one of the uncovered faulty minterms, while the least important digits are determined as the complement to one of the sum of the costs of the implicants included in the trial solution. In this way, we obtain that a complete subset of PIs that covers all faulty minterms has for sure a larger fitness than any other

incomplete subset. It is important to underline that with the "Penalty" fitness function we aim at minimizing the cost of Eq. (4), whereas with the "One complement" fitness function we aim at the maximization of the cost.

In this case study, the fitness value corresponding to the true optimal solution $\bar{x}_{opt}$ is equal to 21 when using the "Penalty" fitness function and to 4074 when using the "One Complement" fitness function. The true solution $\bar{x}_{opt}$ is found using the Quine-McCluskey algorithm that gives a deterministic way to check that the minimal form of a Boolean function has been reached [12]. This is a tabular method that compares each minterm with all the other minterms: if two of them differ in only one variable, that variable is removed and a reduced (merged) implicant is formed; the merging process is repeated for all the minterms until the cycle yields no further elimination of variables; the remaining implicants are thus selected as the PIs [7; 12]. Although more practical than Karnaugh maps when dealing with more than four variables, the Quine–McCluskey algorithm also has a limited range of use since the problem it solves is NP-hard: the runtime of the Quine–McCluskey algorithm grows exponentially with the number of variables. However, in this artificial case study, it is able to provide the optimal PIs $\bar{x}_{opt}$ as listed in Tab. 10, where each row represents one of the 7 PIs of this problem.

	State of component *A*	State of component *B*	State of component *C*	State of component *D*	State of component *E*
PI$_1$	-	B	$\bar{C}$	-	E
PI$_2$	A	-	$\bar{C}$	-	E
PI$_3$	-	B	-	$\bar{D}$	E
PI$_4$	A	-	-	$\bar{D}$	E
PI$_5$	-	-	$\bar{C}$	$\bar{D}$	E
PI$_6$	-	B	$\bar{C}$	$\bar{D}$	-
PI$_7$	A	-	$\bar{C}$	$\bar{D}$	-

Table 10. *Prime implicants set obtained analytically by Quine-McCluskey algorithm (component is failed* $(\bar{X})$*, working* (X) *or it is irrelevant (-) as contributor to the PI)*

It is worth mentioning that, if we would have been searching for traditional MCSs rather than PIs (like in [30]), the actual behavior of the system would not have been straightforwardly identified and the system could have been be exposed to (avoidable) risk states. For example, let us consider the PI$_1$ of Table 10 (where component *C* is failed, components *B* and *E* are working, and the states of components *A* and *D* do not influence the system end state). If component *B* (or *E*) is failed the system end state should remain "Failed", if we assume coherence of the system. On the contrary, due to the

non-coherence of the analyzed system, if component *E* fails and the state of component *B* does not change, the end state of the system is "Safe" (as shown by system configuration 15 in Table 3) rather than "Failed". Therefore, the analysis of the identified PIs would suggest that, in order to avoid system failure, component *E* could be forced to fail as a counteracting measure to component *C* failure; this conclusion could not be reached with a MCS analysis.

The results by MBDE and BDE with the different fitness functions "Penalty" and "One Complement", $\hat{\bar{x}}_{opt}$, are compared with respect to three performance indicators that aim at quantifying the goodness of the results, on a set of 20 trials of optimizations to account for the inherent stochasticity of the search, viz:

- Cpu: cpu time (expressed in seconds) necessary to converge to the solution $\hat{\bar{x}}_{opt}$.
- Success rate: percentage of trials for which the true optimum $\bar{x}_{opt}$ is found.
- Accuracy (λ): the larger λ, the larger the accuracy of the solution [37].

$$\text{if } \bar{x}_{opt} \neq 0 \;\; \lambda = \begin{cases} 0 & \text{if } \dfrac{\left|\hat{\bar{x}}_{opt} - \bar{x}_{opt}\right|}{\left|\bar{x}_{opt}\right|} \geq 1 \\ 11 & \text{if } \dfrac{\left|\hat{\bar{x}}_{opt} - \bar{x}_{opt}\right|}{\left|\bar{x}_{opt}\right|} < 10^{-11} \\ -\log_{10}\left(\dfrac{\left|\hat{\bar{x}}_{opt} - \bar{x}_{opt}\right|}{\left|\bar{x}_{opt}\right|}\right) & \text{otherwise} \end{cases}$$

$$\text{if } \bar{x}_{opt} = 0 \;\; \lambda = \begin{cases} 0 & \text{if } \left|\hat{\bar{x}}_{opt}\right| \geq 1 \\ 11 & \text{if } \left|\hat{\bar{x}}_{opt}\right| < 10^{-11} \\ -\log_{10}\left(\left|\hat{\bar{x}}_{opt}\right|\right) & \text{otherwise} \end{cases} \qquad (15)$$

5.1.1 MBDE Results

We solve the set covering problem (SCP) defined in Section 4 on the problem of Section 2 using an MBDE software developed by LASAR (Laboratorio di Analisi di Segnale e Analisi di Rischio) at the Politecnico di Milano (www.lasar.cesnef.polimi.it). Parameters *F* (see Eq. (6)) and *CR* (see Eq. (7)) are optimized through a trial and error procedure and to the values reported in Tab. 11, for the MBDE with "Penalty" and "One complement" fitness functions.

		Modified Binary Differential Evolution	
	Fitness Function	Penalty	One complement
Parameters	***F***	0.4	0.5
	CR	0.6	0.6

Table 11. *Values of the parameters F and CR used in the MBDE*

We perform the simulation for different population sizes (NP) (*NP=30, 100, 300* and *500*). Results are reported in Tab. 12, Tab. 13, Tab. 14 and Tab. 15, respectively. The only stopping criterion is the generation number, *MAXGEN*, equal to 500.

	Modified Binary Differential Evolution	
Fitness Function	Penalty	One complement
NP	30	30
Cpu [s]	9.07	4.69
Success rate	100 %	100 %
Accuracy	11	11

Tab. 12. *Performance indicators for the MBDE performed with NP=30*

	Modified Binary Differential Evolution	
Fitness Function	Penalty	One complement
NP	100	100
Cpu [s]	30.43	16.15
Success rate	100 %	100 %
Accuracy	11	11

Tab. 13. *Performance indicators for the MBDE performed with NP=100*

	Modified Binary Differential Evolution	
Fitness Function	Penalty	One complement
NP	300	300
Cpu [s]	99.66	53.95
Success rate	100 %	100 %
Accuracy	11	11

Tab. 14 *Performance indicators for the MBDE performed with NP=300*

	Modified Binary Differential Evolution	
Fitness Function	Penalty	One complement
NP	500	500
Cpu [s]	155.32	85.21
Success rate	100 %	100 %
Accuracy	11	11

Tab. 15. *Performance indicators for the MBDE performed with NP=500*

MBDE shows a success rate of 100% with both fitness functions, with very large accuracy (the solution found $\hat{\bar{x}}_{opt}$ is always equal to the true optimum solution $\bar{x}_{opt}$ and the relative error is always null) even when the population is composed by only 30 chromosomes. In general, the Cpu indicator shows that with the "Penalty" fitness function the algorithm is faster than with the "One complement" fitness function, mainly because of its more straightforward computation. Obviously, the Cpu indicator performance worsens when the number of chromosomes in the population becomes larger.

5.1.2 BDE and GA Results

For comparison, we solve the same set covering problem (SCP) using a BDE toolbox and a Genetic Algorithm (GA) toolbox taken from Mathwork's MATLAB® computational software. For both techniques, we implement the same fitness functions as in MBDE, use the same stopping criterion, repeat the simulations for the same population sizes as in MBDE and calculate the same performance indicators.

Parameters *F* and *CR* with "Penalty" and "One complement" fitness function for BDE were set equal to the values reported in Tab. 16, by trial and error.

		Binary Differential Evolution (BDE)	
	Fitness Function	Penalty	One complement
Parameters	***F***	0.7	0.7
	CR	0.1	0.1

Table 16. *Values of the parameters F and CR used in the BDE*

For the GA toolbox, the settings of those parameters whose meaning is the same as for DE are reported in Tab. 17, optimized by a trial and error procedure; details on other parameters to be set for

the use of GA is out of the scope of the comparison: the interested reader may consult [22] for further details.

		Genetic Algorithm	
	Fitness Function	Penalty	One complement
Parameters	***CR***	*0.01*	*0.01*
	MAXGEN	*500*	*500*

Table 17. *Relevant parameters set for the GA*

The results obtained are showed in Tab. 18, Tab. 19 Tab. 20 and Tab. 21.

	Binary Differential Evolution		Genetic Algorithm	
Fitness	Penalty	One complement	Penalty	One complement
NP	30	30	30	30
Cpu [s]	12.91	4.76	20.10	12.33
Success rate	25%	15%	0%	0%
Accuracy	3.71	4.64	0.99	3.23

Tab. 18. *Performance indicators for the BDE and GA performed with NP=30*

	Binary Differential Evolution		Genetic Algorithm	
Fitness	Penalty	One complement	Penalty	One complement
NP	100	100	100	100
Cpu [s]	37.06	16.11	47.11	27.52
Success rate	50%	45%	35%	35%
Accuracy	6.16	6.93	4.59	3.23

Tab. 19. *Performance indicators for the BDE and GA performed with NP=100*

	Binary Differential Evolution		Genetic Algorithm	
Fitness	Penalty	One complement	Penalty	One complement
NP	300	300	300	300
Cpu [s]	108.05	53.54	116.45	66.65
Success rate	95%	65%	100%	85%
Accuracy	10.51	8.4135	11	9.89

Tab. 20. *Performance indicators for the BDE and GA performed with NP=300*

	Binary Differential Evolution		Genetic Algorithm	
Fitness	Penalty	One complement	Penalty	One complement
NP	500	500	500	500
Cpu [s]	170.9818	93.7270	230.58	99.60
Success rate	100%	95%	100%	100%
Accuracy	11	10.6305	11	11

Tab. 21. *Performance indicators for the BDE and GA performed with NP=500*

With respect to MBDE, BDE and GA need a large population to obtain a good success rate (i.e., success rate ≥85% if *NP*=300 for BDE and GA (Table 20), whereas *NP*=30 for MBDE (Table 12)); indeed, the probability estimation operator embedded into the MBDE (Eq. (12)) can provide superior global searching ability and avoid the optimization getting trapped into a local optimum, because the BDE mutation mechanism has a higher probability of producing a bit of value 1 in the evolution process that restricts the search diversity of the optimum solution [38]. On the other hand, in MBDE at least one bit of the trial individual is inherited from the mutant individual, so that it is able to avoid duplication individuals and effectively search within the neighborhood [34].

The success rate is better for BDE compared to GA when the population considered is small (see Tables 18 and 19, *NP*=30,100, respectively), whereas GA becomes better as the population increases (see Tables 20 and 21, *NP*=300,500, respectively); Success rate for BDE and GA is comparable to that of MBDE only with a population of *NP*=500 (see Tables 21 and 12, respectively). Concerning the Cpu performance, BDE is better than GA (see 3rd row of Tables 18 to 21), whereas it is slightly worse when compared to MBDE (see 3rd row of Tables 18 to 21, left, in comparison with 3rd row Tables 12 to 15). Also in these cases, the Cpu shows a superior performance with the "Penalty" fitness function compared with the "One complement", and worsens when the number of chromosomes in the population becomes larger (see 3rd row, 2nd and 3rd column of Tables 18 to 21). These simulations underline the fact that for a smaller population BDE has a higher accuracy in terms of success rate and computational time, whereas when the population is increased GA outperforms BDE in terms of accuracy of the results. These differences are driven by the ability of DE to explore efficiently the search space, even with a small population thanks to its particular mutation phase [33; 34].

5.1.3 Confidence on the results

Compared to MBDE results, BDE and GA do not converge to the true solution $\overline{x}_{opt}$ for all the 20 trials (i.e., in Tables 12 to 15, even with *NP*=30, success rate for MBDE is equal to 100%, whereas Tables 18 to 21 highlight that BDE and GA need *NP*≥300 for achieving success rate equal to 100%).

In Fig. 10, the empirical probability mass functions (pmfs) of the $\hat{\bar{x}}_{opt}$ fitness values obtained by BDE (with population of 30, 100, 300 and 500 chromosomes) are plotted; in Fig. 11 those of the GA results are shown. These Figures allows comparing the confidence of the results provided by MBDE, BDE and GA: since MBDE allow for success rate equal to 100% for any *NP*, i.e., large confidence, its results correspond to a Dirac distribution with mass in $\bar{x}_{opt}$ (21 for "One complement" and 4074 for "Penalty"), whereas, due to their lower values of success rate, pmfs of the $\hat{\bar{x}}_{opt}$ obtained by BDE and GA are spread around $\bar{x}_{opt}$, i.e., smaller confidence.

In particular, Fig. 10 (left) and Fig. 11 (left) show the probability mass functions of the $\hat{\bar{x}}_{opt}$ fitness values when the algorithm is implemented with the "Penalty" fitness function; the right probability mass functions correspond to when the algorithm is implemented with the "One complement" fitness function. Moreover, it can be seen the sensitivity of the results provided by BDE and GA on the population size *NP* can be seen: the increase of the number of individuals in the population moves the mean fitness value of the population towards the fitness value of $\bar{x}_{opt}$, and the increase of the number of individuals in the population and the use of the "Penalty" function gives rise to distributions that are shrinked on the best fitness value, which makes the result more reliable.

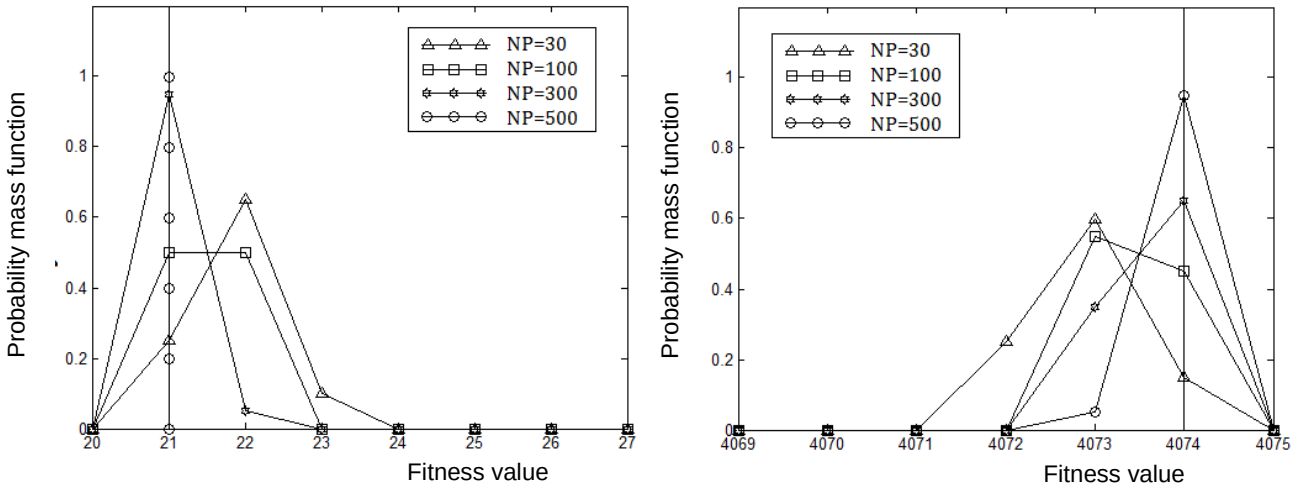

Fig. 10. *Pmfs of the $\hat{\bar{x}}_{opt}$ fitness values obtained with BDE, using the "Penalty" fitness function (left) and the "One complement" fitness function (right)*

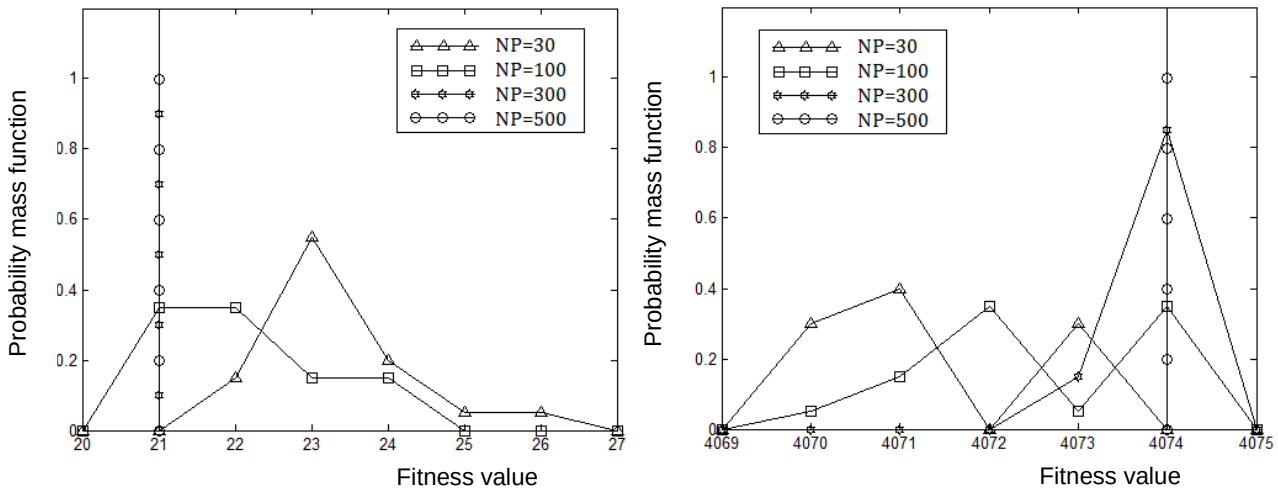

Fig. 11. *Pmfs of the* $\hat{\bar{x}}_{opt}$ *fitness values obtained with GA, using the "Penalty" fitness function (left) and the "One complement" fitness function (right)*

In all cases (MBDE, BDE and GA), the optimization algorithm may be challenged by the timing and order of the sequences of component failure events, and the number of system components. In the analytical case study, for example, the behavior of the system must be accurately modelled in order to be able to handle the set covering problem and, thus, to capture the influence of the timing and order of the sequences of component failure events on the determination of the PIs set, without reducing the DE searching capability. On the other hand, as the number of system components increases, the MBDE, BDE and GA methods can be challenged: in this case, an efficient and accurate PIs set determination can be achieved by a hierarchical method of a multi-steps DE optimization, as shown in [30]. Finally, if the system shows a large number of implicants (i.e., accident sequences), it might become necessary to prioritize the PIs search towards those accident sequences that are more meaningful with respect to the system end state of interest, instead of focusing on the whole implicants set, as done in [39], where authors present a visual interactive method for PI identification rather than resorting to the solution of a SCP.

5.2 The UTSG case study

From the truth-table of Tab. 4, we can identify all the C=9 minterms that make the system fail, listed in Tab. 22. These are the 9 columns m_j, j=1, 2, ..., 9, of the implicants chart A, that have to be covered by the PIs. The rows x_i, i.e. the complete set of implicants of our system structure function, of the implicant chart A are listed in Tab. 23.

We solve the SCP defined in Section 4 on the problem of Section 3 using an MBDE software whose parameters *F* (see Eq. (6)) and *CR* (see Eq. (7)) are optimized through a trial and error procedure and set to the values reported in Tab. 24, for the MBDE with "Penalty" and "One complement" fitness functions. In both cases, the application of the MBDE provides the list of PIs for the UTSG, as listed in Table 25. Results are confirmed by Quine–McCluskey algorithm.

Minterm	**Failure of the outlet steam valve**	**Failure of the safety relief valve**	**Level sensor- PID controller communication interruption**	**Failure of the PID controller**
$\boldsymbol{m}_1$	1	0	0	0
$\boldsymbol{m}_2$	0	0	0	1
$\boldsymbol{m}_3$	1	0	1	0
$\boldsymbol{m}_4$	1	0	0	1
$\boldsymbol{m}_5$	0	0	1	1
$\boldsymbol{m}_6$	1	1	1	0
$\boldsymbol{m}_7$	1	1	0	1
$\boldsymbol{m}_8$	1	0	1	1
$\boldsymbol{m}_9$	1	1	1	1

Tab 22. *List of the faulty minterms m_i of the system*

Implicant	**Failure of the outlet steam valve**	**Failure of the safety relief valve**	**Level sensor- PID controller communication interruption**	**Failure of the PID controller**
$\boldsymbol{x}_1$	1	0	0	0
$\boldsymbol{x}_2$	0	0	0	1
$\boldsymbol{x}_3$	1	0	1	0
$\boldsymbol{x}_4$	1	0	0	1
$\boldsymbol{x}_5$	0	0	1	1
$\boldsymbol{x}_6$	1	1	1	0
$\boldsymbol{x}_7$	1	1	0	1
$\boldsymbol{x}_8$	1	0	1	1
$\boldsymbol{x}_9$	1	1	1	1
$\boldsymbol{x}_{10}$	1	0	-	0
$\boldsymbol{x}_{11}$	1	0	0	-
$\boldsymbol{x}_{12}$	-	0	0	1
$\boldsymbol{x}_{13}$	0	0	-	1
$\boldsymbol{x}_{14}$	1	-	1	0
$\boldsymbol{x}_{15}$	1	0	1	-
$\boldsymbol{x}_{16}$	1	-	0	1
$\boldsymbol{x}_{17}$	1	0	-	1
$\boldsymbol{x}_{18}$	-	0	1	1
$\boldsymbol{x}_{19}$	1	1	1	-
$\boldsymbol{x}_{20}$	1	1	-	1
$\boldsymbol{x}_{21}$	1	-	1	1
$\boldsymbol{x}_{22}$	1	0	-	-
$\boldsymbol{x}_{23}$	-	0	-	1
$\boldsymbol{x}_{24}$	1	-	1	-
$\boldsymbol{x}_{25}$	1	-	-	1

Tab 23. *List of the implicants x_i of the system*

fitness function	Penalty	One complement
NP	30	30
MAXGEN	500	500
F	0.8	0.8
CR	0.3	0.3
CPU [s]	1.11	22.61
Success rate	100%	100%
Accuracy	11	11

Table 24. *Values of the parameters F and CR used and performance indicators*

Prime Implicant	Failure of the outlet steam valve	Failure of the safety relief valve	Level sensor- PID controller communication interruption	Failure of the PID controller
PI_1	$\overline{T}$	U	-	-
PI_2	-	U	-	$\overline{Z}$
PI_3	$\overline{T}$	-	$\overline{V}$	-
PI_4	$\overline{T}$	-	-	$\overline{Z}$

Table 25. *Prime implicants set (component is failed* $(\bar{X})$*, working* (X) *or it is irrelevant (-) as contributor to the PI)*

Again, it is worth noting that the non-coherence of the system, and the difference between MCSs and PIs can be pointed out by analyzing the PIs in Table 25. Indeed, for example, PI_1 of Table 25 shows that the outlet steam valve is failed ($\overline{T}$), the safety relief valve is working (U) and the states of Level sensor- PID controller communication and of the PID controller components are irrelevant to the end state of the steam generator. However, due to the non-coherence of the system, as soon as the steam valve fails, the safety relief valve could be forced to fail in order to have a safe end state of the steam generator (as shown by system configuration 16 in Table 4).

6 CONCLUSIONS

The reliability analysis of dynamic systems calls for the complementation of traditional PRA methods by dynamic reliability methods. For such systems, the sequence and timing of the events in a scenario is relevant and can give rise to non-coherent structure functions, in which failed and working states of the same components can lead the system to failure. Then, traditional minimal cut set analysis cannot be applied and prime implicants identification becomes the only way.

In this paper, the problem of prime implicants identification has been treated as an optimization problem aimed at finding the minimum combination of implicants that can guarantee the best coverage of all the minterms which fail the system. For this, we have developed a new technique to find PIs of a non-coherent structure function resorting to MBDE. The results have been compared with those obtained by BDE and GA.

It has been shown that MBDE has superior performances in terms of computational time and accuracy of the results (i.e., success rate for the convergence to the true solution) compared to BDE and GA, and performs very well even with a small population. Thanks to its more straightforward implementation, the "One complement" fitness function requires less time compared to the "Penalty" fitness function and gives a more robust PI identification, as verified by the success rate of the search results provided by BDE and GA. The ability of the method in PI identification has been confirmed with respect to a dynamic Steam Generator (SG) of a Nuclear Power Plant (NPP).

References

[1] Zio, E., Di Maio, F., Processing Dynamic Scenarios from a Reliability Analysis of a Nuclear Power Plant Digital Instrumentation and Control System, Annals of Nuclear Energy 36, 1386-1399, 2009.

[2] Aldemir, T., Guarro, S., Mandelli, D., Kirschenbaum, J., Mangan, L.A., Bucci, P., Yau, M., Ekici, E., Miller, D.W., Sun, X., Arndt, S.A., Probabilistic risk assessment modeling of digital instrumentation and control systems using two dynamic methodologies, Reliability Engineering and System Safety, 1011-1039, 2010.

[3] Di Maio, F., Vagnoli, M., Zio, E., Risk-based clustering for near misses identification in integrated deterministic and probabilistic safety analysis, Science and Technology of Nuclear Installations, 2015, art. no. 693891, 2015.

[4] Siu, N., Risk assessment for dynamic systems: an overview, Reliability Engineering and System Safety, 43, 43-73, 1994.

[5] Devooght, D., Dynamic reliability, Advances in Nuclear Science and Technology, 25, 215-278, 1997.

[6] Marseguerra, M., Zio, E., Devooght, J., Labeau, P.E., A concept paper on dynamic reliability via Monte Carlo simulation, Mathematics and Computers in Simulation, Volume 47, Issues 2–5, 1 August 1998, Pages 371–382.

[7] Quine, W.V., The problem of simplifying truth functions, Am. Math. Monthly, Volume 59, 521-531, 1952.

[8] Garrett, C., Apostolakis, G., Context in the risk assessment of digital systems, Risk Analysis, 19 (1), pp. 23-32, 1999.

[9] Beeson S.C., "Non-coherent fault tree analysis", Loughborough University UK.

[10] Sharvia, S., Papadopoulos, Non-coherent Modelling in Compositional Fault Tree Analysis, Proceedings of the 17th World Congress, The International Federation of Automatic Control, Seoul, Korea, July 6-11, 2008.

[11] Morreale, E., Partitioned List Algorithms for Prime Implicant Determination from Canonical forms, IEEE Transactions on Electronic Computers, Volume EC-16, No.5, 611-620, 1967.

[12] McCluskey, E.J.Jr., Minimization of Boolean functions, Bell Sys. Tech. J., Volume 35, 1417-1444, 1956.

[13] Karnaugh, M., The Map Method for Synthesis of Combinational Logic Circuits, Transactions of the American Institute for Electrical Engineers part I 72 (9): 593–599, 1953.

[14] Morreale, E., Recursive Operators for Prime Implicant and Irredundant Normal Form Determination, IEEE Transactions on Computers, Volume C-19, No.6, 504-509, 1970.

[15] Jung, W.S. , Han, S.H., Ha, J., A fast BDD algorithm for large coherent fault trees analysis, Reliability Engineering and System Safety, Volume 83, Issue 3, Pages 369-374, 2004.

[16] Worrell, R.B., Stack, D.W., Hulme, B.L., Prime implicant of non-coherent fault trees, IEEE Transactions on Reliability R-30/2, 98-100, 1981.

[17] Rauzy, A., Dutuit, Y., Exact and truncated computations of prime implicants of coherent and non-coherent fault tree, Reliability Engineering and System Safety, 58, 127-144, 1997.

[18] Bjorkman, K., Solving dynamic flowgraph methodology models using binary decision diagrams, Reliability Engineering and System Safety 111, 206–216, 2013.

[19] Sen, S., Minimal cost set covering using probabilistic methods, Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing: states of the art and practice, 157-164, 1993.

[20] Storn, R.; Price, K., Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization, 11: 341–359, 1996.

[21] Christofides, N., Paixão, J., Algorithms for large scale set covering problems, Annals of Operations Research, Volume 43, Issue 5, May 1993, Pages 259-277.

[22] Beasley, J.E., Chu, P.C., A genetic algorithm for the set covering problem, European Journal of Operational Research, vol.94, pp392-404, 1996.

[23] Aubry, J. F., Babykina, G., Barros, A., Brinzei, N., Deleuze, G., De Saporta, B., Dufour, F., Langeron, Y., Zhang, H., Project APPRODYN: APPROches de la fiabilité DYNamique pour modéliser des systèmes critiques, Technical report, collaboration CRAN, EDF R&D, INRIACQFD, UTT-ICD, 2012.

[24] Di Maio, F., Secchi, P., Vantini, S., Zio, E., Fuzzy C-Means Clustering of Signal Functional Principal Components for Post-Processing Dynamic Scenarios of a Nuclear Power Plant Digital Instrumentation and Control System, IEEE Transactions on Reliability, 415-425, 2011.

[25] Baraldi, P., Di Maio, F., Zio, E., Unsupervised Clustering for Fault Diagnosis in Nuclear Power Plant Components, International Journal of Computational Intelligence Systems, Vol. 6, No. 4, July 2013, pp. 764-777.

[26] Di Maio, F., Nicola, G., Zio, E., Yu, Y., Ensemble-based sensitivity analysis of a best estimate thermal hydraulic model: application to a Passive Containment Cooling System of an AP1000 Nuclear Power Plant, Annals of Nuclear Energy, 73, 200–210, 2014.

[27] Di Maio, F., Baronchelli, S., Zio, E., A Computational framework for Prime Implicants Identification in non-coherent Dynamic Systems, Risk Analysis, DOI: 10.1111/risa.12251.

[28] Assessment and management of ageing of major nuclear power plant component important to safety: Steam Generators, IAEA, IAEA-TECDOC-98, Vienna, ISSN 1011-4289, 1997.

[29] Kothare, M.V., Mettler, B., Morari, M., Bendotti, P., Falinower, C.-M., Level control in the steam generator of a nuclear power plant, IEEE Transactions on Control Systems Technology, 8 (1), pp. 55-69, 2000.

[30] Di Maio, F., Baronchelli, S., Zio, E., Hierarchical Differential Evolution for Minimal Cut Sets Identification: Application to Nuclear Safety Systems, European Journal of Operational Research, Volume 238, Issue 2, Pages 645–652, 2014.

[31] Belas, E., A class of location, distribution and scheduling problems: modeling and solution methods, in P. Gray and L. Yuanzhang (ed.), Proceeding of the Chinese-U.S. Symposium on System Analysis, J. Wiley and Sons.

[32] Holland, J.H., Adaptation in Natural and Artificial Systems" University of Michigan Press, Ann Arbor, 1975.

[33] Deng, C., Zhao, B., Yang, Y., Deng, A., Novel Binary Differential Evolution Algorithm for Discrete Optimization, Fifth International Conference on Natural Computation, Volume 4, 346-349, 2009.

[34] Wang, L., Fu, X., Menhas, M.I., A Modified Binary Differential Evolution Algorithm, Life Modelling and Intelligent Computing, Lecture Notes in Computer Science, Volume 6329/2010, 2010.

[35] Baraldi, P., Zio, E., Di Maio, F., Pappaglione, L., Chevalier, R., Seraoui, R., Differential Evolution for Optimal Grouping of Condition Monitoring Signals of Nuclear Components, Advances in Safety, Reliability and Risk Management, ESREL 2011, 410-418, 2011.

[36] Shackleford, B., Snider, G., Carter, R.J., Okushi, E., Yasuda, M., Seo, K., Yasuura, H., A High-Performance, Pipelined, FPGA-Based Genetic Algorithm Machine, Genetic Programming and Evolvable Machines, Volume 2, Number 1, 33-60, 2001.

[37] Tvrdìk, J., Competitive differential evolution, in MENDEL 2006, 12th International Conference on Soft Computing, 7-12, 2006.

[38] Wu, C.-Y., Tseng, K.-Y., Engineering optimization using modified binary differential evolution algorithm, 3rd International Joint Conference on Computational Sciences and Optimization, CSO 2010: Theoretical Development and Engineering Practice, 1, art. no. 5533094, pp. 501-505, 2010.

[39] Di Maio, F., Baronchelli, S., Zio, E., A Visual Interactive Method for Prime Implicants Identification, IEEE Transactions on Reliability, 64, Issue 2, 539-549, 2015.