IEEE Transactions on Reliability

# Development and implementation of a Direct Evaluation solution for Fault Tree Analyses competing with traditional Minimal Cut Sets methods

**J. Cortés[1], D. Di Martino[1], D. Duran[1*], J. López[2], J. Pons-Prats[3], J. Sánchez[4]**

*[1] DMD Solutions, Barcelona, Spain[1]*

*[2] University of Barcelona, Barcelona, Spain[2]*

*[3] International Centre for Numerical Methods in Engineering (CIMNE), Serra Húnter Fellow, Department of Physics - Aeronautics Division, Technical University of Catalonia, Castelldefels, Barcelona, Spain[3]*

*[4] Rutgers University-New Brunswick, New Jersey, USA[4]*

## Abstract

Fault Tree Analysis (FTA) is a well-established technique to analyze the safety risks of a system. Two specific prominent FTA methods, largely applied in the aerospace field, are the so-called Minimal Cut Sets (MCS), which uses an approximate evaluation of the problem, and the Direct Evaluation (DE) of the fault tree, which uses a top-down recursive algorithm. The first approach is only valid for small values of basic event probabilities and has historically yielded faster results than exact solutions for complex fault trees. The second one means exact solutions at a higher computational cost. The paper presents several improvements applied to both approaches in order to upgrade the computing performance. Firstly, improvements to the MCS approach have been performed, where the main idea has been to optimize the number of required permutations and to take advantage of the available information from previous solved subsets. Secondly, improvements to the DE approach have been applied, which deal with a reduction of the number of recursive calls through a deep search for independent events in the fault tree. This could dramatically reduce the computation time for industrial fault trees with a high number of repeated events. Additional implementation improvements have been also applied regarding hash tables, and memory access and usage, but also implementing the so-called "virtual gates", which enable limitless children on each gate. The results presented hereafter are promising, not only because they show that both approaches have been highly optimized compared to the literature, but also because a Direct Evaluation solution has been achieved, which can compete in time resources (and obviously in precision) with the MCS approach. These improvements are relevant when considering the industrial, and more specifically the aeronautical, implementation and application of both techniques.

***Key Words*** *– Fault tree, Minimal Cut Sets, Direct Evaluation, RAMS, independence, permutations, Robin RAMS™, Safety*

## Glossary

**Monte-Carlo –** Method relying on repeated random sampling to obtain numerical results

**Leaves –** Set of basic events of a fault tree, which does not consider repetitions (distinct events)

**Order of Cut Sets –** Number of basic events that conform a Cut Set (i.e. if a Cut Set is formed by two basic events, its order is 2)

* Corresponding author; davidduran@dmd.solutions

[1] Municipi, 14, 08018 Barcelona

[2] Gran Via de les Corts Catalnes, 585, 08007 Barcelona

[3] Esteve Terradas, 7, 08860 Castelldefels

[4] 57 US Highway 1, New Brunswick, NJ 08901-8554

**Disjoint event** – Event, within a set, which does not contain any child in common with the other events conforming the set

### Acronyms

**COTS** – Commercial Off-The-Shelf
**DE** – Direct Evaluation
**DFS** – Depth-First Search
**FTA** – Fault Tree Analysis
**FRACAS** – Failure Reporting, Analysis and Corrective Action System
**MCS** – Minimal Cut Set
**RAMS** – Reliability, Availability, Maintainability, and Safety
**ZBDD** – Zero-suppressed Binary Decision Diagrams

### Nomenclature

$n_{perm}$ – Number of permutations
$N$ – Number of leaves
$f$ – Sample factor
$L$ – Cut Set Cutoff
$V$ – Total number of nodes of the fault tree
$E$ – Total number of connections within the fault tree
$R^2$ – R-squared
$S$ – Set of events
$Q$ – Unavailability
$\{n\}$ – Node of interest
$\{j, k\}$ – Children of node $\{n\}$
$\lambda$ – Failure rate of basic event
$t$ – Time period

## 1. Introduction

Fault Tree Analysis (FTA) is a well-established risk assessment methodology, extensively used in the aeronautical engineering field [1][2][3], as well as other applications like infections disease spread, oil and gas, and maritime engineering [4][5][6][7][8][9][10]. Among other classifications [11], the paper focuses on the qualitative and quantitative techniques to evaluate FTA. While qualitative techniques are used to detect system vulnerabilities (e.g., minimal cut sets), quantitative techniques compute numerical values over a fault tree, such as the probability of a top event based on a combination of events happening at a lower-level. The top event is usually an undesired event (such as a system failure) to be avoided for a safer and more reliable system design.

The contribution on this manuscript can be summarized as (1) the improvement of DE method regarding the reduction of the recursive calls through the identification of independent events, (2) the development of the so-called "Virtual Gates", which enable limitless children on each gate, and finally (3) several computational improvements leading to an upgraded computational efficiency (less computational time and higher accuracy); including the reduction of the required permutations of MCS methods and the use of previously computed data, the use of hash tables, and improved access and usage of memory.

Reliability analyses have been large analyzed, and one can find both comprehensive reviews of general-scope methods, as well as, application-oriented reviews [9][12]. The scope of this study is limited to standard fault trees, which were introduced in the 1960s and are most used in several systems of the aviation field (i.e. landing gear, electrical system, flight controls...). These types of fault tree are composed of classical node types with a different function each, such as OR gates, AND gates or basic events (see [13] for further details on the gate types). Other types of FTAs, such as dynamic fault trees, which consider failure of components in a different order [14][15][16][11], fuzzy FTAs [17][18][4][5][6][7][10], or the hybridization of FTA with other methods [19] are outside the scope of the paper. Generally speaking, researches agree with the limitations of the bottom-up and top-down approaches to evaluate a fault tree, as stated in [20]. Limitations that are related to the computational cost, the accuracy, the ability to compute rare events probability [21], and the limited capacity of improvement due to the maturity of the standard approaches and methods. Several methods have been developed to overcome these limitations [22][23]. Among the methods that can be used to

quantitatively compute a fault tree, the authors want to highlight bottom-up analysis, binary-decision diagrams [23][20], rare event approximation [3][21] (usually, based on MCS) or Bayesian network analysis [15][24]. Although other ones, like Petri Nets, Pivotal Decomposition [25] or Markov methods [26], are also relevant, but out of the scope of the present work. Two of the quantitative analyses of a fault tree are specially used in the aviation sector, the main aim of the present communication: Minimal Cut Sets (MCS) [27] and Direct numerical Evaluation (DE) without computing the cut sets. Although the MCS approach produces an approximate result [28], it is widely used due to its good performance, the ability to truncate the number of cut sets as required, and because it is not necessary to input all the basic events. However, the Direct Evaluation of fault trees solves the tree exactly. A comparison of the main advantages and drawbacks of each type of analysis can be observed in ***Table 1***.

| Minimal Cut Sets | Direct Evaluation |
|---|---|
| - Approximate solution | - Exact solution |
| - Produce cut sets | - Does not produce cut sets |
| - Good computational performance | - Not so good computational performance |
| - Do not require input for each component | - Requires input for each component |
| - A change of probabilities does not require a complete rerun | - A change of probabilities requires a complete rerun |

**Table 1 - Comparison of MCS vs DE approach**

In this study, both approaches (direct and MCS evaluation) are presented and compared with published algorithms from the literature. A comparison between these approaches is also provided highlighting the points of interest.

The aim of the research is to show that it is possible to implement a direct evaluation of a fault tree which can compete in performance with the MCS methods in standard aeronautical fault trees. This means getting the same level of accuracy as a Direct Evaluation technique at the same cost as MCS. The aforementioned goal has been achieved by a thorough search of independence in the fault tree and making use of hash tables and dynamic memory. Moreover, our solution is applicable to any standard fault tree, regardless of the number of children per gate.

Another aim of this document is to show several degrees of optimization on both approaches, which, in the case of the minimal cut sets, can be achieved by using only the necessary permutations and by making use of previous computed subsets of cut sets.

To generate the diagrams of the fault trees, introduce and collect all the data, and perform all the calculations, the innovative software Robin RAMS™, developed by DMD Solutions, has been used throughout the whole study. Specifically, for drawing and building the fault trees, a recursive node-positioning algorithm has been implemented according to the method explained by John Walker [29].

Robin RAMS™ is a Reliability, Availability, Maintainability, Safety (RAMS) web-based software suite capable of performing reliability prediction analyses, fault tree analyses, and other RAMS-related modules. The solutions explained hereafter have been implemented in the aforementioned software suite, in order to be able to perform all the required calculations.

## 2. Minimal Cut Sets Methods

The Minimal Cut Sets are the smallest groups of basic events which, combined, cause the top event to occur.

The method of Minimal Cut Sets is extensively used and accepted by many industries as it provides an approximation that can be tailored to obtain a result suitable for certification substantiation, which usually involves usage of small probabilities (order of magnitudes smaller

than 0.1). For applications where higher probabilities are used, MCS is not a suitable solution.

Although the MCS enables the analytical computing of Fault Tree Analysis, a maximum order of cut sets has to be set to improve the performance of the computer-aided calculation. A cut set cutoff allows the higher-order cut sets, which have a small contribution to the total probability (e.g. a few orders of magnitude less than $10^{-9}$), to be ignored as they would not have a relevant effect on the overall probability calculation.

### 2.1. Developed Methods

With the aim of achieving better performance, four different methods involving minimal cut set combinations have been tested and evaluated for various exemplary fault tree computer-aided computations.

The minimal cut sets are constructed based on the built fault tree, and then the program finds its minimal cut sets by a modified approach as explained by Rosenberg [30] and Vatn [31]. Specifically, the computation of minimal cut sets has been implemented in the current study in two different phases:

1. Boolean/binary values are assigned to all the existing leaves (basic nodes)
2. The top event and intermediate gates boolean values are computed based on the assigned basic events

Once the minimal cut sets are found, the top event probability $Q_T$ can be approximated with the so-called "rare event approximation" [13] and assuming that two or more minimal cut sets cannot occur simultaneously. The so-called "rare event approximation" is accurate within about 10% of the exact unavailability when $Q_i$<0.1 (low value). Furthermore, any error lays on the conservative side:

$$Q_T \cong \sum_i Q_i \quad (1)$$

where $Q_i$ denotes the unavailability of each considered cut set, which can be expressed as the product of all the basic events conforming that cut set:

$$Q_i = \prod_k Q_k \quad (2)$$

The unavailability of each basic event $Q_k$ can be obtained as follows:

$$Q_k = \int_0^t \lambda(t) e^{-\lambda(t)t}\, dt \quad (3)$$

Following the Constant Failure Rate per Hour Model [13], the previous equation is converted to:

$$Q_k = 1 - e^{-\lambda t} \quad (4)$$

being $\lambda$ the constant failure rate of the basic event and $t$ the time period. Actually, this equation can be further simplified if $\lambda$ tends to 0 by using Taylor series:

$$Q_k \approx \lim_{\lambda \to 0}\left(1 - e^{-\lambda t}\right) = \lambda t - \frac{\lambda^2 t^2}{2} + O(\lambda^3) \approx \lambda t \quad (5)$$

The differences between the methods presented hereafter lay in the first step. Specifically, based on the nomenclature of reference [32], the first method can be defined as a Monte Carlo program, while the others can be categorized as combinatorial algorithms. The second phase of the algorithm is mutual through all the methods and has been implemented using the Depth-First Search (DFS) algorithm [33].

Method 1: Monte-Carlo approach

This method makes use of the Monte-Carlo approach to randomly assign 0s and 1s to the basic events. Therefore, repeated random sampling of basic events are applied to obtain the top event. In order to ensure that all the possible combinations have been tested (thus the desired result is randomly obtained), enough samples (big sample pool) are required. Thus, the number of tested permutations $n_{perm}$ is:

$$n_{perm}(N) = 2^{N+f} \quad (6)$$

where $N$ is the number of leaves in the Fault Tree and $f$ is just a factor that makes the pool sample big enough to account for random repetitions (usually greater than 3).

Method 2: All possible permutations

It involves calculating all the possible permutations leading to the top event and then calculating the boolean value of the top event. The permutations formula is:

$$n_{perm}(N) = 2^N \quad (7)$$

For instance, if $N$ were 2, the possible permutations would be 00, 01, 10, 11. If $N$ were 3, permutations would be 000, 001, 010, 011, 100, 101, 110, 111.

Method 3: Cut-down permutations

This method applies the same philosophy of the previous one, but the permutations where there are more than $L$ True basic events are not computed ($L$ being the Cut Set Cutoff). The equation of the number of possible permutations that ensures the combinations to be cut down is given by:

$$n_{perm}(N) = \sum_{i=0}^{L} \frac{N!}{i!\,(N-i)!} \quad (8)$$

Being $N$ the number of leaves, the program will not calculate a cut set composed of more than $L$ -True basic events. The number of leaves $N$ has to be greater than the Cut Set Cutoff $L$ by at least 1. For instance, if $N$ were 3 and $L$ were 2, the cut random combination set would be 000, 001, 010, 011, 100, 101, 110, ~~111~~, noting that the 111 element is not included in the calculation because is larger than the $L$ of true desired events.

Note that the combinations can be cut-down only because high-order cut sets have negligible contribution to the total probability, since its probability is given by the product of all the events forming the cut set. Moreover, the exact value of $L$ depends on the type and shape of the fault tree; it is different to always have only two children per gate than having around 10 children per gate.

Method 4: Improved cut-down permutations

This method is essentially an improved version of Method 3 as it also uses the same cut-combination Equation (8), but it improves the way the data is stored. In this method, the code makes set combinations that test all possibilities; however, as it is doing that, if there is a combination of events that would make a certain gate always true or that would lead to the top event, then it would store it into a subset and it would make sure that it does not test the combinations containing this subset.

For example, if there was a fault tree that had different basic events, where Basic Event 1 and Event 3 are minimal cut sets of order 1, when the program tried to calculate a different set containing either one of these events (i.e. [Basic 1, Basic 2] combination), this method would not calculate it, since it surely produces a true top event. This reduces the time the computer spends to obtain the top event probability.

In the worst-case scenario, the performance and complexity of this method is the same as Method 3.

### 2.2. Algorithm performances

As explained above, the algorithm can be basically divided into two phases: one to compute the possible permutations and another to evaluate the top event based on these permutations (using the DFS algorithm). When the number of leaves $N$ is big compared to the cutoff $L$ (and Method 2 always lays in this assumption since there is no cut set cutoff), the total time complexity of the implemented code can be approximated to:

$$O(perm + DFS) \quad (9)$$

where the cost of recursively evaluating the top gate with the DFS algorithm is:

$$O(DFS) = O\left(n_{perm} \cdot V \cdot E\right) \quad (10)$$

where $V$ is the total number of nodes of the fault tree and $E$ is the total number of connections within the fault tree. The cost of the DFS algorithm is as stated

above since, for each pre-computed permutation, the algorithm needs to calculate the binary value of each node of the fault tree based on its children. In the case of standard fault trees, it is always verified that $E = V - 1$, leading to:

$$O(DFS) \approx O\left(n_{perm} \cdot V^2\right) \quad (11)$$

On the other hand, the cost of computing all the permutations depends on each of the methods. For instance, for Method 3 (and the worst-case of Method 4), its complexity is:

$$O(perm) = O\left(\frac{N!}{L!\,(N-L)!}\right) \quad (12)$$

For Method 1 and 2, respectively, the cost of computing the permutations is:

$$O(perm) = O\left(2^{N+f}\right) \quad (13)$$

$$O(perm) = O(2^N) \quad (14)$$

Since the permutations are also implicitly included in the cost of the DFS, it can be demonstrated that $O(DFS) > O(perm)$.

The theoretical time performance of the MCS methods for different number of leaves can be observed in ***Figure 1***, where it is forecasted that the best algorithm complexity is to be achieved with either Method 3 or 4. Indeed, Method 1 is impractical to perform for more than 25 leaves in a usual personal computer due to its time performance.

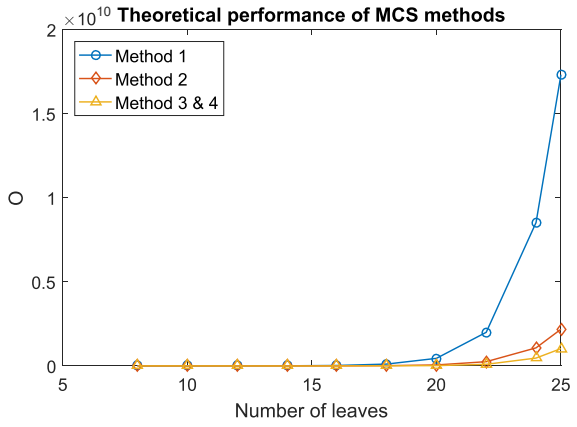


**Figure 1 - Theoretical performance of MCS methods**

Regarding the space complexity, the four methods make use of the same types of variables (linear at most) to store the necessary intermediate values. The only difference lays on the size of some of these variables: Method 1 has the biggest size, since much more combinations are stored. Method 2 has a lower number of permutations, but still bigger than Method 3 or 4. These latest methods use the same space complexity, but the time complexity is slightly better for Method 4.

As it has been shown, the time and space complexity depends on the type of implemented MCS method. The next steps of our research regarding Minimal Cut Sets are to be devoted to compare Method 4 with other types of MCS algorithms, such as ZBDD [34] or other COTS codes, where a different complexity can be achieved.

## 3. Direct Analysis Evaluation

As commented above, the Minimal Cut Sets methods are widely used due to its performance advantages and not needing the data of all the basic events. However, an exact solution can only be achieved using the direct analysis evaluation, which involves creating a top-down recursive algorithm that directly analyzes the exact probability of the desired event, as shown by Patterson-Hine and Koen [28], not considering minimal cut sets or maximum order of cut sets.

### 3.1. The Top-Down Recursive Algorithm

Three recursive algorithms which analytically calculate the top event probability are explained by Page and Perry in [35]. In this study, the most advanced one, which leads to less recursive calls, has been implemented and improved. For usual realistic problems (less than 30 leaves), these types of algorithms are adequate. However, for greater fault trees, they are really time consuming and the minimal cut sets methods are preferred.

The aim of this study has been to enhance the performance of Direct Evaluation to get a similar performance as MCS. To achieve this purpose, among others, modern object-oriented techniques,

hash tables, and database queries optimization have been used. However, the foundation of this enhancement is described in the flowchart below:

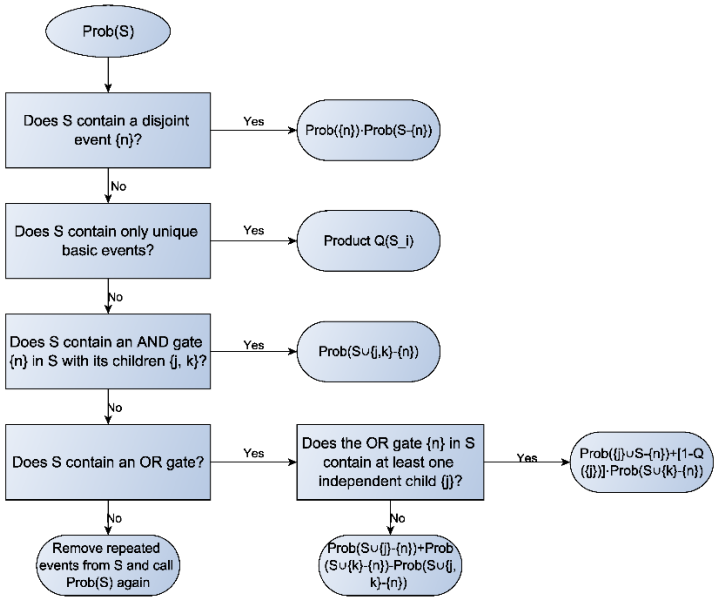


**Figure 2 - Flowchart for Top-Down Recursive Algorithm**

The algorithm usually starts with a set $S$ containing the top event and, throughout the execution of the program, this set evolves, containing the intersection of basic events and gates. The function $Prob(S)$ is recursively called and executes the algorithm detailed in ***Figure 2***, whose steps are explained hereafter.

1. If there is a disjoint event/gate within the set $S$, then the process can be simplified with the following formula (removing the disjoint event from the set):

$$Prob(S) = Prob(\{n\}) \cdot Prob(S - \{n\}) \quad (15)$$

   A disjoint event is a gate which does not have any event below itself in common with the events below any other gate in the set $S$. The gain in recursions when using Equation (15) is really important: for the fault tree displayed in ***Figure 6***, it passes from 2757 recursive calls of function $Prob(S)$ to only 164 calls.

2. If all the events contained in set $S$ are unique basic events, then the function returns the product of their unavailability $Q$:

$$Prob(S) = \prod_i Q(S_i) \quad (16)$$

3. Otherwise, if $S$ contains an AND gate, this gate is expanded (removing itself from the set and adding its children to the set $S$) and the function $Prob(S)$ is called again.

$$Prob(S) = Prob(S \cup \{j,k\} - \{n\}) \quad (17)$$

4. Otherwise, if $S$ contains an OR gate and, at least, one independent child $\{j\}$, the following recursion can be used:

$$Prob(S) = Prob(\{j\} \cup S - \{n\}) + [1 - Q(\{j\})] \cdot Prob(S \cup \{k\} - \{n\}) \quad (18)$$

   being $\{k\}$ the dependent child. An independent child $\{j\}$ is one child whose leaves (distinct basic events below) are disjoint from the leaves of the other children $\{k\}$ of the OR gate and the leaves of the events conforming $S$. This additional search for independence is more computing demanding than Equation (15), but, for most fault trees, it is faster than not doing it (more recursive calls are required). An idea to adapt this search depending on the fault tree is to limit the search for independence to the grandchildren to avoid too much depth. However, as Page and Perry establish [35], there is no "correct" way of searching for independence, as it depends on the nature of the tree.

5. Otherwise, if $S$ contains an OR gate and there is no independent child, $Prob(S)$ is defined by:

$$Prob(S) = Prob(S \cup \{j\} - \{n\}) + Prob(S \cup \{k\} - \{n\}) - Prob(S \cup \{j,k\} - \{n\}) \quad (19)$$

   where both $\{j,k\}$ are dependent children.

6. Finally, if the code has not entered in any of the previous steps, it means that all elements are basic events and there is a dependency within them (repeated basic event). Hence,

the repeated basic event is removed from $S$ and the function is called again.

This algorithm can be better understood on a simple fault tree, such as the one displayed in ***Figure 3***.

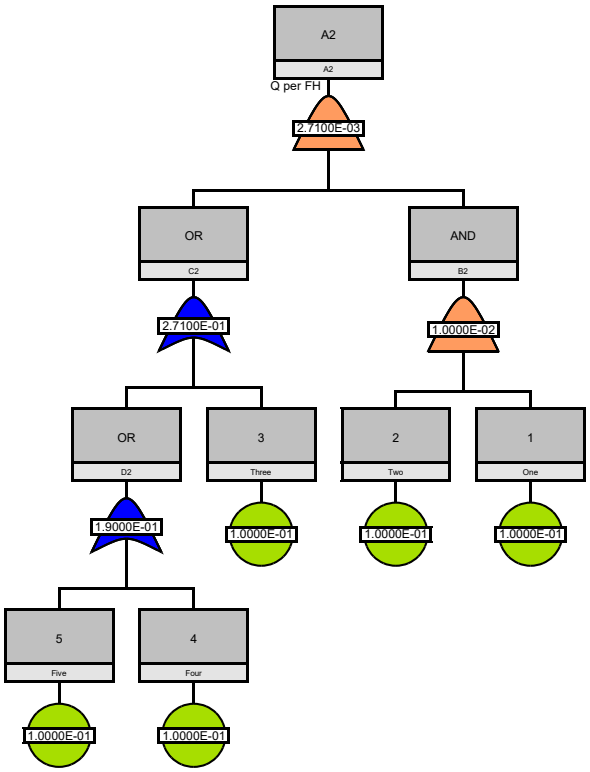


**Figure 3 - Simple FTA to show the algorithm**

On that simple fault tree, the recursive calls would be as depicted in ***Figure 4***, where the used step of the aforementioned algorithm is also detailed for each call. In this case, there is no basic event repeated, otherwise step 5 would have been also used.

Hence, the final formula solving the fault tree is:

$$\begin{aligned} Prob(S) = \; & Prob(\{One, Two, Three\}) \\ & + [1 - Q(\{Three\})] \\ & \cdot [Prob(\{One, Two, Four\}) \\ & + [1 - Q(\{Four\})] \\ & \cdot Prob(\{One, Two, Five\})] \end{aligned} \tag{20}$$

$$\begin{aligned} Prob(S) &= 0.1^3 + 0.9 \cdot [0.1^3 + 0.9 \cdot 0.1^3] \\ &= 0.00271 \end{aligned} \tag{21}$$

Step 6 of the algorithm would have been also used if there was an AND gate that expands itself into all of its children and then checks for independence. When the children are all expanded out, not all basic events would be independent, but they would not be neither OR nor AND gates. In that case, the algorithm would open all the gates and take into account the repeated events. Then, it would leave only distinct basic events in the set $S$ and proceed to calculate the probability of $S$ as usual.

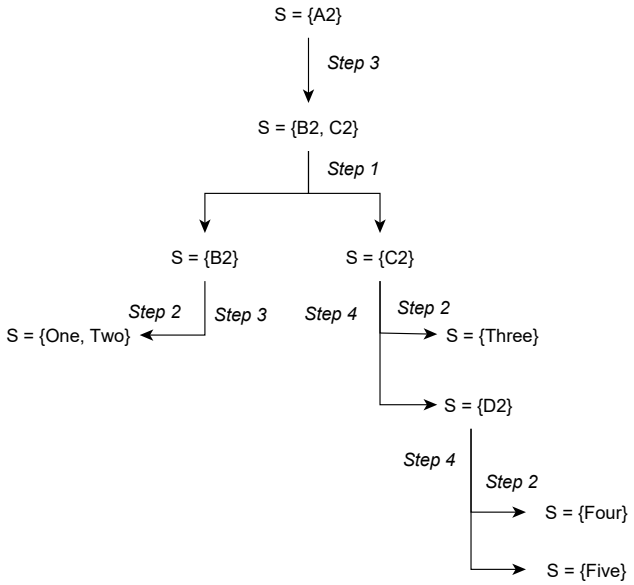


**Figure 4 – Algorithm flow of set *S* to compute the fault tree depicted in *Figure 3***

## 3.2.Direct Evaluation performance

The complexity of the Direct Evaluation depends on the selected steps of ***Figure 2***. The worst-case scenario, corresponding to a deep search of independence and disjoint events, is:

$$O(S^2) \cdot \#_{recursions}$$

The best-case scenario is when the set is already cached (performance of a hash table):

$$O(1)$$

Compared to the complexity of MCS methods, it can be clearly seen that it is faster than method 1 and 2, which follow an exponential law (this is a cubic law). Being faster than method 3 and 4 depends on the number of necessary recursions, which can be

dramatically decreased by searching for independence. However, note that an independence search implies also a high computation cost.

### 3.3.Novel features and improvements

Although the analytical algorithm has been initially adapted from existing solutions (see [28], [35] and [36]), several novelties and improvements have been implemented. They are described hereafter:

Search for independence

As explained in the previous steps, one of the focuses of the algorithm has been devoted to find independence through using different types of search (see step 1 and 4 of the explained pseudocode). This search allows to dramatically reduce the number of necessary recursions.

For traditional fault trees applied in the aeronautical field, where usually considerable events are repeated through these fault trees, deeply searching for independence can drastically reduce the total computation time. This search is a key differentiate feature compared to former FTA models.

Multiple children

The ability to compute gates with more than two (infinite) children has been implemented. This is a special case as most papers concerning the direct-analysis method only describe cases for nodes of only 2 children each.

For the AND gates, there is no problem on having more than two children, since the algorithm also works for multiple children: it expands them, checks for repetitions and make the product of them. However, the algorithms described in the aforementioned papers only work for OR gates with a pair of children.

Our solution to this problem has been that the algorithm virtually creates "virtual" OR gates, as shown in ***Figure 5*** so that an equivalent fault tree with at most two children per event is used. The two fault trees of ***Figure 5*** are mathematically equivalent (yielding the same top probability), but the second one can be easily computed with the implemented algorithm. Then, after performing the calculation, the program deletes these virtual OR gates and displays the original fault tree.

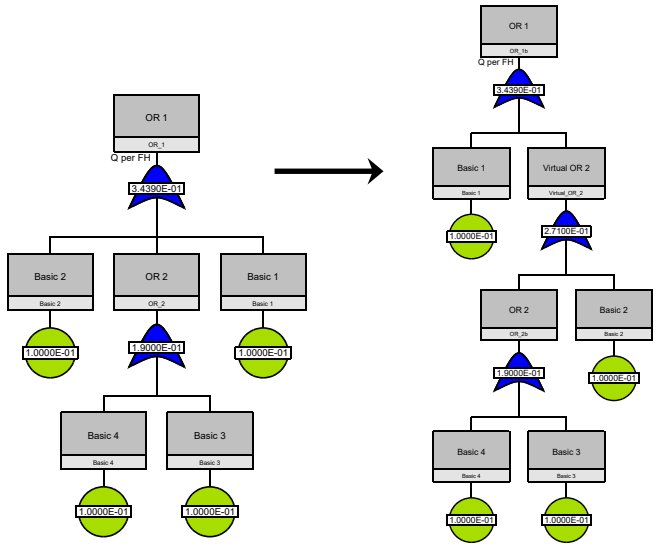


**Figure 5 – Fault tree transformation into virtual events**

Technical improvements: hash tables & database and memory optimization

Another novelty of the present algorithm in the FTA area is the usage of hash tables. These kind of tables or dictionaries are used on many applications (i.e. by chess engines for evaluating each position [37]) to save previous computations and associate a unique key to each result. Therefore, before starting each recursive call, the algorithm checks if it has previously called the function with the same inputs (same nodes with the same unavailability).

Using hash tables on big fault trees can be quite useful, since some parts of the tree are called more than once to compute the top probability. In fact, for the fault tree used to validate all the implemented algorithms in this document (see ***Figure 6***), the number of recursive calls has been reduced by a 37.6% when using hash tables.

Finally, a drastic reduction (close to 90%) of the necessary time to execute the algorithm has been achieved by making use of computer memory instead of database queries. In few words, the fault tree is saved in a dynamic object before starting its

computation and this object is updated and used to make all the necessary computations.

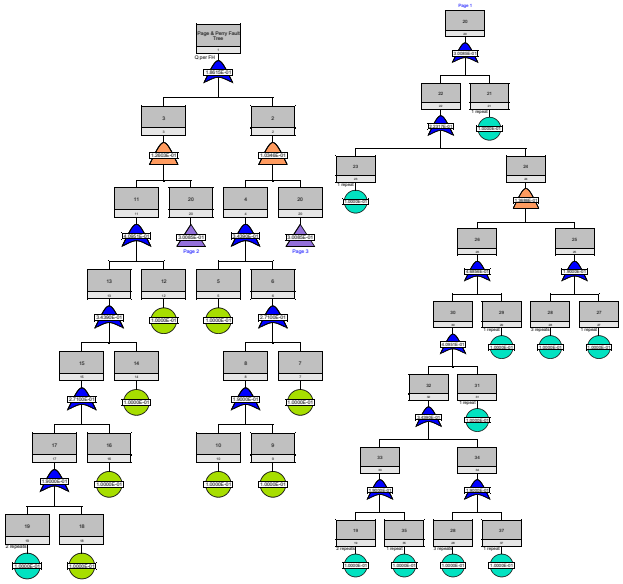

**Figure 6 – Validation reference fault tree** [35]

## 4. Results

Hereafter, the main results of both the MCS methods and the Direct Evaluation algorithm are presented. Firstly, the environment and operating conditions under which the tests have been performed are provided. Then, all methods are compared using the same typology of fault tree. Finally, the DE is compared with the literature and the MCS methods using complex and exemplary fault trees.

### 4.1. Environment and operating conditions

The results presented hereafter have been obtained after performing, at least, four different trials per test case under the same operating conditions to minimize any possible discrepancy. Indeed, the following environment has been used to compute all the fault trees of this study

- Hewlett Packard Enterprise ML30 Gen10
- Linux Ubuntu 18.04 LTS
- Four Intel Xeon processors @ 3.5 GHz (4.5 GHz turbo)
- 16 GB of RAM

### 4.2. Influence of number of leaves on performance

After several trials verifying that all the MCS methods and Direct Evaluation yielded the same top event probability, the four MCS methods and the Direct Evaluation have been tested on various exemplary fault trees with different numbers of leaves, ranging from eight to thirty. All the tested fault trees, whose characteristics can be observed in ***Table 2***, follow the same gate setup and probability values. Method 3 and 4 have been run with a cutoff of $L = 5$.

| ID | *N* | *V* | *E* |
|---|---|---|---|
| FTA_1 | 8 | 16 | 15 |
| FTA_2 | 10 | 18 | 17 |
| FTA_3 | 12 | 20 | 19 |
| FTA_4 | 14 | 22 | 20 |
| FTA_5 | 16 | 24 | 23 |
| FTA_6 | 18 | 26 | 25 |
| FTA_7 | 20 | 27 | 26 |
| FTA_8 | 22 | 30 | 29 |
| FTA_9 | 24 | 32 | 31 |
| FTA_10 | 25 | 33 | 31 |
| FTA_11 | 30 | 38 | 37 |

**Table 2 - Test Cases for the MCS methods study**

Using the environment described in section 4.1, the average time response for different number of leaves has been plotted in ***Figure 7***, where a third-degree polynomial curve in dashed line has been also fitted for each method.

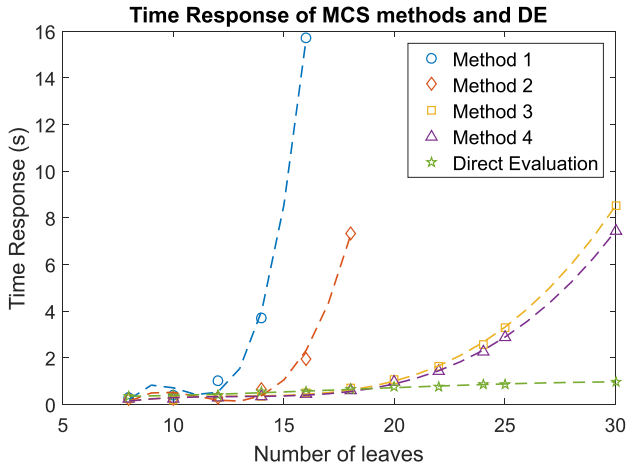


**Figure 7 - Time response of MCS methods and DE**

Method 1 has been only sampled for up to 16 leaves while Method 2 has been run for up to 18

leaves. In this way, not only do we limit the computational cost, but also all methods can be easily compared in a single plot. Method 3 and 4 have been sampled for up to 30 leaves (e.g. FTA_11).

It is worth noting that the plotted third-degree polynomial curves fit correctly ($R^2$ is always above 0.98) to the scattered average times.

These third-degree polynomial curves are useful to predict how much time it would take for significantly larger fault trees to be analyzed. Comparing methods 1 and 2 (see ***Figure 7***), it is clear that their behaviors are similar. However, since the growth is exponential, the difference between these methods increases with the number of leaves.

On the other hand, methods 3 and 4, which have been both sampled for all the fault trees made for this study, have a much flatterer curve. Evidently, these two methods have a very close correlation with each other, Method 4 being 10%-15% faster on average.

As it can be observed, ***Figure 7*** (real data) follows quite well the theoretical trends predicted in ***Figure 1*** for the four different MCS methods.

The best evolution of performance in time, however, lays on the Direct Evaluation as it can be observed in ***Figure 7***, where a more linear growth can be deduced. This agrees with the complexity formula depicted in section 3.2, which highlights nearly a quadratic dependency with the set size if a good independence search is performed. Specifically, the time response of the Direct Evaluation is better than Method 4 for fault trees with more than 18 leaves, while Method 4 behaves better for small number of leaves.

***Figure 7*** shows that, although all methods generate the same probability result, the time taken to produce the answer differs, clearly marking the eminence by MCS methods 3 and 4, and specially by Direct Evaluation. This is a major achievement, since, historically, Direct Evaluation techniques have yielded a worse time response than Minimal Cut Sets methods.

The observed time response trend of Methods 3 & 4 and Direct Evaluation is expected to be followed for fault trees of a bigger size. However, the different layouts (and dependencies) of the fault tree may affect the execution time and should be analyzed in a future study.

### 4.3. Accuracy and performance of Direct Evaluation algorithm

In order to verify the probability results given by the implemented analytic method, several fault trees from the literature (see [28], [35], [36]) have been used with positive result. Specifically, five different fault trees have been computed to present the following tables: the trees displayed in ***Figure 3*** and ***Figure 6***, Fig. 4 of reference [35] and Fig. 5 and 6 of reference [28]. The main characteristics of these fault trees, referred as FTA-1 to FTA-5 from now on, can be observed in ***Table 3***. It is worth mentioning that all these fault trees are coherent according to [38], which are the usual type of fault trees used in the RAMS field.

| FTA ID | Reference | # Leaves | # Nodes[5] | # Repeated basic events |
|---|---|---|---|---|
| FTA-1 | *Figure 3* | 5 | 9 | 0 |
| FTA-2 | Fig. 4 in ref [35] | 9 | 31 | 7 |
| FTA-3 | Fig. 5 in ref [28] | 15 | 43 | 7 |
| FTA-4 | *Figure 6* | 17 | 57 | 9 |
| FTA-5 | Fig. 6 in ref [28] | 18 | 55 | 10 |

**Table 3 – Characteristics of tested fault trees**

Hereafter, both the probability figures and the performance of our top-down recursive algorithm are compared to the minimal cut sets methods (using a cut set cutoff of 20 to have the most

[5] This includes both the gates and all the basic events (even duplicates)

possible precision) and to existing published algorithms. For simplicity purposes and so that the differences with the MCS methods are highlighted, all the basic events have been assigned an unavailability of 0.1.

***Table 4*** and ***Table 5*** respectively summarizes the results (accuracy) and time performance of the direct-evaluation algorithm, which has no errors when computing $Q$, compared to the MCS methods, specifically Method 4.

| FTA ID | $Q$ Exact Method | $Q$ MCS Methods | MCS Error ($Q$) compared to Exact (Analytical) method |
|---|---|---|---|
| FTA-1 | $2.71x10^{-3}$ | $2.71x10^{-3}$ | 0% |
| FTA-2 | 0.613 | 0.9 | 46.92% |
| FTA-3 | $3.89x10^{-2}$ | $4.38x10^{-2}$ | 12.61% |
| FTA-4 | 0.186 | 0.312 | 68.04% |
| FTA-5 | $3.81x10^{-6}$ | $4.24x10^{-6}$ | 11.17% |

**Table 4 – Accuracy error of the MCS methods of this paper**

| FTA ID | Execution Time (s) MCS Method 4 | Execution Time (s) Exact Method | Time reduction of Exact Method |
|---|---|---|---|
| FTA-1 | 0.120 | 0.118 | 1.9% |
| FTA-2 | 0.43 | 0.60 | -41.6% |
| FTA-3 | 1.16 | 1.35 | -15.5% |
| FTA-4 | 2.52 | 0.64 | 74.6% |
| FTA-5 | 12.08 | 0.92 | 92.4% |

**Table 5 – Time performance of DE vs MCS**

As it can be seen in ***Table 4***, there are important differences between the exact and the MCS methods. It is true that big probabilities have been used for each basic event (0.1), while the minimal Cut Set approximation is only accurate for small probabilities, but in some cases the Minimal Cut Sets produce an error close to 70% due to cumulative errors. The bigger and more complex the fault tree, the larger are the produced errors. However, as it can be seen in ***Table 4***, all the errors produced by the MCS methods are conservative, which is important in the safety industry.

Moreover, the results of performance provided in ***Table 5*** show that sometimes, our implemented and optimized direct evaluation is even faster (positive percentages on the last column) than the usage of Minimal Cut Sets, which was unimaginable 10 years ago. This is especially true for big fault trees.

Finally, as it can be observed in ***Table 6***, in most of the cases, the implemented direct evaluation is faster and requires less recursive calls than the published literature (sometimes it is not even possible to compute all the nodes of the fault tree in the literature as marked by an asterisk in the table). Note that the execution time cannot be directly compared, since different computers are used. However, the decrease in the number of recursive calls compared to the literature shows that the implemented Direct Evaluation algorithm has been really optimized regarding independence search.

## 5. Conclusion

Fulfilling the objectives of the study, it has been demonstrated that the implemented Direct Evaluation of a fault tree can compete in performance with the usage of Minimal Cut Sets. When compared to the optimized MCS method, which uses the minimum number of permutations and the Cut Set Cutoff, the analytical method shows faster results on most fault trees. Moreover, the described solution requires less recursions than most algorithms described in the literature. Being a Direct Evaluation, the accuracy of the results keeps the expected level compared to MCS method.

This combination of exact results and good performance has been achieved due to several means; a combination of statistical techniques on one hand, and software engineering on the other. First of all, a careful search for independence has been implemented to reduce the number of necessary recursive calls. Previous results of subsets of the fault tree are also stored in hash tables to avoid

unnecessary computations. Furthermore, when possible, queries to the database have been optimized by using object-oriented techniques to store the fault tree structure.

The results of the implemented exact algorithm are promising. The next steps of this work should be focused on testing the algorithm on more different types and sizes of fault tree. On the other hand, the combinatorial algorithms for the MCS have been optimized to a high degree. In the near future, different types of MCS algorithms can be implemented to see if even a further time gain is possible.

Possibly even a hybrid method involving the Minimal Cut Set and the Direct Evaluation could be introduced in the future. For instance, a good option would be to use the MCS for large fault trees without dependencies, and opt for the Direct Evaluation in case of having a medium fault tree with a lot of dependencies. This could make a great improvement in the calculation of complex fault trees; and not only that, this could also result in an advancement in the RAMS field.

Finally, it is worth mentioning, as already pointed out in section 2.2, that a detailed analysis and comparison of MCS methods with BDD and related methods is required.

| FTA ID | Execution Time (s) Exact Method | # Recursive calls | Time per recursion (s) | # Recursive calls in literature | Execution Time (s) in literature |
|---|---|---|---|---|---|
| FTA-1 | 0.118 | 11 | $1.07x10^{-2}$ | N/A | N/A |
| FTA-2 | 0.60 | 385 | $1.57x10^{-3}$ | *1778 ([35]) | *43 ([35]) |
| FTA-3 | 1.35 | 1086 | $1.24x10^{-3}$ | 7368 ([28]) | 155 ([28]) |
| FTA-4 | 0.64 | 160 | $4.14x10^{-3}$ | 758 ([35]) | 26 ([35]) |
| FTA-5 | 0.92 | 568 | $1.62x10^{-3}$ | 2223 ([28]) | 45 ([28]) |

**Table 6 - Performance of the direct-evaluation algorithm of this paper**

## 6. Acknowledgments

The author Jordi Pons-Prats acknowledges the support from Serra Hunter programme, Generalitat de Catalunya, as well as the support through the Severo Ochoa Centre of Excellence (2019-2023) under the grant CEX2018-000797-S funded by MCIN/AEI/10.13039/501100011033.

## References

[1] A. J. Kornecki and M. Liu, "Fault tree analysis for safety/security verification in aviation software," *Electronics* , vol. 2, no. 1, pp. 41–56, 2013, doi: 10.3390/electronics2010041.

[2] Q. Geng, H. Duan, and S. Li, "Dynamic fault tree analysis approach to Safety Analysis of Civil Aircraft," *Proceedings of the 2011 6th IEEE Conference on Industrial Electronics and Applications, ICIEA 2011*, pp. 1443–1448, 2011, doi: 10.1109/ICIEA.2011.5975816.

[3] J. Morio and M. Balesdent, *Estimation of Rare Event Probabilities in Complex Aerospace*. 2015.

[4] A. Mentes and I. H. Helvacioglu, "An application of fuzzy fault tree analysis for spread mooring systems," *Ocean Engineering*, vol. 38, no. 2–3, pp. 285–294, Feb. 2011, doi: 10.1016/j.oceaneng.2010.11.003.

[5] S. M. Lavasani, N. Ramzali, F. Sabzalipour, and E. Akyuz, "Utilisation of Fuzzy Fault Tree Analysis (FFTA) for quantified risk analysis of leakage in abandoned oil and natural-gas wells," *Ocean Engineering*, vol. 108, pp. 729–737, Nov. 2015, doi: 10.1016/j.oceaneng.2015.09.008.

[6] N. Ramzali, M. R. M. Lavasani, and J. Ghodousi, "Safety barriers analysis of offshore drilling system by employing Fuzzy event tree analysis," *Safety Science*, vol. 78,

pp. 49–59, Oct. 2015, doi: 10.1016/j.ssci.2015.04.004.

[7] L. Shi, J. Shuai, and K. Xu, “Fuzzy fault tree assessment based on improved AHP for fire and explosion accidents for steel oil storage tanks,” *Journal of Hazardous Materials*, vol. 278, pp. 529–538, Aug. 2014, doi: 10.1016/j.jhazmat.2014.06.034.

[8] Y. Liu, Z. P. Fan, Y. Yuan, and H. Li, “A FTA-based method for risk decision-making in emergency response,” *Computers and Operations Research*, vol. 42, pp. 49–57, 2014, doi: 10.1016/j.cor.2012.08.015.

[9] P. Zhang, W. Li, S. Li, Y. Wang, and W. Xiao, “Reliability assessment of photovoltaic power systems: Review of current status and future perspectives,” *Applied Energy*, vol. 104. Elsevier Ltd, pp. 822–833, 2013. doi: 10.1016/j.apenergy.2012.12.010.

[10] J. H. Purba, D. T. Sony Tjahyani, A. S. Ekariansyah, and H. Tjahjono, “Fuzzy probability based fault tree analysis to propagate and quantify epistemic uncertainty,” *Annals of Nuclear Energy*, vol. 85, pp. 1189–1199, Nov. 2015, doi: 10.1016/j.anucene.2015.08.002.

[11] V. Makis and J. Wu, *Effective Fault Detection and CBM Based on Oil Data Modeling and DPCA*. 2008. doi: 10.1007/978-1-84800-131-2_50.

[12] C. L. T. Borges, “An overview of reliability models and methods for distribution systems with renewable energy distributed generation,” *Renewable and Sustainable Energy Reviews*, vol. 16, no. 6. pp. 4008–4015, Aug. 2012. doi: 10.1016/j.rser.2012.03.055.

[13] W. E. Vesely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl, “Fault Tree Handbook (NUREG-0492),” p. 209, 1981, doi: NUREG–0492.

[14] S. Kabir, “An overview of fault tree analysis and its application in model based dependability analysis,” *Expert Systems with Applications*, vol. 77, pp. 114–135, 2017, doi: 10.1016/j.eswa.2017.01.058.

[15] E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Computer Science Review*, vol. 15, pp. 29–62, 2015, doi: 10.1016/j.cosrev.2015.03.001.

[16] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie, “Fault Tree Analysis, Methods, and Applications - A Review,” *IEEE Transactions on Reliability*, vol. R-34, no. 3, pp. 194–203, 1985, doi: 10.1109/TR.1985.5222114.

[17] R. Ferdous, F. Khan, B. Veitch, and P. R. Amyotte, “Methodology for computer aided fuzzy fault tree analysis,” *Process Safety and Environmental Protection*, vol. 87, no. 4, pp. 217–226, 2009, doi: 10.1016/j.psep.2009.04.004.

[18] K. B. Misra and G. G. Weber, “A new method for fuzzy fault tree analysis,” *Microelectronics Reliability*, vol. 29, no. 2, pp. 195–216, 1989, doi: 10.1016/0026-2714(89)90568-4.

[19] R. Duan and H. Zhou, “A New Fault Diagnosis Method Based on Fault Tree and Bayesian Networks,” *Energy Procedia*, vol. 17, pp. 1376–1382, 2012, doi: 10.1016/j.egypro.2012.02.255.

[20] R. Sinnamon, “Binary Decision Diagrams for Fault Tree Analysis,” 1996.

[21] J. Morio *et al.*, “A survey of rare event simulation methods for static input – output models To cite this version : HAL Id : hal-01081888,” 2014.

[22] S. Contini, “A new hybrid method for fault tree analysis,” *Reliability Engineering and System Safety*, vol. 49, no. 1, pp. 13–21, 1995, doi: 10.1016/0951-8320(95)00021-S.

[23] R. M. Sinnamon and J. D. Andrews, “Improved efficiency in qualitative fault tree analysis,” *Quality and Reliability Engineering International*, vol. 13, pp. 293–298, 1997.

[24] N. Khakzad, F. Khan, and P. Amyotte, “Safety analysis in process facilities: Comparison of fault tree and Bayesian network approaches,” *Reliability Engineering and System Safety*, vol. 96, no. 8, pp. 925–932, 2011, doi: 10.1016/j.ress.2011.03.012.

[25] Y. Kai, “Multistate fault-tree analysis,” *Reliability Engineering and System Safety*, vol. 28, no. 1, pp. 1–7, 1990.

[26] J. Yang, B. Zou, and M. Yang, “Bidirectional implementation of Markov/CCMT for dynamic reliability analysis with application to digital I&C systems,” *Reliability Engineering and System Safety*, vol. 185, no. December 2018, pp. 278–290, 2019, doi: 10.1016/j.ress.2018.12.024.

[27] A. Rauzy, “Mathematical foundations of minimal cutsets,” *IEEE Transactions on Reliability*, vol. 50, no. 4, pp. 389–396, 2001, doi: 10.1109/24.983400.

[28] F. A. Patterson-Hine and B. V. Koen, “Direct Evaluation of Fault Trees Using Object-Oriented Programming Techniques,” *IEEE Transactions on Reliability*, vol. 38, no. 2, pp. 186–192, 1989, doi: 10.1109/24.31103.

[29] J. Q. Walker II, “A node-positioning algorithm for general trees,” *Software - Practice and Experience*, vol. 20, no. 7, pp. 685–705, 1990.

[30] L. Rosenberg, “Algorithm for finding minimal cut sets in a fault tree,” *Reliability Engineering and System Safety*, vol. 53, no. 1, pp. 67–71, 1996, doi: 10.1016/0951-8320(96)00034-8.

[31] J. Vatn, “Finding minimal cut sets in a fault tree,” *Reliability Engineering and System Safety*, vol. 36, no. 1, pp. 59–62, 1992, doi: 10.1016/0951-8320(92)90152-B.

[32] S. Garribba, G. Reina, P. Mussio, F. Naldi, G. Reina, and G. Volta, “Efficient Construction of Minimal Cut Sets from Fault Trees,” *IEEE Transactions on Reliability*, vol. R-26, no. 2, pp. 88–94, 1977, doi: 10.1109/TR.1977.5220058.

[33] K. Kobayashi and H. Yamamoto, “New algorithm in enumerating all minimal paths in a sparse network,” *Reliability Engineering and System Safety*, vol. 65, no. 1, pp. 11–15, 1999, doi: 10.1016/S0951-8320(98)00076-3.

[34] Z. Tang and J. B. Dugan, “Minimal cut set/sequence generation for dynamic fault trees,” *Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 207–213, 2004, doi: 10.1109/rams.2004.1285449.

[35] L. B. Page and J. E. Perry, “A Simple Approach to Fault-Tree Probabilities,” *Computers & Chemical Engineering*, vol. 10, no. 3, pp. 249–257, 1986.

[36] L. B. Page and J. E. Perry, “DIRECT-EVALUATION ALGORITHMS FAULT-TREE,” *Computers & Chemical Engineering*, vol. 15, no. 3, pp. 157–169, 1991.

[37] T. Fenner and M. Levene, “Move generation with perfect hash functions,” *ICGA Journal*, vol. 31, no. 1, pp. 3–12, 2008, doi: 10.3233/icg-2008-31102.

[38] S. Contini, G. G. M. Cojazzi, and G. Renda, “On the use of non-coherent fault trees in safety and security studies,” *Reliability Engineering and System Safety*, vol. 93, no. 12, pp. 1886–1895, 2008, doi: 10.1016/j.ress.2008.03.018.