Nuclear Engineering and Technology 50 (2018) 854–859

Contents lists available at ScienceDirect

Nuclear Engineering and Technology

journal homepage: www.elsevier.com/locate/net

Original Article

A top-down iteration algorithm for Monte Carlo method for probability estimation of a fault tree with circular logic

Sang Hoon Han

Risk & Environmental Safety Research Division, Korea Atomic Energy Research Institute, 1045 Daedeokdaero, Yuseong-gu, Daejeon 305-353, Republic of Korea

ARTICLE INFO

Article history:
Received 28 December 2017
Received in revised form
21 March 2018
Accepted 2 April 2018
Available online 9 April 2018

Keywords:
Circular Logic
Fault Tree
FTeMC
Monte Carlo Approach
Top-Down Iteration Approach
Top Event Probability

ABSTRACT

Calculating minimal cut sets is a typical quantification method used to evaluate the top event probability for a fault tree. If minimal cut sets cannot be calculated or if the accuracy of the quantification result is in doubt, the Monte Carlo method can provide an alternative for fault tree quantification. The Monte Carlo method for fault tree quantification tends to take a long time because it repeats the calculation for a large number of samples. Herein, proposal is made to improve the quantification algorithm of a fault tree with circular logic. We developed a top-down iteration algorithm that combines the characteristics of the top-down approach and the iteration approach, thereby reducing the computation time of the Monte Carlo method.

1. Introduction

Probabilistic Safety Assessment (PSA) analysis requires the quantification of large fault trees. Fault tree quantification can be done using the minimal cut set method [1], the binary decision diagram method [2], or the Monte Carlo method [3]. The binary decision diagram method provides accurate values but is difficult to apply to large fault trees. The method using the calculation of minimal cut sets is the typical quantification method. This method has several kinds of error in the quantification result [4]. One error is a consequence of using “rare event approximation” or “minimal cut upper bound (MCUB)” method. The second error is a consequence of using “delete term approximation” to handle “NOT gates.” The third error comes from the use of truncation value, which can be minimized by establishing an appropriate truncation limit [10].

In addition, in multi-unit PSAs, which have recently been subjected to many studies, it is difficult to calculate minimal cut sets due to the increased size of the PSA models.

The Monte Carlo approach can be used when it is difficult to calculate minimal cut sets or when it is necessary to check the quantification result from the minimal cut set method. The accuracy of the Monte Carlo approach is proportional to the number of samples, and the computation time tends to be proportional to the product of the size of the model and the number of samples. The Monte Carlo method can be applied to very large fault tree; however, the calculation time could be long. For example, it might take several days to quantify all sequences of a large Level 1 PSA model using 10^9 sample size.

This article proposes an algorithm able to reduce the computation time when calculating the top event probability of a fault tree with circular logic. To quantify a fault tree using the Monte Carlo method, it is necessary to determine the state of the top event. A kind of top-down approach can be applied for fault trees with circular logic (called the top-down circular approach in this article) [5]. However, the top-down circular approach generally takes a long time. Therefore, an iteration approach is actually used.

We have reviewed the characteristics of existing approaches and developed a new approach, called the top-down iteration approach in this article, which combines the advantages of the top-down and iteration approaches.

We have developed and used software called FTeMC [9], which performs quantification of fault trees using the Monte Carlo method. All approaches discussed in this article have been implemented in FTeMC so that a user can select a calculation option.

E-mail address: shhan2@kaeri.re.kr.

https://doi.org/10.1016/j.net.2018.04.002

Section 2 describes the features of existing approaches and the new top-down iteration approach. Section 3 compares the calculation time of each approach for several fault tree models, and Section 4 provides a summary and conclusions.

2. Monte Carlo method for fault trees

2.1. Basic algorithm for the Monte Carlo method

The Monte Carlo method is to estimate the occurrence probability by testing how many times it occurs out of a number of trials. The basic algorithm of the method for calculating the top event probability of a fault tree is illustrated in Fig. 1.

- Repeat the following steps for N number of samples.
  - Determine the state (failure or success) of each basic event randomly.
  - Determine the state (failure or success) of the top event, and increase the failure count F if the state of the top event is true.
- If F failures occur during N trials, the top event probability is evaluated as F/N.

Let me show an example. It consists of three gates and three basic events as shown in Fig. 2. For each trial, the states of basic events are determined randomly. For example, A and C are True, and B is False at the first trial. Then, G1 and G2 are True, and in turn, TOP is True. For the second trial, A is True, and B and C are False. Then, G1 is True, and G2 is False, and in turn, TOP is False. The procedure is repeated for the sample size. If TOP is True two times out of three trials, the top event probability is estimated as 2/3.

```
// Repeat n times (# of samples)
For each sample
    // Decide state Si of each basic event Ei randomly
    For each Ei
        // Generate a random number r (0 ~ 1)
        r = rand()
        // Decide the state Si of Ei
        if (r ≤ Pi) then
            Si = True
        else
            Si = False
    // Calculate the state of a top event T
    St = GetState(T)
    // Count the number of failure for the top event
    If St ≡ True then
        p = p + 1
// Calculate the Top event probability Pt
Pt = p/n
```

Fig. 1. Monte Carlo main algorithm.

For large fault trees, it takes much time to determine the state of the top event. One of the following approaches can be used to determine the state of the top event:

- The top-down approach is used for a fault tree without circular logic.
- The top-down circular approach can be used for a fault tree with circular logics, but it takes a lot of calculation time.
- The iteration approach has been used for a fault tree with circular logics.
- A new top-down iteration approach is developed for a fault tree with circular logics.

The first three approaches, the top-down, the top-down circular, and the iteration approaches have been used previously and are described in Section 2.2–2.4, respectively. Section 2.5 describes the top-down iteration approach developed in this article.

2.2. Top-down approach

For a fault tree without circular logic, the top-down approach is effective in determining the state of the top event. Figs. 3 and 4 show the algorithm of the top-down approach. This algorithm starts from the top event and determines the state of the top event using the states of the children. The state of each child is determined using the states of its children, in turn.

GetState is a function that gets the state of each gate or basic event. To save computation time, a vector C is introduced to indicate whether the state of each gate has already been computed. This saves the calculation time by not recalculating the state of an already calculated gate.

The state of the undetermined gate is calculated using the states of its children. This process is described in a subfunction CalculateGateState. Each child's state is computed using the GetState function recursively.

Inside CalculateGateState, if the gate state is determined during calculation, we do not need to check the remaining children. For example, in the case of OR gate, if any of the children is true, the state of the gate is true; so, the rest are skipped without calculation. This process is important for reducing calculation time.

2.3. Top-down circular approach

In the presence of circular logic in a fault tree, the top-down approach cannot be used because an infinite loop is encountered.

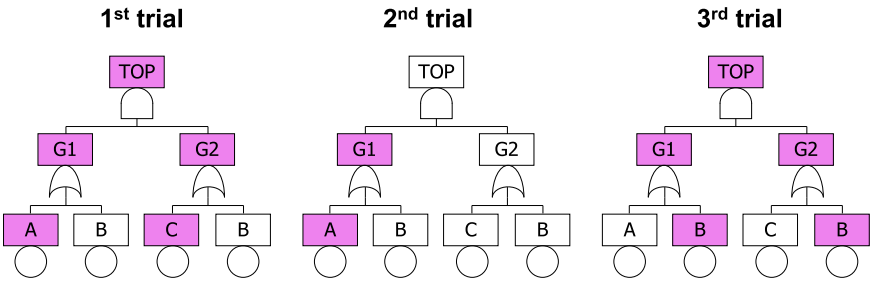

Fig. 2. An example of Monte Carlo method.

```
// Get the state of g, Sg
Function GetState(g)
  If (g is a basic event) then  // for a basic event
    return Sg  // State of g
  Else if (Cg ≡ True) then   // Sg is already calculated
    Return Sg
  Else  // Calculate the State for a Gate
    Cg = True    // If Calculated, Cg = True
    Sg = CalculateGateState (g)
  Return Sg
```

Fig. 3. Algorithm to check the state of a gate in the top-down approach.

```
// Calculate the state of a Gate, Sg
Function CalculateGateState(g)
  If (g is a OR gate) then  // for a OR gate
    For each child j ∈ g  // for each child of g
      Sj = GetState (j)
      if (Sj ≡ True) then   // if any one child is True
        return True        //    Sg = True
    return False           // Otherwise, Sg = False
  Else If (g is a AND gate) then  // for a AND gate
    For each child j ∈ g  // for each child of g
      Sj = GetState (j)
      if (Sj ≡ False) then   // if any one child is False
        return False        //    Sg = False
    return True            // Otherwise, Sg = True
  Else if (g is a NOT gate) then  // for a NOT gate
    For a child j          // For a child
      u = Not GetState(j) // Get Complement
    return u
```

Fig. 4. Algorithm to calculate the state of a gate in the top-down approach.

The top-down circular approach can be used in this case [5]. In the top-down circular approach, the function GetState is changed (see Fig. 5). Vector D is introduced to handle the circular logic encountered while navigating a fault tree.

The top-down circular approach implements Yang's algorithm [6], which modifies the fault tree logic of a specific gate according to the path of the circular logic. Therefore, in the top-down circular approach, the state of a specific gate may be different depending on which path of the circular logic the gate is in. Therefore, it must be repeatedly calculated if the gate appears in several places. This greatly increases the number of calculations. Note that in the top-down approach, once calculated, a gate does not need to be recalculated.

```
// Get the state of g, Sg
Function GetState(g)
  If (g is a basic event) then  // for a basic event
    return Sg          // State of g
  Else
    Dg = Dg + 1  // Increase the depth counter for g
    if (Dg = 2)    // encounter at the 2nd time, Circular logic made
      Return False  // State of g
    Else
      Sg = CalculateGateState (g)
      Dg = Dg – 1   // Decrease the depth count
      Return Sg     // State of g
```

Fig. 5. Algorithm to get the state of a gate in the top-down circular approach.

2.4. Iteration approach

For calculation of the minimal cut sets of a fault tree with circular logic, the iteration approach was developed [7,8]. The Monte Carlo method can also use the iteration approach to calculate the state of the top event, and this is more efficient than the top-down circular approach.

In the iteration approach, the main algorithm given in Fig. 6 is used instead of the function GetState. The algorithm repeats the calculation until the states of all gates converge. That is, it calculates the states of all the gates in each iteration and checks for changes. If there is no change, the result is converged, and the iteration is terminated.

The function CalculateGateState is different from that in the previous approaches and is shown in Fig. 7. It does not include the process of recursively calculating the states of children. The states of children are used as they were during the iteration process.

2.5. Top-down iteration approach

The iteration approach is more efficient than the top-down circular approach. However, in the top-down or top-down iteration approaches, it is possible to skip unnecessary calculation when the state of the gate can be determined using a preceding child, it is not necessary to calculate the states of the remaining children. Thus, we can skip the calculations for a large number of gates. In contrast with the iteration approach, calculations must be performed for all gates, each time.

This article proposes a top-down iteration approach that combines the advantages of top-down approach and iteration approach to reduce the computation time. The top-down iteration approach is based on the iteration approach. The iteration approach should compute states for all gates. Instead of computing all gate states, a modified top-down approach is incorporated to calculate the states of only the gates needed to determine the state of the top event. This approach can reduce the number of calculations compared with the iteration approach.

The top-down iteration approach also performs the iteration until the states of the gates no longer change, as in the iteration approach. Its main algorithm is given in Fig. 8. Basically, Fig. 8 performs the same function as Fig. 6 for the iteration approach. The only difference is where to check the convergence.

The function GetState for the top-down iteration approach is given in Fig. 9. The logic that calculates the state of each gate is similar to that in the top-down approach. It uses Vector C to check if it has been computed so that the computed gate is not recalculated. It is also used to prevent an infinite loop formed for circular logic. Once the state of gate g is calculated, it is stored in Sg. It is reused when the gate appears elsewhere in the fault tree. This is different from the top-down circular approach. A routine is also added to check for convergence.

The function CalculateGateState, which computes the state of each gate using the children's state, is given in Fig. 10 which is the same as that for the top-down approach given in Fig. 4.

The iteration approach should check all the gates. But, the top-down iteration approach checks only the gates needed to determine the state of the top event.

3. Application and results

We compared the computation times of the four approaches for various fault tree models. These calculations were performed not to

```
// Repeat until converged
Do
  Converged = True  // Initialize a variable for checking convergence
  For each gate g      // For each gate
    s = CalculateGateState (g) // Calculate the State for a Gate g
    if (s ≢ Sg) then    // Check convergence
      Converged = False
    Sg = s              // the State of a Gate g
Until (Converged)
```

Fig. 6. Algorithm to check the convergence in the iteration approach.

```
// Calculate the state of a Gate, Sg
Function CalculateGateState(g)
  If (g is a OR gate) then  // for a OR gate
    For each child j ∈ g  // for each child of g
      if (Sj ≡ True) then   // if any one child is True
        return True        //    Sg = True
    return False           // Otherwise, Sg = False
  Else If (g is a AND gate) then  // for a AND gate
    For each child j ∈ g  // for each child of g
      if (Sj ≡ False) then   // if any one child is False
        return False       //    Sg = False
    return True            // Otherwise, Sg = True
  Else if (g is a NOT gate) then  // for a NOT gate
    For a child j          // For a child
      u = Not Sj           // Get Complement
    return u
```

Fig. 7. Algorithm to calculate the state of a gate in the iteration approach.

obtain the required accuracy in quantification but to compare the calculation times. To get the required accuracy, the sample size should be adjusted (more samples = greater accuracy).

The fault tree models were selected to provide two models without circular logic and eight models with circular logic. Note that the top-down approach is the most efficient among the four approaches for fault trees without circular logic. For fault trees with circular logic, the top-down approach gives the wrong answer, but the point here was to perform the calculations to compare the calculation times of the approaches. Results are given in Table 1.

3.1. Description of results for each case

Cases N1 and N2 are large fault trees without circular logic. These were selected to check the efficiency of each approach. The top-down iteration approach took slightly more computation time than the top-down approach, but did not make much difference.

```
// Get the state of g, Sg
Function GetState(g)
  If (g is a basic event) then  // for a basic event
    return Sg  // State of g
  Else if (Cg ≡ True) then   // Sg is already calculated
    Return Sg
  Else  // Calculate the State for a Gate
    Cg = True    // If Calculated, Cg = True
    s = CalculateGateState (g)  //
    if (s ≢ Sg) then    // Check convergence
      Converged = False
    Sg = s
  Return Sg
```

Fig. 9. Algorithm to check the state of a gate in the top-down iteration approach.

The top-down circular and iteration approaches required at least three times more computation time than the top-down approach did. In Case N2, the top-down circular approach took about three times more computation time than the iteration approach did.

P1 and P3-L are small fault tree models with circular logic. Results for the top-down approach for a fault tree with circular logic are unreliable. However, calculations were performed to compare the calculation times. The top-down iteration approach took slightly more computation time than the top-down approach did. The top-down circular approach and the iteration approach took approximately 2–3 times longer time than the top-down circular approach.

A1 and A2 are large fault tree models with circular logic. The iteration approach took more than three times as much computation time as the top-down iteration approach did. Computation time for the top-down circular approach took about 2–5 times longer than for the iteration approach.

M1-L and M2-S are multi-unit PSA models that combine PSA models for several units. These also have circular logic. In case of M1-L, the top-down iteration approach computation time was about 1.2 times longer than that of the iteration approach. This was

```
// Repeat until converged
Do
  Converged = True  // Initialize a variable for convergence checking
  C() = False          //  Initialize , Ci : to check i-th gate is calculated
  St = GetState(T)   //  Calculate the state of the top event, recursively
Until (Converged)
```

Fig. 8. Algorithm to check the convergence in the top-down iteration approach.

```
// Calculate the state of a Gate, Sg
Function CalculateGateState(g)
  If (g is a OR gate) then  // for a OR gate
    For each child j ∈ g  // for each child of g
      Sj = GetState (j)
      if (Sj ≡ True) then   // if any one child is True
        return True         //   Sg = True
    return False            // Otherwise, Sg = False
  Else If (g is a AND gate) then  // for a AND gate
    For each child j ∈ g  // for each child of g
      Sj = GetState (j)
      if (Sj ≡ False) then   // if any one child is False
        return False         //   Sg = False
    return True             // Otherwise, Sg = True
  Else if (g is a NOT gate) then  // for a NOT gate
    For a child j           // For a child
      u = Not GetState(j) // Get Complement
    return u
```

Fig. 10. Algorithm to calculate the state of a gate in the top-down iteration approach.

the only case in which the iteration approach was faster than the top-down iteration approach. The top-down circular approach took too much computation time, making computation almost impossible. In case of M2-S, the top-down iteration approach appeared four times faster than the top-down circular approach and the iteration approach did.

C13 and C2 are fault tree models with several initiating events such as a risk monitor. They repeat calculations for each initiating event and combine the results. The CDF is estimated as CDF $= \Sigma$ (f(IE$_i$) * CCDP(IE$_i$ = True, IE$_{j\neq i}$ = False)) where f(IE$_i$) and CCDP are the frequency of each ith initiating event and the conditional failure probability for the initiating event, respectively. The top-down circular approach and the iteration approach took four times more computation time than the top-down iteration approach did.

3.2. Summary of results of the example calculations

The top-down approach showed the best performance for fault trees without circular logic. The top-down iteration approach took slightly more computation time than the top-down approach did.

In most cases, the top-down circular approach took the longest calculation time. Especially when the model was large and complex such as M1-L, it took too much calculation time relative to the other approaches.

The iteration approach was more efficient than the top-down circular approach but required more computation time than the top-down iteration approach did.

The top-down iteration approach performed the fastest calculation for most of the fault tree models with circular logic. Only one of eight fault tree models with circular logic was calculated quickly using the iteration approach.

The top-down approach produced the different answer for two cases. The other three approaches such as the top-down iteration approach, top-down circular approach, and iteration approach produced exactly the same answer. Note that the calculation was performed with the same value for the initial random number seed and then the same set of inputs was used for every approach. Thus, it produces exactly the same result if the algorithm is correct.

4. Summary and conclusion

The Monte Carlo method can be used for fault tree quantification when the minimal cut sets cannot be generated or when the

Table 1
Calculation time for test cases.

Case	Calculation time (s) [a] Probability[b]				Number of samples	Model
	Top-down[c]	Top-down circular[d]	Iteration[d]	Top-down iteration[d]		
N1	8 7.4e-5	42 7.4e-5	48 7.4e-5	8 7.4e-5	10^6	No circular logic, 4530 gates/1635 BEs[f]
N2	12 4.6e-5	116 4.6e-5	48 4.6e-5	13 4.6e-5	10^6	No circular logic, 3654 gates/2095 BEs
P1	7 9.61e-4	24 9.61e-4	13 9.61e-4	8 9.61e-4	10^7	Circular logic, 213 gates/194 BEs
P3-L	15 2.89e-3	51 2.89e-3	48 2.89e-3	16 2.89e-3	10^7	Circular logic, 561 gates/428 BEs
A1	19 5.1e-5[c]	370 6.6e-5	72 6.6e-5	21 6.6e-5	10^6	Circular logic, 7593 gates/3379 BEs
A2	14 1.78e-4	189 1.78e-4	79 1.78e-4	15 1.78e-4	10^6	Circular logic, 4530 gates/1258 BEs
M1-L	57 2.1e-4	>10,000 sec [e]	81 2.1e-4	90 2.1e-4	10^5	Circular logic, 49231 gates/23734 BEs
M2-S	52 1.9e-5	235 1.9e-5	329 1.9e-5	48 1.9e-5	10^6	Circular logic, 37345 gates/21217 BEs
C13	16 7.729e-7[c]	339 5.864e-9	288 5.864e-9	31 5.864e-9	10^6	Circular logic, 4530 gates/1258 BEs/8 IEs [g]
C2	22 8.526e-5	270 8.526e-5	148 8.526e-5	38 8.526e-5	10^5	Circular logic, 5056 gates/2458 BEs/20 IEs[g]

[a] The calculations are performed on a personal computer with Intel i7-6700 CPU (3.4 GHz) and Windows 7.
[b] In C13 and C2 cases, the probability represents the CDF, otherwise it is the top event probability.
[c] The top-down approach cannot calculate the state of a fault tree with circular logic. To prevent infinite loop, FTeMC assumes the state of a gate that cannot be calculated to be False (we may encounter a gate twice in a path because of circular logic). Thus, the quantification result of the top-down approach for a fault tree with circular logic is not reliable. The top-down approach produced the wrong values for the cases A1 and C13. However, analysis is performed to compare the calculation time.
[d] Three approaches (top-down circular, iteration, and top-down iteration approaches) produced exactly the same values for all cases.
[e] It takes too much calculation time (800 s for 1000 samples, 80,000 s expected for 100,000 samples).
[f] BE represents Basic event.
[g] The model is for a kind of risk monitor model. It includes several initiating events (IEs). CDF is calculated as CDF $= \sum$ (f(IE$_i$)* CCDP(IE$_i$ = True, IE$_{j\neq i}$ = False)). CDF and CCDP represents the core damage frequency and conditional core damage probability, respectively.

Table 2
Characteristics of approaches.

Approach	Characteristics	Calculation time
Top-down approach	- It implements a typical top-down recursive algorithm for navigating a fault tree. - It checks the states of only the gates needed to determine the state of the top event. - It reuses the calculated state for each gate. - It shows the best performance for fault trees without circular logic. - The result is not reliable for fault trees with circular logic.	The fastest
Top-down circular approach	- It implements Yang's algorithm [6] developed for the analysis of fault trees with circular logic. - It checks the states of only the gates needed to determine the state of the top event. - It repeatedly calculates the gate that appears in several places which greatly increases the number of calculations.	The longest calculation time
Iteration approach	- It iterates the calculation until the result converges [7,8]. - It checks the states of all gates.	Faster than the top-down circular and slower than the top-down iteration approach
Top-down iteration approach	- It iterates the calculation until the result converges. - It checks the states of only the gates needed to determine the state of the top event.	The fastest for fault trees with circular logic

PSA quantification results need to be validated. The Monte Carlo method requires a long calculation time because it repeats the calculation for a large number of samples. Because PSA models include circular logic, it was required to develop an algorithm that could efficiently perform quantification of fault trees with circular logic.

This article describes a Monte Carlo method for calculating the top event probability of a fault tree with circular logic. We developed a new algorithm called the top-down iteration approach to reduce computation time. New approach and existing approaches are tested for several fault tree models. The characteristics of each approach are summarized in Table 2.

The top-down iteration approach developed in this article is fairly efficient, meaning that it performs calculations much faster than is done by the top-down circular or iteration approaches.

Conflicts of interest

No conflicts of interest to declare.

Acknowledgments

The author gratefully appreciates the helpful supports and input from Jin Hee PARK and Dong San KIM. This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean government (MSIT: Ministry of Science and ICT) (No. 2017M2A8A4015287).

References

[1] Fault Tree Handbook, U.S. Nuclear Regulatory Commission, NUREG-0492, 1981.
[2] A. Rauzy, New algorithms for fault trees analysis, Reliability Engineering and System Safety 40 (1993) 203–211.
[3] H. Kumamoto, 1980. Dagger-sampling Monte Carlo for system unavailability evaluation, IEEE Transactions on Reliability R 29 (2) (1980) 122–125.
[4] Sang Hoon Han, Jin Hee Park, Dong San Kim, Ho-Gon Lim, Seung Cheol Jang, Comparison of Various Methods to Quantify a Fault Tree for Seismic PSA, Transactions of the Korean Nuclear Society Spring Meeting, Jeju, Korea, 2015. May 7–8.
[5] Sang Hoon Han, Ho-Gon Lim, Top event probability evaluation of a fault tree having circular logics by using Monte Carlo method, Nuclear Engineering and Design 243 (2012) 336–340.
[6] Joon-Eon Yang, Sang-Hoon Han, Jin-Hee Park, Young-Ho Jin, Analytic method to break logical loops automatically in PSA, Reliability Engineering and System Safety 56 (1997) 101–105.
[7] A Study for Nuclear Safety Improvement - Improvement of Level 1 PSA Computer Code Package, KAERI, KAERI/RR-1487/94, 1995.
[8] Jussi K. Vaurio, A recursive method for breaking complex logic loops in Boolean system models, Reliability Engineering and System Safety 92 (2007) 1473–1475.
[9] FTeMC Quick Guide, Fault Tree Top Event Probability Evaluation Using Monte Carlo Simulation, KAERI, KAERI-ISA-memo-FTeMC-01, Rev. 1, 2017.
[10] Addenda to ASME/ANS RA-S-2008 Standard for Level 1/Large Early Release Frequency Probabilistic Risk Assessment for Nuclear Power Plant Applications, ASME/ANS RA-Sb-2013, The American Society of Mechanical Engineers, 2013.