

HYBRID FAULT TREE ANALYSIS IN SCRAM++: A MODULAR FRAMEWORK FOR SCALABLE PROBABILISTIC RISK ASSESSMENT

Nailah Afshan¹, Saran Srikanth Bodda², Abhinav Gupta³, and Kevin Han⁴

¹PhD Student, CCEE, North Carolina State University, USA (nafshan@ncsu.edu)

²Research Faculty, CCEE, NCSU, USA

³Director, Center for Nuclear Energy Facilities and Structures, NCSU, USA

⁴Associate Professor, CCEE, NCSU, USA

ABSTRACT

Probabilistic Risk Assessment (PRA) systematically evaluates risks in complex systems, widely applied in sectors like nuclear energy and aerospace. Fault Tree Analysis (FTA) is essential within PRA, utilizing algorithms such as Binary Decision Diagrams (BDD), Zero-Suppressed BDD (ZBDD), Method of Obtaining Cut Sets (MOCUS), and the recently proposed Compressed Truth Table (CTT) algorithm. The CTT algorithm introduces a structured, closed-form probability calculation approach while maintaining computational efficiency. However, it is still under development with limited implementation to validate its effectiveness for complex scenarios. Each FTA algorithm uniquely addresses specific fault tree structure but has limitations in efficiently managing large, complex systems due to computational intensiveness and complex interrelationships. This paper proposes a hybrid implementation method, integrating various FTA algorithms, in particular, the CTT algorithm, through a structured workflow: module identification, module analysis, module replacement, and final analysis. Initially, fault trees are decomposed into simpler modules. Each module undergoes analysis tailored to its specific characteristics, employing the most suitable algorithm for accurate failure probability computation. Modules are then simplified by substituting detailed structures with equivalent basic events reflecting their failure probabilities. Finally, a comprehensive analysis evaluates overall system failure probability, effectively addressing challenges inherent to traditional FTA. The proposed hybrid approach demonstrates that selective use of the CTT algorithm within a hybrid modular framework can improve computational efficiency and scalability in PRA for large, complex fault trees.

INTRODUCTION

PRA is a widely adopted methodology for evaluating the reliability and safety of complex engineered systems, especially in nuclear power plants (NPPs) and aerospace industries. FTA, a foundational tool within PRA, models the logical interrelationships among component failures leading to system-level faults. Classical FTA methods often rely on deriving Minimal Cut Sets (MCS), with early algorithms like MOCUS performing top-down enumeration of failure combinations. While effective for small systems, such approaches are susceptible to exponential complexity as model complexity increases, often requiring truncation or rare-event approximations that may compromise accuracy ([Rauzy, 2003](#)). BDDs address the limitations of MOCUS, providing exact and canonical representations of Boolean logic in fault trees. BDDs enable precise computation of top event probabilities without approximations but remain constrained by exponential memory growth and sensitivity to variable ordering ([Jung et al., 2004](#)). ZBDDs further refine this concept by modifying reduction rules to better handle sparse failure combinations, significantly improving efficiency for certain fault tree structures. ([Jung, 2009](#)).

Recent comparative studies have evaluated the performance of these algorithms under varying fault tree structures. [Afshan et al. \(2024a\)](#) demonstrate that algorithmic efficiency is highly dependent on tree

structure, with specific gate structures and interrelationships between the events triggering large memory requirements or excessive runtimes. Similarly, [Aras et al. \(2022\)](#) benchmark the BDD-based XFTA engine and the open-source SCRAM tool ([Rakhimov, 2015](#)), finding comparable numerical accuracy but differing computational performance due to intermediate decision diagram optimization. These studies collectively highlight that no single algorithm performs optimally across all cases, motivating the need for adaptive or hybrid strategies. To address this, [Afshan et al. \(2023, 2024b\)](#) introduced SCRAM++, an enhanced version of the SCRAM analysis platform, supporting traditional algorithms (MOCUS, BDD, ZBDD) and the recently proposed CTT algorithm. SCRAM++ also incorporates a modularization framework, which decomposes fault trees into smaller, structurally distinct modules to improve solver efficiency and scalability ([Chen et al., 2019](#)). This modular decomposition enables independent analysis of subtrees (modules), effectively reducing computational overhead ([Chen et al., 2019](#); [Yllera, 2018](#)).

Building on this foundation, the present study introduces an advanced hybrid implementation within SCRAM++. The proposed method integrates multiple FTA algorithms: BDD, ZBDD, MOCUS, and CTT; by assigning the most suitable solver to each module based on its structural features. The hybrid workflow follows a four-stage process: module identification, targeted module analysis, replacement of modules with equivalent basic events, and system-level recombination. This strategy is informed by prior work emphasizing the need to address interdependent structures such as modules containing multiple event interrelationships ([Vaishanav et al., 2024, 2020](#)). By leveraging the strengths of different algorithms within a unified, modular framework, this research advances the computational feasibility of PRA for large, highly interconnected systems, ultimately contributing to more efficient and scalable safety assessments.

BACKGROUND OF SCRAM++

SCRAM++ is an enhanced, research grade extension of the open-source SCRAM command-line PRA engine ([Afshan et al., 2024b](#)). While the legacy tool already supports event and fault tree modeling, probabilistic analysis, importance measures, and uncertainty propagation, SCRAM++ introduces a unified and modular workflow that streamlines these capabilities for large-scale, heterogeneous fault tree problems. The input files to SCRAM++, written in the Open-PSA Model Exchange Format (opsa-mef) are run through a four-stage processing pipeline to standardize data flow and enable multiple algorithm selection ([Afshan et al., 2024b](#)). An overview of the algorithm suite used in SCRAM++ is given in [Table 1](#).

Table 1: Overview of FTA algorithms available in SCRAM++ ([Afshan et al., 2024b](#))

Class	Algorithm	Methodology	Strengths	Limitations
Cut-set	MOCUS	Minimal cut set enumeration	Simple and effective for small trees	Poor scalability; exponential growth for large trees
Compressed Boolean	ZBDD	Zero-suppressed BDD encoding	Memory efficient for sparse logic	Not suitable for dense trees; exponential runtime
Decision diagram	BDD	Binary decision diagram traversal	Compact and exact for coherent trees	Exponential time and memory in complex trees
State table	CTT	Compressed truth tables	Scalable and exact for independent trees	Limited support for trees with dense and nested event interrelationships

SCRAM++ implements modularization through a single, breadth first pass of the fault tree graph as

shown in Fig. 1

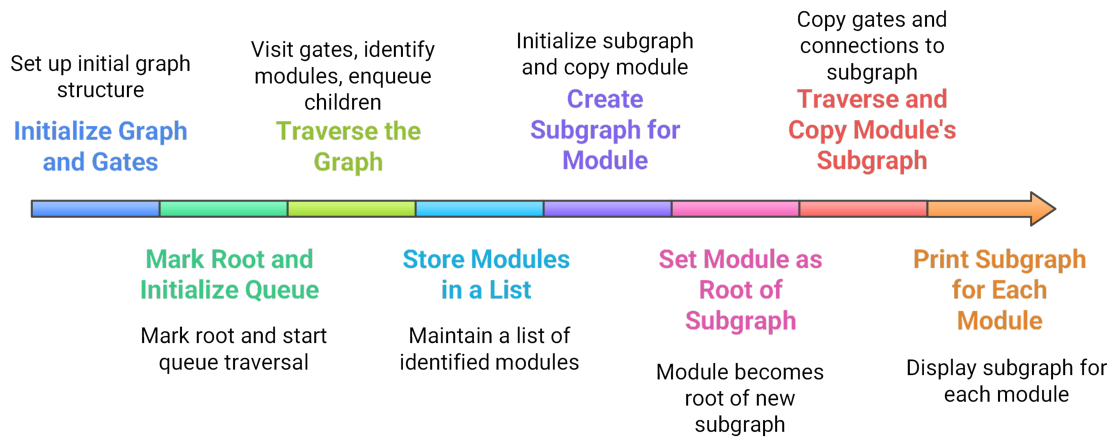


Figure 1. Modularization in SCRAM++

The procedure begins by instantiating the full graph, marking the top event (root gate), and pushing that root onto a traversal queue. During the traversal, each gate is dequeued, labelled as visited, and inspected against user-defined heuristics that qualify it as a module, a subtree that can be evaluated independently of the rest of the model. Qualified gates are appended to a running module list while all of their child gates are enqueued for continued exploration. When the queue empties, the list captures every detachable sub-tree in the fault tree. The framework then iterates through this list: for each module, it spins up a fresh graph object, copies the module gate, designates that copy as the new root, and performs a second breadth first copy of all descendant gates and links. This coarse grained decomposition unlocks two key advantages: it enables the implementation of a hybrid approach that tailors each algorithm choice to module characteristics, and it paves way for parallel execution in the future.

HYBRID APPROACH IN SCRAM++

The proposed hybrid approach follows a structured workflow comprising four main components. These components include module identification, module analysis, module replacement, and the final analysis of the resultant fault tree. Firstly, module identification is done to detect, label, and store different modules within an input fault tree. This step isolates and focuses on the most challenging components of the fault tree that require specialized analysis. Secondly, each identified module is analyzed to calculate its failure probability. This step involves a condition-based approach and an FTA algorithm is implemented for analysis based on specific features of each module. This step leverages the strengths of these algorithms, as shown in Table 1, on a fault tree that is now more manageable due to the simplifications introduced. Next, the module replacement step substitutes the analyzed modules in the fault tree with basic events that represent the calculated failure probabilities. This simplification reduces the complexity of the fault tree without compromising the accuracy of the risk assessment, as the essential risk contributions of the modules are preserved in the basic events. Lastly, a final analysis of the resultant fault tree is done using the suite of FTA algorithms. It ensures that the overall reliability of the system is accurately assessed and that the simplifications have maintained the necessary detail for effective risk evaluation.

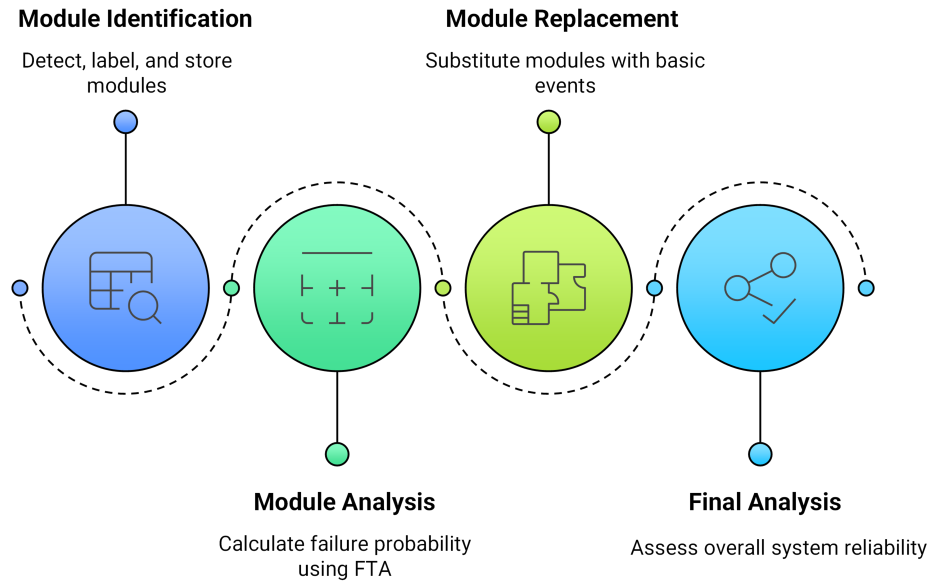


Figure 2. Hybrid approach steps in SCRAM++

METHODOLOGY

This section presents the methodology used to evaluate the efficiency of the proposed hybrid approach in SCRAM++. The efficiency is represented in terms of the total analysis time of an algorithm or procedure. The total analysis time includes both the computational time and algorithmic implementation time (Afshan et al., 2024b). For hybrid approach, it includes the time taken for modularization, determination of module specifications and other necessary steps. The analysis utilizes a suite of synthetically generated fault trees created using the Fault Tree Generator (FTG) code of SCRAM tool. These fault trees vary in width, height, number of basic events, and the presence of event interrelationships, with no fixed structural pattern. Instead, they feature randomized structures, where basic events have varying path lengths to the top event and dense interrelationships due to shared components. This randomness introduces realistic complexity, particularly through the event interrelationships. The key steps of the analysis methodology are outlined in the following subsections.

Fault Tree Specifications and Module Characteristics

To evaluate the performance of the hybrid approach, a diverse set of fault trees is considered. These fault trees feature variations in the number of basic events (BEs), intermediate events (IEs), and interrelationships between events in the tree. The presence of more interrelationships in a fault tree adds to its complexity. These interrelationships are represented by multiple occurring events (MOEs), which represent shared failure events across different components of a fault tree. Table 2 summarizes the key specifications of each fault tree used in the analysis.

Table 2: Specifications of FTG-generated fault trees and their module characteristics

Case#	$n(\text{BE})$	$n(\text{IE})$	$n(\text{Modules})$			
			Type I	Type II	Type III	Total
I	100	54	43	0	1	44
II	500	277	189	0	4	193
III	1000	543	387	0	6	393
IV	1500	816	555	0	16	571
V	2000	1128	759	0	17	776
VI	2100	1160	777	0	26	803
VII	2300	1223	869	1	26	896
VIII	2500	1324	961	0	25	986
IX	3000	1668	1134	0	29	1163

Table 2 highlights the structural diversity across the test cases. The modules in a fault tree are divided into three types based on the type of interrelationships between events: (I) Independent modules, (II) Modules with MOEs at BE level, and (III) Modules with all other types of interrelationships. Independent modules are the logically independent subtrees with no shared events or MOEs. While early fault trees consist primarily of independent modules, larger trees (e.g., Case VII, Case VIII, Case IX) include an increasing number of modules with multiple interrelationships. These interrelationships add significant complexity and are critical in assessing the effectiveness of the hybrid solver under realistic PRA conditions.

Hybrid Analysis Scenarios

To evaluate the effectiveness of the hybrid implementation in SCRAM++, a series of fault tree scenarios is used. These scenarios are designed to compare the hybrid approach with traditional baseline algorithms across various fault tree structures and logic gate types. The analysis specifically focuses on evaluating the hybrid implementation, wherein individual modules within a fault tree are analyzed using different FTA algorithms selected based on their structural properties. The hybrid framework leverages a module-wise classification scheme informed by the characteristics summarized in Table 2. Based on the type of module, the most appropriate algorithm is assigned. For example, CTT algorithm is used to analyze modules of type (I) and (II), as it provides exact failure probability estimates with high computational efficiency. Type (III) modules are directed to the ZBDD algorithm over BDD or MOCUS as these algorithms result in memory or MCS explosion respectively, as noted in Table 1. Moreover, ZBDD is more efficient for modules with NOT gates. This modular, condition-based solver assignment enables the hybrid framework to effectively manage interrelationships while maintaining scalability and accuracy.

The fault trees used in this analysis are listed in Table 2, which summarizes their structural characteristics including the number of BEs, IEs, and types of modules based on interrelationships between events. To evaluate the hybrid solver under different logical structures, two variations of each fault tree listed in Table 2 are considered:

- **Scenario 1: Coherent fault trees (i.e., only AND/OR gates)** – These cases use the original fault trees generated by the FTG tool in SCRAM++, which constructs trees composed exclusively of AND and OR logic gates.
- **Scenario 2: Non-coherent fault trees (includes NAND/NOR gates)** – For this scenario, a modified version of each fault tree is created by manually replacing approximately 50% of the AND and OR

gates with their NOT equivalents (NAND and NOR). This allows evaluation of the hybrid solver’s robustness under increased logical complexity.

These scenarios provide a structured framework to quantify improvements in computational efficiency and scalability achieved by the hybrid implementation across realistic PRA modeling conditions. To quantify the improvement in computational efficiency achieved by the hybrid approach, a performance enhancement metric termed as *Speedup* is defined as the ratio of the total analysis time of the baseline algorithm to that of the hybrid solver, as shown in Eq. (1). A speedup greater than 1 indicates that the hybrid implementation takes lesser runtime compared to the baseline algorithms in terms of total analysis time, whereas a value less than 1 reflects performance overhead introduced by extra steps involved in hybrid implementation like modularization, analysis of individual modules to extract different specifications, etc.

$$Speedup = \frac{T_{Baseline}}{T_{Hybrid}} \quad (1)$$

RESULTS AND DISCUSSION

This section presents the results from the two analysis scenarios described in the previous subsection, focusing on the efficiency of hybrid solver under varying fault tree structures.

Scenario 1: Exclusion of NOT Gates (AND/OR Only)

Fig. 3 shows the total analysis time for the hybrid solver against standalone executions of baseline algorithms such as MOCUS, ZBDD, and BDD. It can be seen that the the hybrid solver shows lower total analysis times compared to the baseline algorithms across various fault tree cases. This finding highlights the computational advantage of the hybrid method, especially for medium to large scale problems.

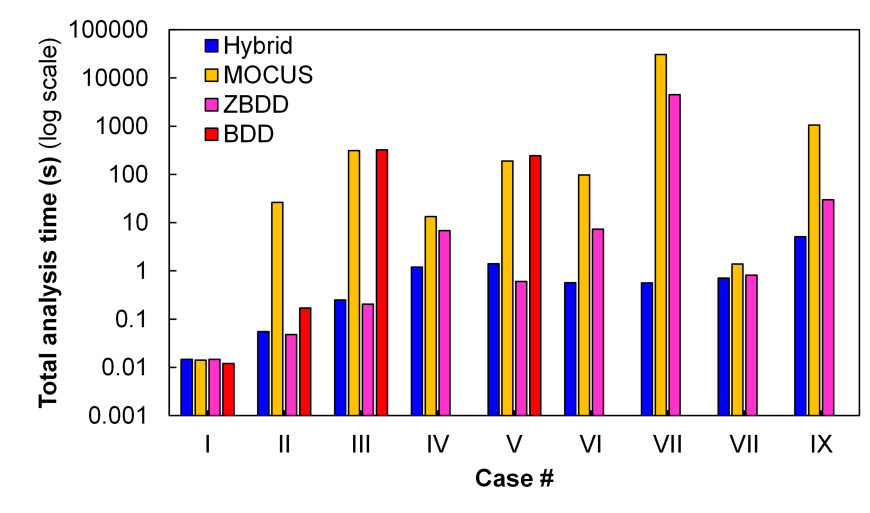


Figure 3. Total analysis time for hybrid approach and baseline algorithms for Scenario 1

In cases such as III, VI, VII and IX, MOCUS and ZBDD exhibit total analysis times that are several orders of magnitude higher than the hybrid solver, indicating the latter’s scalability and efficiency in complex FTA. Notably, the BDD algorithm performs comparably only in the smallest cases (I–III) but fails or is unable to complete execution beyond Case V, highlighting its limitations in handling large or deeply nested

trees due to memory explosion (Afshan et al., 2023). In contrast, the hybrid approach maintains consistently low total analysis times across most cases, demonstrating its robustness and effectiveness in reducing the computational burden associated with baseline algorithms. This performance gain is largely attributed to the modular decomposition and algorithm-specific tailoring embedded in the hybrid strategy.

The speedup table shown in Table 3 clearly demonstrates the computational advantages of the hybrid implementation over baseline algorithms across a range of fault tree scenarios representative of realistic PRA modeling conditions. As defined in Eq. (1), a speedup value greater than 1 indicates a performance improvement. When compared to MOCUS algorithm, the hybrid approach achieves exceptional speedups ranging from 134× in Case V to over 54000× in Case VII. These high values reflect the hybrid approach’s ability to avoid exhaustive top-down cut set enumeration that MOCUS relies on. In larger trees (Cases VI, VII, IX), the hybrid solver processes modules independently and avoids redundant traversal, making it several orders of magnitude faster.

Table 3: Improvement of hybrid approach with respect to baseline algorithms for Scenario 1

Case#	Speedup w.r.t.		
	MOCUS	ZBDD	BDD
I	1.0	1.0	0.8
II	475.3	0.9	3.1
III	1248.1	0.8	1281.2
IV	11.3	5.8	NA
V	134.5	0.4	174.3
VI	170.3	12.9	NA
VII	54407.5	8045.6	NA
VIII	2.0	1.2	NA
IX	206.5	5.8	NA

In large-scale cases (VI, VII, VIII, and IX), the hybrid approach achieves 1.2× to 8000× speedups, demonstrating performance improvements when diagrammatic compression becomes insufficient due to multiple interrelationships. However, in smaller or moderately sized cases (e.g., II–V), ZBDD occasionally performs comparably or better, with speedups close to or slightly below 1.0. This observation indicates that when the overhead of modularization outweighs the complexity of the tree, symbolic and zero-suppression methods can perform equally well. In the case of BDD algorithm, the hybrid approach generally shows higher speedups (e.g., 174× in Case V and 1281× in Case III). It must be noted that BDD fails to compute results in Case IV and all cases beyond Case V. This failure indicates the scalability limitations of BDD when applied to deep or highly complex fault trees with larger and nested interrelationships. In contrast, the hybrid approach maintains consistent performance throughout all cases, reinforcing its robustness and practical applicability to large-scale PRA models.

Case VII stands out as a special case as it is the only case containing modules of type (II), a structure well-suited to the CTT algorithm. Here, the hybrid solver achieves more than 50000× speedup over MOCUS and more than 8000× over ZBDD, highlighting the effectiveness of selective algorithm assignment within the modular framework. While baseline algorithms suffer from exponential growth in runtime, the hybrid solver efficiently handles the complexity by applying CTT to modules with recurring but structurally manageable MOEs at BE level.

Overall, the speedup results indicate that the hybrid solver delivers computational gains, especially

in complex scenarios, by using modular decomposition and targeted algorithm selection, most importantly, the integration of the CTT algorithm for FTA.

Scenario 2: Inclusion of NOT Gates (NAND/NOR)

Fig. 4 shows the total analysis time for the hybrid solver against standalone executions of traditional algorithms considering NOT gates.

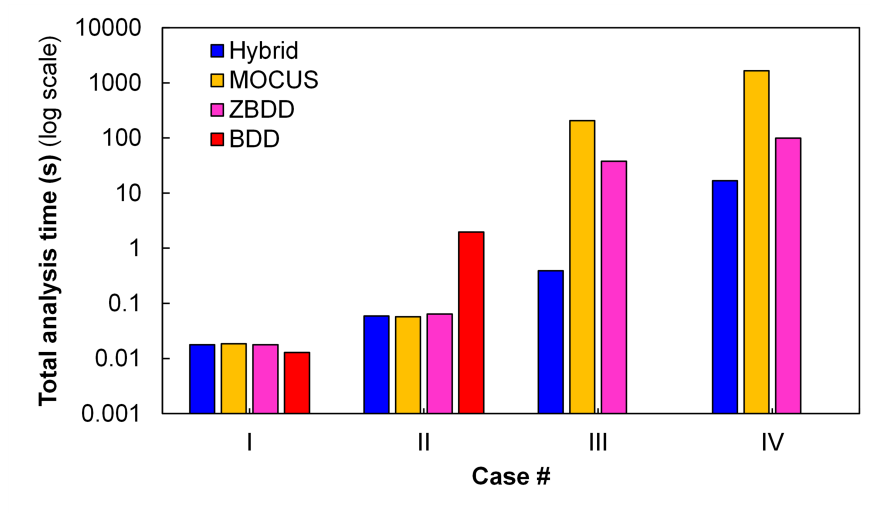


Figure 4. Total analysis time for hybrid approach and baseline algorithms for Scenario 2

The performance improvements in Scenario 2 follow a similar trend as Scenario 1, despite the added NOT-gate complexity. For MOCUS, the hybrid solver achieves comparable performance in small cases (I–II) but outperforms it in Cases III and IV, with speedups of 528× and 98× as shown in Table 4, respectively. Similarly, the performance of hybrid solver compared to ZBDD algorithm is nearly equal in Cases I–II, but the hybrid solver becomes performs better as complexity increases. In Case III, hybrid reduces runtime by 97×, and in Case IV, by nearly 6×, showing its scalability advantage over symbolic and zero-suppressed techniques when interrelationships grow. A similar observation is made for BDD algorithm, which again fails after Case II. In contrast, the hybrid solver runs efficiently throughout all scenarios.

Table 4: Improvement of hybrid approach with respect to baseline algorithms for Scenario 2

Case#	Speedup w.r.t.		
	MOCUS	ZBDD	BDD
I	1.1	1.0	0.7
II	1.0	1.1	33.5
III	527.9	96.6	NA
IV	98.4	5.9	NA

These results confirm that even with the inclusion of NOT gates, the hybrid approach maintains low analysis times, generally showing better scalability than MOCUS, ZBDD or BDD. This finding further supports the hybrid approach’s adaptability to logical complexity. In summary, the comparative analysis across

all scenarios highlights several important findings regarding hybrid approach's performance, modularization, and the effectiveness of the hybrid implementation as enlisted below:

- The hybrid solver generally outperforms traditional algorithms by assigning algorithms based on module-specific characteristics. It remains stable even under logical complexity (NAND/NOR gates), where baseline solvers may fail or experience significant slowdowns.
- CTT is highly effective for independent modules and modules with MOEs at BE level, while ZBDD is better suited for sparse but complex modules.
- In large and complex fault trees, the CTT algorithm enables the hybrid approach to remain both scalable and efficient, delivering performance improvements of up to 100% in several cases. Even in modules with limited interrelationships, such as those of type (II), the CTT algorithm maintains its effectiveness, making it a key contributor to the overall efficiency and robustness of the hybrid approach.

Overall, the hybrid approach provides a scalable and reliable solution for realistic FTA in PRA, effectively balancing runtime performance and scalability in PRA applications.

CONCLUSION

This study presents a hybrid FTA framework implemented within the SCRAM++ platform to enhance the efficiency and scalability of PRA for complex systems. By combining multiple FTA algorithms and applying them selectively based on module-specific structural characteristics, the hybrid approach addresses key limitations of traditional algorithms. The methodology involved generating fault trees of varying size and complexity using the SCRAM FTG tool, including manually constructed variants with NOT gates (NAND/NOR) to introduce logical diversity. The hybrid approach generally delivers improved performance across most cases. It achieves this performance by leveraging CTT for independent modules as well as modules with MOEs at BE level, and ZBDD for sparse, densely interconnected ones, while avoiding memory issues and failures, especially in complex cases. Overall, the hybrid implementation proves to be a scalable and computationally efficient solution for PRA applications. It offers a flexible analysis pipeline that can adapt to diverse fault tree structures and logic structures, making it well suited for high fidelity risk modeling in nuclear, aerospace and other safety critical domains. In future work, the hybrid approach will be extended to evaluate even larger and more complex fault trees, as well as event trees, to further assess its scalability and robustness. Additionally, the framework will be applied to actual NPP scenarios to validate its practical effectiveness and enhance its applicability in real world scenarios.

ACKNOWLEDGMENTS

This research was supported by the Center for Nuclear Energy Facilities and Structures at NC State University. Resources for the Center come from the dues paid by member organizations and from the Civil Engineering Department and College of Engineering.

REFERENCES

- Afshan, N., Bodda, S. S., Gupta, A. and Han, K. (2023). "Development of an integrated platform for probabilistic risk assessment using fault tree analysis." In *2023 ASCE Engineering Mechanics Institute Conference*.
- Afshan, N., Bodda, S. S., Gupta, A. and Han, K. (2024a). "Evaluation of Fault Tree Analysis Algorithms

for Probabilistic Risk Assessment: A Systematic Comparative Study.” In *International Probabilistic Workshop*, pp. 137–146, Springer.

- Afshan, N., Bodda, S. S., Gupta, A. and Han, K. (2024b). “SCRAM+: A Unified Probabilistic Risk Assessment Platform for Improved Decision-Making in Reliability Analysis.” In *Transactions of the 27th International Conference on Structural Mechanics in Reactor Technology*.
- Aras, E. M., Farag, A. S., Earthperson, A. and Diaconesa, M. A. (2022). “Benchmark Study of XFETA and SCRAM Fault Tree Solvers Using Synthetically Generated Fault Trees Models.” In *ASME International Mechanical Engineering Congress and Exposition*, volume 86717, p. V009T14A016, American Society of Mechanical Engineers.
- Chen, M., Xiao, N.-C., Zuo, M. J. and Ding, Y. (2019). “An efficient algorithm for finding modules in fault trees.” *IEEE Transactions on Reliability*, 70(3), pp. 862–874.
- Jung, W. S. (2009). “ZBDD algorithm features for an efficient Probabilistic Safety Assessment.” *Nuclear Engineering and Design*, 239(10), pp. 2085–2092.
- Jung, W. S., Han, S. H. and Ha, J. (2004). “A fast BDD algorithm for large coherent fault trees analysis.” *Reliab Eng Syst Saf*, 83, pp. 369–374, doi:10.1016/J.RESS.2003.10.009.
- Rakhimov (2015). “SCRAM: Command Line Risk Analysis Tool.” Accessed: Mar. 14, 2023.
- Rauzy, A. (2003). “Toward an efficient implementation of the MOCUS algorithm.” *IEEE Transactions on Reliability*, 52(2), pp. 175–180.
- Vaishnav, P., Bodda, S. S. and Gupta, A. (2024). “Computationally efficient approach for risk-informed decision making.” *Progress in Nuclear Energy*, 167, p. 104983.
- Vaishnav, P., Gupta, A. and Bodda, S. S. (2020). “Limitations of traditional tools for beyond design basis external hazard PRA.” *Nuclear Engineering and Design*, 370, p. 110899.
- Yllera, J. (2018). “Modularization methods for evaluating fault trees of complex technical systems.” In *Engineering risk and hazard assessment*, pp. 81–100, CRC Press.