DEVELOPMENT OF AN ADVANCED FAULT TREE QUANTIFICATION ENGINE BASED ON BDD/ZBDD ALGORITHM

Wei GAO[1], Qinfang ZHANG[1], Guofeng TANG[1]

[1] *Shanghai Nuclear Engineering Research & Design Institute, Shanghai, China, 200233, gaowei@snerdi.com.cn, zhangqf@snerdi.com.cn, tangguofeng@snerdi.com.cn*

Binary Decision Diagram (BDD) and Zero-suppressed Binary Decision Diagram (ZBDD) are popular techniques for solving Probabilistic Risk Assessment (PRA)/ Probabilistic Safety Analysis (PSA) models nowadays. This paper describes development of an advanced fault tree quantification engine based on both BDD and ZBDD algorithms. A brief introduction including the ways of fault tree construction, conversion from fault tree to BDD/ZBDD structure, obtaining Minimal Cut-sets (MCS), and calculation of top event unavailability is presented in this paper. A compiling process is also introduced based on these methods. This engine is validated by calculating PSA models of certain nuclear power plants and comparing results with RiskSpectrum PSA. The comparison results show that this engine can efficiently quantify PSA models in large scale. Finally, application of this engine to a plant-level event tree/ fault tree model is presented, with results of MCS and quantification, post-process, importance & sensitivity analysis and uncertainty analysis.

I. INTRODUCTION

Fault Tree Analysis (FTA) is a prevailing method used in Probabilistic Safety Analysis PSA to derive risk measures for Nuclear Power Plant. Most of the quantification engines used for PSA models are based on traditional FTA algorithm. Usually, the traditional method will result in good approximation when the probability of basic event is relatively low. However, the error to the exact value will become larger or even unacceptable when the basic event's unavailability is increased to a certain value (e.g. an order of 10^{-1}), or under some particular logic structures. To solve this deficiency, the Binary Decision Diagram (BDD) for FTA was introduced in the 1990s.

BDD algorithm will derive the exact value of PSA model based on its inherent feature of analytically encoding fault tree structure (i.e. Boolean formula) in a binary tree structure, which could also reduce the time consumed in minimum cut-sets (MCS) solving and quantification. However, large efforts may have to be taken when a fault tree model is converted to its binary tree structure, since the efficiency of conversion depends on the basic events ordering scheme and it is proved that the complexity for optimizing the order is NP-complete[1]. As a result, the time and memory space consumed by a quantification engine using BDD algorithm may be much higher than the traditional engine when solving large PSA models, which often counteracts the benefit of accuracy brought by this new algorithm. Although with this conversion problem, the BDD algorithm is still a state of the art technique for solving PSA models with basic events of large unavailability, such as the Seismic PSA model, and it has already been integrated in several quantification engines[2].

As binary tree's advantage in fault tree logic storage, it is still an appealing way for quantifying fault tree model. Therefore, more studies have been focused on reducing the complexity of fault tree conversion. It is not until Zero-suppressed Binary Decision Diagram (ZBDD) algorithm was introduced that there begins a breakthrough in fault tree solving. ZBDD is also a data structure developed to compress the binary tree with high sparseness, and is introduced by Rauzy to encode MCS[4]. Then Woo went a further step by developing a set of formulas that allow using ZBDD structure in fault tree conversion, which efficiently reduces the consumption in time and memory space[5]. However, as the ZBDD structure simplifies the special branches in binary structure, it no longer stands for the exact Boolean formula of the fault tree, thus it reduces the accuracy of the quantification result. But this error is usually neglectable for most of the PSA models. In this case, ZBDD algorithm is still a most efficient way for FTA at present.

This paper describes the development of a FTA quantification engine based on BDD and ZBDD algorithm. This engine combines these two algorithms into one software, providing two alternative ways for large PSA model quantification. Then application of this engine to a plant-level event tree/ fault tree model is presented, with results of MCS and quantification, post-process, importance & sensitivity analysis and uncertainty analysis results.

II. BDD Algorithm

BDD structure encodes fault tree model in the form of binary tree. It can be interpreted by Shannon Theory[3] by assuming the Boolean expression of the fault tree logic is $f = f(x_1, x_2, x_3, \cdots; x_n)$. This formula is then expanded as

$$f(x_1, \cdots;x_{i-1},x_i,x_{i+1}, \cdots;x_n) = x_i f(x_1, \cdots;x_{i-1},1,x_{i+1}, \cdots;x_n) + x_i f(x_1, \cdots;x_{i-1},0,x_{i+1}, \cdots;x_n)$$

in which x_i stands for the basic event of fault tree, $f(x_1, \cdots;x_{i-1},1,x_{i+1}, \cdots;x_n)$ and $f(x_1, \cdots;x_{i-1},0,x_{i+1}, \cdots;x_n)$ are defined as the 1-branch and 0-branch of f respectively. A binary tree structure is indicated in this formula. For example, let $f = f(x_1,x_2,x_3, \cdots;x_n)$, then f can be expanded to the following expression in the order of $x_1<x_2<x_3$. The symbol "<" defines the priority of the variables in the Shannon decomposition process.

$$f = f(x_1,x_2,x_3) = x_1 f(1,x_2,x_3) + \bar{x}_1 f(0,x_2,x_3)$$

The binary tree structure is shown in figure 1(a) with 1-branch and 0-branch labeled as F_{11} and F_{10}. The two edges pointing to 1 and 0-branch are represented by a solid line and a dotted line respectively. Eventually the complete Shannon expression of f is shown in figure 1(b) by further expanding F_{11} and F_{10} with the same procedure.

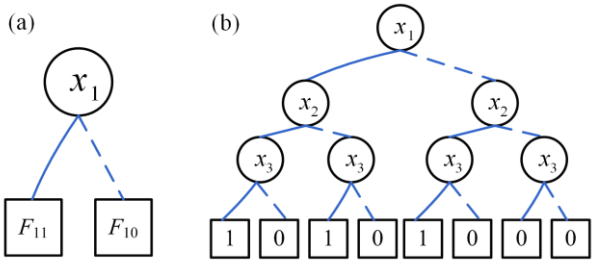

Figure.1. Binary Tree of f = f(x1,x2,x3)=(x1+x2)x3
(a) after 1st Shannon Decomposition (b) Final Binary Tree

For figure 1(b), x_1 is defined as the root node, 0 and 1 are defined as the end nodes, while x_2, x_3 are intermediate nodes. It should be noted that the variable order will greatly affect the size of the binary tree, hence order optimization is expected to be implemented prior to fault tree conversion. In addition, simplification rules should also be applied to the conversion to reduce the size of binary tree.

Then the fault tree can be converted to a binary decision diagram based on a pre-defined basic event order. Basic event conversion is usually the start point of the entire fault tree conversion. For example, basic event *A* is converted to a binary tree with root node *A* and two end nodes 0 and 1. The conversion process goes upwards from the basic event to the top event of the fault tree.

An operator called *ite* is developed in convenience of fault tree conversion. Therefore, the Shannon expansion can be expressed as

$$f = \text{ite}(x_i, G, H)$$

in which $G = f(x_1, \cdots; x_{i-1},1,x_{i+1}, \cdots; x_n)$, $H = f(x_1, \cdots; x_{i-1},0,x_{i+1}, \cdots; x_n)$. The implication of operator *ite* is that if $x_i=1$, then $f=G$, otherwise $f=H$. G and H stands for the 1 and 0-branch of f. Each basic event can be treated as an *ite* operator when converting the fault tree. Taking basic event *A* as an example, the *ite* form of *A* is ite(*A*,1,0). As a result, a certain fault tree can be expressed as a set of *ite* operator with the same logic combination as that in the fault tree. For example, a fault tree shown in fig.2 can be converted to its *ite* form as the following expression.

$$T4=[\text{ite}(A,1,0)+\text{ite}(B,1,0)]\ [\text{ite}(B,1,0)+\text{ite}(C,1,0)]\ [\text{ite}(C,1,0)+\text{ite}(D,1,0)] \quad (1)$$

in which, *A*=PM01AER, *B*=ECES1, *C*=PM01BER, *D*=ECES2.

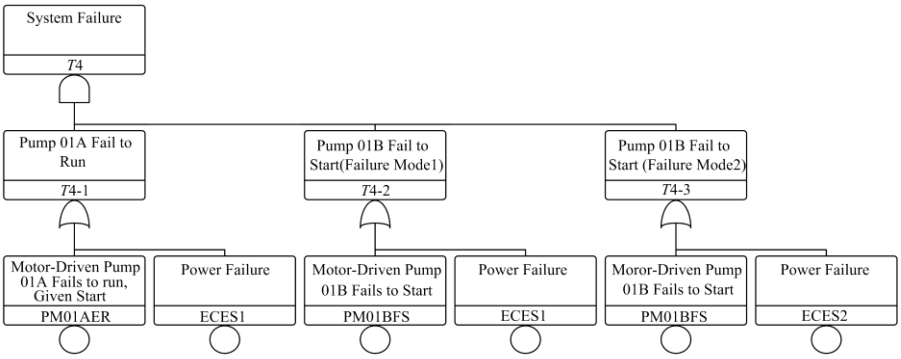

Fig.2. T4 fault tree

The following theorem is applied to the OR and AND logic in Expression (1): Let x, y be two basic events with an order of $x<y$, and $G1$, $G2$, $H1$, $H2$ be four Boolean expressions, $\diamond$ be any kind of a Boolean operation (OR, AND, XOR, etc.), then the following equations hold[4]:

$$\begin{aligned} &\text{ite}(x,G1,G2)\diamond\text{ite}(x,H1,H2)=\text{ite}(x,G1\diamond H1,G2\diamond H2)\\ &\text{ite}(x,G1,G2)\diamond\text{ite}(y,H1,H2)=\text{ite}(x,G1\diamond\text{ite}(y,H1,H2),G2\diamond\text{ite}(y,H1,H2)) \end{aligned} \quad (2)$$

Based on this set of equations, Expression (1) is equivalent to

$$T4=\text{ite}(A,\text{ite}(B,\text{ite}(C,1,\text{ite}(D,1,0)),\text{ite}(C,1,0)),$$
$$\text{ite}(B,\text{ite}(C,1,\text{ite}(D,1,0)),0))$$

Then the BDD form of $T4$ can be depicted as shown in Fig. 3 based on this expression.

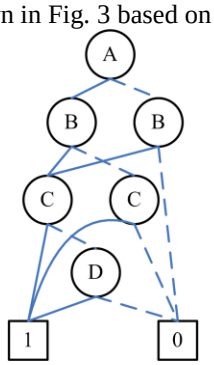

Fig. 3. BDD form of f=(A+B)(B+C)(C+D)

Fig.3 essentially encodes the truth table of *f*. Therefore, the exact probability can be derived by assigning unavailability of these basic events.

MCS of *f* can be derived from Fig.3[4]. Let F be the BDD form of $f=f(x_1,x_2,x_3,...,x_n)$, and each path from the root to the end node 1 be a solution of *f*, defined as σ. Then {*A*,*B*,*C*}, {*A*,*B*,*D*}, {*A*,*C*}, {*B*,*C*}, {*B*,*D*} are the whole solutions of *f*. Not all of these solutions are minimal. For example, {*A*,*C*} is the subset of {*A*,*B*,*C*}, hence {*A*,*B*,*C*} should be excluded from the final MCS. This indicates that if σ is a minimal solution of *f*, then there is only one path from the root of *F* to the end node 1 defining σ.

Based on this indication, MCS of *f* can be derived by switching the end node of each non-minimal solution from 1 to 0, which results in a BDD structure with large sparseness. This sparseness leads to the use of ZBDD structure to encode the MCS. Several simplification rules are applied to the BDD encoded MCS by removing the intermediate nodes with their 1-branches pointing to the end node 0, and connecting their father nodes with the end node. The ZBDD encoded MCS of *f* is shown in Fig.4.

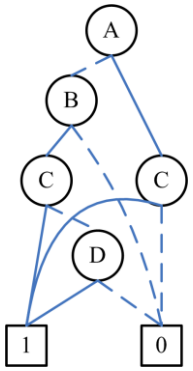

Fig. 4. ZBDD form of f.

Solutions of {*B*,*C*}, {*A*,*C*}, {*B*,*D*} can be derived from this structure to compose the MCS of *f*.

III. ZBDD Algorithm

As ZBDD structure's benefit in encoding MCS, the idea of ZBDD algorithm is to convert fault tree in the form of ZBDD structure to reduce the large consumption of time and memory resulted from the intermediate converting phase from fault tree to BDD. ZBDD algorithm is also based on the Shannon Theory, while a slight change is applied when processing with the 0-

branch, which is represented in the following expression[5]

$$f(x_1, \cdots;x_{i-1},x_i,x_{i+1}, \cdots;x_n) = x_i f(x_1, \cdots;x_{i-1},1,x_{i+1}, \cdots;x_n) + f(x_1, \cdots;x_{i-1},0,x_{i+1}, \cdots;x_n)$$

Then the operation rules of Boolean expression in Equations (2) can be changed accordingly[5],

$$ite(x,G1,G2)\cdot ite(x,H1,H2) = ite(x,G1H1+G1H2+G2H1,G2H2)$$

$$ite(x,G1,G2)+ite(x,H1,H2) = ite(x,G1+H1,G2+H2)$$

$$ite(x,G1,G2)\cdot ite(y,H1,H2) = ite(x,G1ite(y,H1,H2),G2ite(y,H1,H2))$$

$$ite(x,G1,G2)+ite(y,H1,H2) = ite(x,G1,G2+ite(y,H1,H2))$$

in which x, y, *G*1, *G*2, *H*1, *H*2 are the same as in Equations (2).

However, as the ZBDD structure no longer holds the truth table of Boolean expression, only approximation results can be derived.

IV. Development of Quantification Engine

A quantification engine was developed based on the BDD and ZBDD algorithm. This chapter mainly describes the compiling process of this engine. As aimed for a commercial software, this engine, coded in C++ language, was developed through a rigorous QA procedure. Several modules compose this engine, as shown in the flowchart of Fig.5.

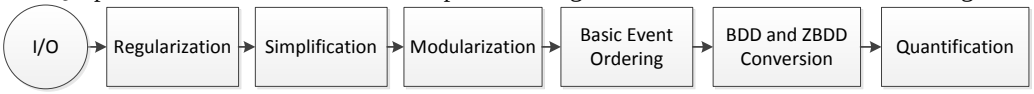

Fig.5 Flowchart of the Engine

An important part of a newly developed quantification engine is to be compatible with several prevailing commercial PSA modeling software so as to read the event tree and fault tree structure directly from these model files and to compare the results. RiskSpectrum PSA and CAFTA were selected as two typical codes for their larger user group worldwide. An input/output (I/O) module was first developed to interface with these two codes to read PSA models as input. This input module consists of two basic functions which are to convert the event tree, if any, to the fault tree structure, and to read the fault tree structure into the memory in the form of a certain data structure, providing input information for BDD or ZBDD conversion and quantification. Risk measures including top event unavailability, MCS, importance and sensitivity analysis, uncertainty analysis will be written in text files through the output module.

Models for PSA are not always built in the regular form of Boolean logic, namely the OR, AND and negation. For instance, in RiskSpectrum PSA, a typical PSA model for nuclear plant may have contained non-regular logic mainly to facilitate the modeling process, such as K/N and XOR gate, as well as special events like house event, exchange event, common cause failure (CCF) event, events with definite state, boundary condition set and continuous gate. All of these logic shall be regularized to normal ones. Unfortunately, most of these regularization processes will inevitably increase the final size of the fault tree, as well as the work in the next phase.

The methodology of fault tree simplification and modularization were well established owing to the development of tradition quantification engine through decades. The FAUNET method[6] and Linear Time Algorithm (LTA)[7] were both updated and incorporated into this Engine. For FAUNET-based method, sub-processes are included, namely the gate merging, node extraction, branch compressing and factorization. The LTA method runs faster in modularization. However, it does not simplify the fault tree, hence increasing the work for BDD and ZBDD conversion. In addition, the number of modules produced is greatly dependent on the correlation between basic events. Table 1 shows some statistics to characterize the modularization efficiency of these two methods, by using this Engine to calculate two different models. The time cost in modularization, modules produced, as well as time cost in ZBDD conversion are listed, with the modularization efficiency defined as the rate of total time cost to the number of modules.

TABLE I. Modularization Efficiency of FAUNET-based and LTA-based methods

	Size	Modularization Time T1 (ms)		Number of Modules N		ZBDD Conversion Time T2 (ms)		Efficiency (=(T1+T2)/N)	
		(1)	(2)	(1)	(2)	(1)	(2)	(1)	(2)
Model 1	BE: 1757 OR: 5484 AND: 2720 NOR: 357	25423	144	435	121	69971	336517	219.30	2782.32
Model 2	BE: 663 OR: 436 AND: 107	52	2	155	13	200	4102	1.62	315.69

(1) FAUNET-based method
(2) LTA-based method
CPU: Xeon X5690 3.46GHz, 4GB RAM

As can be seen from Table 1, LTA-based method has a higher module production rate (modules generated per ms) than that of FAUNET-based method, but is far less efficient if comparing the total time cost per module. Therefore, for nuclear plant PSA models which usually consist of basic events with strong dependency, this Engine chooses FAUNET-based method as default.

The fault tree reading, regularization, simplification and modularization discussed above essentially constitute the preprocessing of PSA model. The fault tree structure processed through these modules will be used for BDD or ZBDD conversion.

Basic event ordering is a significant part of this engine. Usually, the ordering scheme can be divided into static ordering and dynamic ordering. Both of these two schemes contain several specific ordering methods. Some of these have been integrated into this Engine, such as depth first left most (DFLM), ordering by weight, FORCE, etc. For most models, dynamic ordering scheme takes more time than static ordering, while having a better efficiency on BDD or ZBDD conversion. DFLM is selected as the default ordering scheme for this Engine regarding the balance between time cost and accuracy. The user could also choose specific ordering method manually.

The BDD and ZBDD conversion is the core process of this Engine, which determines the accuracy of the risk measures and the source cost (time and memory space). Algorithms introduced in part 2 and 3 of this paper were coded in this Engine. A well-defined data structure and use of Hash table are significant to the efficiency during converting. Chain table was used to represent the *ite* nodes of the converted fault tree. Most of the fault trees will result in large amount of binary tree nodes when converted to BDD or ZBDD structure, making node searching become the most important operation. Hash table is the best way for node searching. Three hash tables were compiled into this Engine, namely the node table, *ite* table, and *without* table, which are used for storing relationship between fault tree node to binary tree node, *ite* operation to binary tree node and *without* operation (used to generate MCS[4]) to binary tree node. It should be noted that for large PSA models, huge hash tables will nearly always be produced along with the conversion, and most of the element in hash tables may never be searched. In this case, an optimization algorithm should be invented to reduce the size of the hash tables.

For large PSA models, usually it is not possible to derive the results in an acceptable time and memory cost. The truncation process is essential for these models. Both probabilistic and cut-set number truncation methods are compiled into this Engine.

With BDD or ZBDD encoded fault tree been built, the quantification result and MCS can be derived by traversing the binary tree. Also, the MCS containing distinct initiating event (or frequency type event) will be added up separately.

The de-modularization process is also necessary for the modularized cut-sets. A recursive algorithm is needed when de-modularizing these cut-sets, in case that sub-modules are contained in modules. In addition, optimization algorithm should also be invented to prevent data explosion when de-modularizing some extremely modularized cut-sets.

For MCS post-processing, this Engine reads rules from the model file or flag file, and apply them to the MCS-encoded ZBDD or the MCS list.

Importance and sensitivity analysis, as well as uncertainty analysis will both be calculated through the MCS-encoded ZBDD or MCS list, depend on the user configuration. Monte Carlo sampling method is used for uncertainty analysis.

All of the modules discussed above were combined together to compile the complete quantification engine. Benchmark models and several nuclear plant PSA models were tested. The results are shown in part 5 of this paper with comparison to RiskSpectrum.

V. Tests and Application

Benchmark and nuclear plant models were used for test of this Engine. Table 2 shows the risk measures, time cost, and comparisons with RiskSpectrum. As for benchmark models, probability of 0.001 is assigned to each basic event's unavailability. Probabilistic truncation was applied to some large PSA models (NPP1 & NPP2). The test environment is the same with the information listed below Table I.

TABLE II. Test Results of the Engine

Model	Truncation(1)	Engine in this paper			Exact Value(2)	Relative Error	RiskSpectrum PSA		
		Top Event Unavailability	MCS Size	Time Cost (ms)			Top Event Unavailability	MCS Size	Time Cost (ms)
edf9202	0	1.30483E-01	130,112	237	1.30483E-01	0.00%	1.30530E-01	130,112	320
edf9203	0	4.39226E-02	20,807,446	21,257	4.39226E-02	0.00%	4.45810E-02	327,178(3)	21,880
edf9204	0	3.83362E-02	32,580,630	103,460	3.83362E-02	0.00%	3.87670E-02	137,062(3)	145,090
cea9601	0	1.18262E-06	130,281,976	230,856	1.18262E-06	0.00%	1.19654E-06	1,615,876(3)	25,204,100
edfpa14o	0	2.03314E-02	105,927,244	123,375	2.03314E-02	0.00%	2.03890E-02	110,308(3)	52,980
NPP1	1E-15	1.64956E-07	55,875	78666	NA	NA	1.65088E-07	55,875	79,820
NPP2	1E-15	1.74691E-07	91053	12,944	NA	NA	1.74684E-07	91060	12,750

(1) If truncation is 0, it indicates that BDD conversion is applied, resulting in the exact value, otherwise ZBDD conversion is applied.
(2) The exact value comes from reference [8]
An automatic truncation occurred

As can be seen from Table II, the benchmark results derived from this Engine matched exactly with the exact value, while spending a shorter or equivalent calculation time as compared to the time cost in RiskSpectrum PSA. For NPP1 and NPP2, results with a 1E-15 probability truncation were compared to those of RiskSpectrum PSA. The two engines yielded the same size of MCS, and a slight deviation in the top event's unavailability due to the different quantification mechanism. In addition, the time cost of this Engine is less than or equivalent to RiskSpectrum PSA.

This Engine was also used for current engineering project carried out in SNERDI. The risk measures are shown in Table III, Table IV and Fig.6.

TABLE III. MCS Results after Post-processing*

No.	Unavailability	Basic Event 1	Basic Event 2	Basic Event 3
1	1.80E-08	IEV-SLOCA	REN-ST-GP	
2	1.50E-08	MLOCAFG	REN-ST-GP	
3	1.00E-08	IEV-RV-RP		
4	6.48E-09	LLOCAFG	ACACV028GO	
5	6.48E-09	LLOCAFG	ACACV029GO	
6	6.48E-09	LLOCAFG	ACBCV028GO	
7	6.48E-09	LLOCAFG	ACBCV029GO	
8	3.80E-09	IEV-SLOCA	CCX_PMS_CI	REC-MANDAS
9	3.69E-09	IEV-SI-LB-B	REN-ST-GP	
10	3.51E-09	IEV-SI-LB-A	REN-ST-GP	

Only the top 10 MCS are listed.

TABLE IV. Importance and Sensitivity Analysis Results*

No.	Basic Event	Unavailability	FV	RAW	RRW
1	REN-ST-GP	3.00E-05	3.07E-01	1.02E+04	1.44E+00
2	REC-MANDAS	8.62E-02	1.86E-01	2.97E+00	1.23E+00
3	CCX_PMS_CI	7.34E-05	7.58E-02	1.03E+03	1.08E+00
4	CCX_PMS_SRNC	4.82E-05	4.98E-02	1.03E+03	1.05E+00
5	ACACV029GO	1.30E-03	4.05E-02	3.21E+01	1.04E+00
6	ACACV028GO	1.30E-03	4.05E-02	3.21E+01	1.04E+00
7	ACBCV029GO	1.30E-03	4.05E-02	3.20E+01	1.04E+00
8	ACBCV028GO	1.30E-03	4.05E-02	3.20E+01	1.04E+00
9	CCX-SFTW-PMS	1.14E-05	2.63E-02	2.31E+03	1.03E+00
10	OTH-PRSOV	1.00E-02	2.41E-02	3.39E+00	1.03E+00

Only the top 10 basic events are listed.

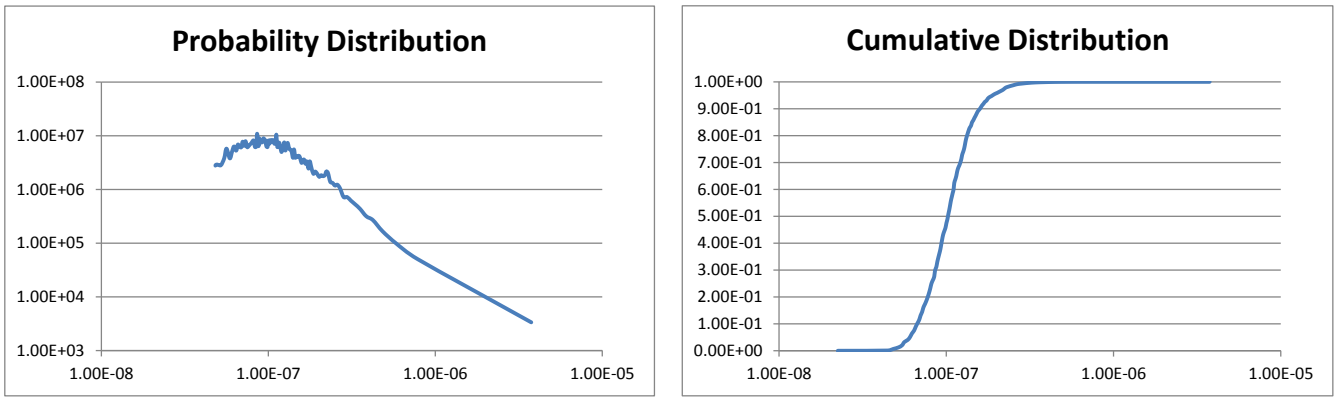

Fig. 6. Uncertainty Analysis Results

VI. CONCLUSIONS

The development of an advanced fault tree quantification engine is introduced in this paper. BDD and ZBDD algorithms are compiled into this Engine together with the I/O modules and fault tree pre-processing modules. Comprehensive tests have been conducted to this Engine. The results indicate that this Engine is efficient in quantifying large PSA models and has been used in several PSA project for nuclear power plants. This Engine is also compatible with RiskSpectrum PSA model, and can be run in Windows and Linux 32 or 64bit System.

REFERENCES

1. Bollig B, Wegener I. Improving the variable ordering of OBDD is NP-complete[J]. IEEE Trans. Computers, 1996, 45(9): 993–1002.
2. EPRI, Seismic Probabilistic Risk Assessment Implementation Guide, EPRI, Palo Alto, CA: 2013. 3002000709: 5-120.
3. GU Tianlong, XU Zhoubo. Ordered binary decision diagram and its application[M]. Beijing: Science Press, 2009.
4. Rauzy A. New algorithms for fault trees analysis[J]. Reliability Engineering and System Safety, 1992.
5. Woo Sik Jung, ZBDD algorithm features for an efficient Probabilistic Safety Assessment, Nuclear Engineering and Design, 2009.
6. A.R.Karen, Effiecient Fault Tree Analysis Using Binary Decision Diagrams, 2002.9.
7. Y.Dutuit, A.Rauzy, A Linear-Time Algorithm to Find Modules of Fault Trees, 1996.3.
8. Woo Sik Jung, Fast BDD truncation method for efficient top event probability calculation. Nuclear Engineering and Technology, 2008.