Monte-Carlo Simulation to Propagate Uncertainties in Fault Trees encoded by means of Binary Decision Diagrams[*]

Yves Dutuit
LADS – Université Bordeaux I,
351, cours de la Libération
33405 Talence cedex, France
dutuit@iuta.u-bordeaux.fr

Antoine Rauzy
LaBRI Université Bordeaux I,
351, cours de la Libération
33405 Talence cedex, France
rauzy@labri.u-bordeaux.fr

J.-P. Signoret
Direction Sécurité Environnement,
Elf Aquitaine Production
64018 Pau Cedex, FRANCE,
Signoret_Jean-Pierre/pau@com-pau.elf-p.fr

Abstract

Since their introduction in the reliability analysis framework, Binary Decision Diagrams have proved to be the most efficient technique to assess fault trees. With BDDs, not only the computation of the probability of the top event is accelerated (up to factor 1000) but also the obtained results are exact. This increases dramatically the interest of Monte-Carlo simulation in order to study how uncertainties on parameters of probability laws of terminal events propagate through the tree. The aim of this paper is to present algorithms to do so and practical results we got on medium-size fault trees.

1 Introduction

Binary Decision Diagrams are the state-of-the-art data structure to encode and manipulate boolean functions. They have been introduced by Akers [Ake78] and improved by Bryant [Bry86, BRB90]. They are nowadays used in a wide range of areas (see [Bry92] for a survey of BDDs and their applications). It has been shown that BDDs can be use to compute very efficiently the exact probability of the top event of a fault tree from the probabilities of its primary events [Rau93, Kri93, CM94, SA95, SA96]. Namely, such an evaluation is linear in the size of the BDD [Rau93, Kri93].

It is sometimes of interest to study the relative importance of failures of the various components a system is made of. Technically, this involves the computation, for each component under study, of the conditional probabilities $p(f = 1|g = 1)$ and $p(g = 1|f = 1)$, where f and g denote respectively the Boolean functions describing the failures of the whole system

[*]This works has been partly supported by the Aralia project that is a partnership between our two laboratories — the Laboratoire d'Analyse des Dysfonctionnements des Systèmes (LADS) and the Laboratoire Bordelais de Recherche en Informatique (LaBRI) — and the following compagnies: Commissariat à L'Énergie Atomique (LETI), Dassault Aviation, Électricité de France, Elf Aquitaine, Institut de Protection et de Sûreté Nucléaire, Renault, Schneider Electric, Société Générale pour les techniques Nouvelles (COGEMA) and Technicatome.

and the component from the failures of basic components. In this paper, we propose two BDD algorithms to do so. The first one is simply derived from the well known equality $p(f = 1|g = 1) = p(f = 1 \wedge g = 1)/p(g = 1)$. The second is one is dedicated to the case where g is a terminal event (and is indeed more efficient than the former). By the way, we are able to compute Vesely-Fussel's importance factors [Fus75] from which many other importance factors can derived [PG80, Vil88].

We then propose a technique to "compile" BDDs in order to achieve a significant improvement of the efficiency of probability computations. It consists basically in duplicating BDDs into arrays, so-called Binary Decision Arrays.

In general, probabilities of terminal events of fault trees vary through the time and are given as probability laws. The well known exponential law $F(t) = 1 - e^{-\lambda t}$ is one of the most popular examples of such laws. Often, the paramaters of the laws are known only up to an uncertainty, i.e. that they themselves vary under certain laws. For instance, the lognormal law is often chosen as a model of the variation for the paramater λ of the exponential law.

So, it is of interest to perform Monte-Carlo simulations to have a more precise idea of how this uncertainty on probabilities of terminal events propagates through the tree. Until the introduction of BDDs in the reliability analysis framework, such computations were limited to very small trees because of the cost of the basic computation: assessing the probability of the top event. Moreover, the obtained results were only approximated ones. With BDDs, not only this basic computation is accelerated (up to factor 1000) but also the obtained results are exact. This increases dramatically the interest of Monte-Carlo simulation on Boolean models of industrial systems. The aim of the paper is thus clear. We do not present new mathematical results but we show by means of a number of examples that improvements on efficiency of calculations produce a qualitative jump in what can be done for the treatment of uncertainties. This holds not only for probability assessment, but also for importance factors assessment.

2 Propogation of Uncertainties through Fault Trees

In a probabilistic safety study for any complex system, a failure of which may have significant consequences, an important topic is the treatment and analysis of the uncertainties involved. One of them is the parametric uncertainty which causes the end result, for instance the fault tree top event probability, to become a distributed variable. This can be due to factors such as the statistical uncertainty due to the fact that only small component test data are available or because the data evaluation requires subjective interpretations of failure data.

Usually, the approach of the propagation by means of Monte-Carlo simulation is used to analyze such a parametric uncertainty. The Monte-Carlo method is widely applicable, but it requires a mathematical model from the system and a large amount of computer time to reduce statistical fluctuations due to finite number of trials.

Let us put that in a more formal way. A fault tree defines the probability P of occurrence of the event of interest as a function of the parameters X_i that determine the probabilities of existence of the basic events. Thus, one can write $P = f(X_1, \ldots, X_n)$. For the components of interest in the quantitative risk assessment of complex systems, the parameters X_i have large uncertainties associated with them. In this paper, we consider that the X_i's vary through the time according to exponential laws of the form: $X_i = 1 - e^{-\lambda_i \cdot t}$. Thus, the uncertainties stand on the λ_i's. We assume that the λ_i's are distributed according to lognormal laws [PG80], and that these laws are given by their mean values and their error factors.

So, each trial of a Monte-Carlo simulation is performed as follows.

1. The λ_i's are drawn at random according to a lognormal distribution (see [PTVF92] for more details on this process).

2. For each instant t of interest, first the X_i's are determined, and second P is computed.

This gives a sample from which several statistical parameters of interest can be computed: mean, standard deviation, 90% confidence range, ...

In order to make such simulations meaningful, two conditions must be fulfilled. First, the number of trials has to be sufficiently large, to reduce statistical fluctuations. Second, the assessment of P from the X_i's must be precise, for obvious reasons. This is the reason why BDDs outperform definitely other knowm methods: as we will see in the next sections, not only they make the computation of P from the X_i's a very efficient process, but the obtained values are exact.

3 Binary Decision Diagrams

To start with, we define Binary Decision Diagrams (BDDs) as introduced in [Bry86] (and further improved in [BRB90, Bry92]).

Definition A BDD is a rooted directed graph with node set V containing two types of nodes. A non terminal node v has as attributes an index $if(v) \in \{0, \ldots, n\}$ and two children $then(v)$ and $else(v) \in V$. A terminal node has as attribute a Boolean value $value(v) \in \{0, 1\}$. In addition, for any non terminal node v, if $then(v)$ (resp. $else(v)$) is a non terminal node, then $if(v) < if(then(v))$ (resp. $if(v) < if(else(v))$). A BDD having this property is said to be ordered. A BDD is reduced if it contains no node v with $then(v) = else(v)$, nor does it contain distinct nodes v and w such that the BDDs rooted by v and w are isomorphic. In this paper, we consider only reduced ordered BDDs. We denote respectively by 0, 1 and $\Delta(i, v_1, v_0)$ the two terminal nodes whose values are 0 and 1 and a non terminal node v with $if(v) = i$, $then(v) = v_1$ and $else(v) = v_0$.

BDDs are mainly designed for the description of Boolean functions. The function $Shannon[\![.]\!]$, that associates a Boolean function $\{0, 1\}^n \to \{0, 1\}$, over the variables $x_1, \ldots, x_n$ to each BDD, is defined by the following equations.

$$\begin{aligned} Shannon[\![0]\!] &\overset{\text{def}}{=} 0 \\ Shannon[\![1]\!] &\overset{\text{def}}{=} 1 \\ Shannon[\![\Delta(i, v_1, v_0)]\!] &\overset{\text{def}}{=} (x_i \wedge Shannon[\![v_1]\!]) \vee (\neg x_i \wedge Shannon[\![v_0]\!]) \end{aligned}$$

For a given variable ordering, the BDD representation of Boolean functions is unique, up to an isomorphism. Moreover, it can be built efficiently (see [Bry86, BRB90]).

Probability The Boolean variables $x_1, \ldots, x_n$ can be also interpreted as independent random variables defined on the elementary events of the probability space $\Omega = \{0, 1\}$. The probability distribution that defines the probability that such a random variable has the value

1 is denoted by $P_r(x_i = 1)$ and the probability that a Boolean function f built over the x_i's yields the value 1 for a random argument depending on P_r is denoted by $P_r(f = 1)$.

The point is that $P_r(f = 1)$ can be efficiently assessed when f is encoded by means of a BDD [Rau93, Kri93]. The function $P_r[\![.]\!]$ that computes $P_r(f = 1)$ from the $P_r(x_i = 1)$'s is defined by the following equations.

$$\begin{aligned} P_r[\![0]\!] &\overset{\text{def}}{=} 0 \\ P_r[\![1]\!] &\overset{\text{def}}{=} 1 \\ P_r[\![\Delta(i, v_1, v_0)]\!] &\overset{\text{def}}{=} (P_r(x_i = 1).P_r[\![v_1]\!]) + ((1 - P_r(x_i = 1)).P_r[\![v_0]\!]) \end{aligned}$$

From an operational point of view, the efficiency of the function P_r comes from the fact that intermediate results are memorized into a hashcache (a hashtable without collision chain). This table contains pairs (v, p) where v is the address of a BDD node and p is the probability of the function encoded by the BDD rooted by v. Before computing $P_r[\![v]\!]$, one consults first the table. If it contains already a pair (v, p) then p is returned. Otherwise, $P_r[\![v]\!]$ is actually computed and then the pair $(v, P_r[\![v]\!])$ is inserted into the table. If during the process of computing the probability of the function f encoded by the BDD F, there is no two BDD nodes v and w of F with the same hashcode, the computational complexity of $P_r[\![.]\!]$ is linear in the number of nodes of F.

Conditional Probabilities, Importance Factors Recall that the Vesely-Fussel's importance factor of a component whose faults are described by a Boolean function g is defined as the conditional probability $P_r(g = 1|f = 1)$, where f is the fault tree describing the system under study. The only way to assess such a conditional probability is to use the well known equality :

$$P_r(g = 1|f = 1) = \frac{P_r(g \wedge f = 1)}{P_r(f = 1)} \tag{1}$$

So, the process consists first in computing the BDDs encoding f and $g \wedge f$, and second in assessing, from these BDDs, the two right hand side probabilities. This is however othen the case that g is reduced to a single variable x_j, or in other words that the component whose relative importance is to be assessed is a basic one. In such a situation, the computation of the BDD encoding $x_j \wedge f$ can be avoided by assessing $P_r(x_j \wedge f = 1)$ directly from the BDD encoding f. This just requires to modify the equations as indicated below.

$$\begin{aligned} P_r[\![0|j]\!] &\overset{\text{def}}{=} 0 \\ P_r[\![1|j]\!] &\overset{\text{def}}{=} 1 \\ P_r[\![\Delta(j, v_1, v_0)|j]\!] &\overset{\text{def}}{=} P_r[\![v_1|j]\!] \\ P_r[\![\Delta(i, v_1, v_0)|j]\!] &\overset{\text{def}}{=} (P_r(x_i = 1).P_r[\![v_1|j]\!]) + ((1 - P_r(x_i = 1)).P_r[\![v_0|j]\!]) \end{aligned}$$

Note that this can be extended to the case where g is a conjunction of literals. In practice, it appears that this way of assessing $P_r(x_j \wedge f = 1)$ is more efficient that the first one.

4 Binary Decision Arrays

As said in the previous section, the efficiency of the procedure $P_r[\![.]\!]$ comes from the memorization of intermediate results. Several remarks can be done about this process.

1. The BDD encoding a formula is synthetized from the BDDs encoding its sub-formulae. As a consequence, its nodes may be scattered in the computer memory, especially when it is large. This means that the procedure requires, when applied on large BDDs, a number of changes of the current memory page, which is costly.

2. The memorization is implemented by means of a hashcache, because a full hashtable would be too expensive to manage. But this means that the memorization is not perfect. Two nodes may have the same hashcode, especially when the BDD is large. Moreover, looking up to the hashcache is a rather costly process.

Indeed, the two above remarks stand at a very low level and have nothing to do with mathematical or even algorithmic aspects of the problem. However, computers are not mathematical objects and there is a long road from algorithms to programs. Any improvement, even by a small constant factor, of the implementation of $P_r[\![.]\!]$ is good to take. Recall that this procedure should be applied very large number of times in order to perform Monte-Carlo Simulations.

The idea is thus to duplicate temporarily the BDD F under study into an array T of BDD nodes. T is as large as F. The nodes in T are arranged in such way that if a node v is a descendant of a node w in F, then the copy of v in T is located at lower index than the copy of w. In addition, each node of T contains a cell to store a probability (note that it would be too expensive to add such a cell to all of the BDD nodes). The implementation of $P_r[\![.]\!]$ on such an array T consists in traversing T from the first node to the one of highest index. By construction, when the node $w = \Delta(i, w_1, w_0)$ is examined, the probabilities of w_1 and w_0 are already computed and stored in the corresponding cells. This implementation is more efficient than the regular one for the following reasons.

1. The recursive calls to the procedure are replaced by a traversal of the array.

2. Nodes of T are less scattered into the computer memory than those of F.

3. The memorization of intermediate results becomes a perfect and very cheap process, for it does not involve hasching techniques.

An important point is that T can be built from F in linear time. The algorithm works in two passes. First, F is traversed in a depht-first way and nodes of F are copied into T in post-order. This means that to copy the BDD rooted by the node v one first copies the nodes of the BDD rooted by $then(v)$, then those of the BDD rooted by $else(v)$, and finally the node v is itself copied. During this process, $then(v)$ is set to the copy w of v, $then(w)$ and $else(w)$ are respectively set to v and the previous value of $then(v)$. Then, T is traversed from the node of highest index to the first node. For each node w of T, one gets the corresponding node v of F (the address of which is stored in $then(w)$). Then, one restores $then(v)$ to its previous value (which is stored in $else(w)$). Finally, one sets $if(w)$, $then(w)$ and $else(w)$ respectively to $if(w)$, $then(then(v))$ and $then(else(v))$.

5 Experimental Results

In order to illustrate the approach described in this paper, we consider the fault trees `baobab1`, `baobab2` and `baobab3` of the Aralia benchmark[1]. The following table reports the numbers of variables and gates these trees are made of, their numbers of prime implicants, the number of nodes of the BDDs encoding them, the running times to build these BDDs and the corresponding BDAs. The running times indicated (in seconds) below were obtained on a PC 486/33 with a 8 megabytes memory.

Names	#var.	#gates	#PI	$\|BDD\|$	BDD	BDA
`baobab1`	61	84 (16 $\wedge$, 59 $\vee$, 9 k-out-of-n)	46188	7362	0.66s	0.09s
`baobab2`	32	40 (5 $\wedge$, 29 $\vee$, 6 k-out-of-n	4805	197	0.34s	0.01s
`baobab3`	80	107 (46 $\wedge$, 61 $\vee$)	24386	11789	1.07s	0.18s

In this paper, we are not concerned with numerical values, but with the running times to get these values. So, in order to test the algorithms presented above, we considered that the probability of each terminal event follows an exponential law of parameter λ and that this parameter λ is itself distributed according to a lognormal law of mean value 10^{-3} and error factor 3. The table below reports running times to assess the probability of the top event of each tree by means of Monte-Carlo simulations made of 10, 100 and 1000 trials. These running times include the computation of the mean value, the standard deviation and 90% confidence range for each of the simulations.

	$P_r[\![BDD]\!]$			$P_r[\![BDA]\!]$		
Names	10	100	1000	10	100	1000
`baobab1`	1.24s	11.65s	125.71s	0.30s	2.29s	25.39s
`baobab2`	0.06s	0.52s	5.03s	0.03s	0.25s	2.49s
`baobab3`	1.89s	21.41s	410.23s	0.45s	3.63s	60.72s

The table below reports running times to assess the Vesely-Fusel's importance factor of each terminal event by means of Monte-Carlo simulations made of 10 and 100 trials. As previously, these running times include the computation of the mean value, the standard deviation and 90% confidence range for each of the simulations.

	$P_r[\![BDD]\!]$		$P_r[\![BDA]\!]$	
Names	10	100	10	100
`baobab1`	68.22s	715.45s	14.00s	188.62s
`baobab2`	1.40s	11.20s	0.67s	3.95s
`baobab3`	159.41s	1989s	39.54s	294.35s

A number of conclusions can be drawn from the above results.

1. Once the BDD encoding the fault tree has been computed, the assessment of the (conditional) probability of its top event from the probabilities of its terminal events is efficiently achieved. None of the classical methods based on cutsets would be able to compute it so efficiently.

[1]The Aralia benchmark is constituted of real-life fault trees coming from various sources. It is freely available upon request to A. Rauzy (rauzy@labri.u-bordeaux.fr).

2. On the contrary to these methods, the obtained results are exact. This makes Monte-Carlo simulations significant.
3. When there is enough room in the computer memory, it is always interesting to compile the BDD into a BDA. This process takes very few time and it improves the efficiency of probability assessments by a factor five or even more.
4. It remains that Monte-Carlo Simulations are quite time consuming, and that any further improvement would be welcome in order to handle very large fault trees and/or to perform more trials per simulation.

6 Conclusion

In this paper, we presented BDD algorithms to perform Monte-Carlo simulations to assess various quantities of interest in fault tree analyses. We showed that BDDs make it feasible to study how uncertainties on parameters of probability laws of terminal events propagate through the fault trees under study. Running times required to assess medium size fault trees remain acceptable, even on a small personnal computer.

In should be noticed that all of the results reported in this paper were obtained with brute trees. It is well known that BDD algorithms can be made far more efficient by using good variable ordering heuristics and/or by rewriting the initial formula. Moreover, it has be shown recently, by A. Rauzy, that it is also of interest to build truncated BDDs (from which approximated probabilities can be computed).

It is clear today that the BDD technology outperforms the other known techniques, by several orders of magnitude. Moreover, the previous remarks show that it remains many things to do to improve it.

References

[Ake78] B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, 27(6):509–516, 1978.

[BRB90] K. Brace, R. Rudell, and R. Bryant. Efficient Implementation of a BDD Package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 40–45. IEEE 0738, 1990.

[Bry86] R. Bryant. Graph Based Algorithms for Boolean Fonction Manipulation. *IEEE Transactions on Computers*, 35(8):677–691, August 1986.

[Bry92] R. Bryant. Symbolic Boolean Manipulation with Ordered Binary Decision Diagrams. *ACM Computing Surveys*, 24:293–318, September 1992.

[CM94] O. Coudert and J.-C. Madre. MetaPrime: an Iteractive Fault Tree Analyser. *IEEE Transactions on Reliability*, 43(1):121–127, March 1994.

[Fus75] J.B. Fussel. How to hand-calculate system reliability characteristics. *IEEE Transactions on Reliability*, R-24(3), 1975.

[Kri93] R. Krieger. PLATO: A Tool for Computation of Exact Signal Probabilities. In *Proceedings of the VLSI Design Conf.*, pages 65–68, 1993.

[PG80] A. Pagès and M. Gondrand. *Fiabilité des systèmes*, volume 39 of *Collection de la Direction des Études et Recherches d'Électricité de France*. Eyrolles, 1980. ISSN 0399-4198.

[PTVF92] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Falnnery, editors. *Numerical Recipes in C: the Art of Scientific Computing*. Cambridge University Press, 1988-1992. ISBN 0-521-43108-5.

[Rau93] A. Rauzy. New Algorithms for Fault Trees Analysis. *Reliability Engineering & System Safety*, 05(59):203–211, 1993.

[SA95] R.M. Sinnamon and J.D. Andrews. New Approaches to Evaluating Fault Trees. In *Proceedings of European Safety and Reliability Association Conference, ESREL'95*, pages 241–254, June 1995.

[SA96] R.M. Sinnamon and J.D. Andrews. Fault Trees Analysis and Binary Decision Diagrams. In *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'96*, pages 215–222, 1996.

[Vil88] A. Villemeur. *Sûreté de fonctionnement des systèmes industriels*. Eyrolles, 1988.