A practical comparison of methods to assess sum-of-products

A. Rauzy, E. Châtelet, Y. Dutuit and C. Bérenguer,

Abstract

Many methods have been proposed in the literature to assess the probability of a sum of products. This problem has been shown computationaly hard (namely #P-hard). Therefore, algorithms to solve it can be compare only from a practical point of view.

In this article, we propose first an efficient implementation of the pivotal decomposition method. This kind of algorithms is widely used in the Artificial Intelligence framework. It is unfortunately almost never considered in the reliability engineering framework, but as a pedagogical tool. We report experimental results that shown that this method is in general much more efficient than classical methods that work by rewriting the sum of products under study into a sum of disjoint products.

Then, we derive from our method a factorization algorithm to be used as a pro-processing method for Binary Decision Diagrams. We show by means of experimental results that this latter technology outperforms the formers.

keywords: Sum-of-(Disjoints)-Products, Pivotal Decomposition, Binary Decision Diagrams.

1 Introduction

Many methods have been proposed in the literature to assess the probability of a sum of products or the probability that there exists a working st-path in a reliability network (see for instance [Shi91, Mis93, RVT95] for surveys on these methods). Several authors consider both problems at once by encoding the set of st-paths of a network as a sum of products. Both problems are computationaly hard (namely #P-hard). As a consequence, it is hopeless to find efficient algorithms to solve them (by efficient, we mean, as usual, of polynomial worst case complexity). Therefore, algorithms and heuristics cannot be compared from a general and theoretical viewpoint. They should be compared experimentally with respect to the following criteria:

- The form and the size of the results.
- The size of intermediate data structures.

- The computation time.

Most of the algorithms proposed up to now in the reliability engineering literature work by rewriting the sum of products into an equivalent sum of disjoint products, from which it is easy to compute the probability of the original formula (see [Mis93, RVT95] for a survey on these methods). Our experiments concern three of these methods, that are considered as among the most efficient ones: KDH [Hei89], DA [Bou93] and SODA [CDRB99].

In this article, we propose first an efficient implementation of the so-called pivotal decomposition method. This method works according to the following recursive principle. Given a formula F, either F is reduced to a constant or it is possible to select a pivot variable v and to study recursively the two formulae $F_{\overline{v}}$ and F_v, i.e. the formula F in which the constants respectively 0 and 1 have been substituted for the variable v. In other words, the algorithm builds, at least implicitly, a binary tree. Leaves of this tree encode constants. Internal nodes encode formulae of the form $F = v.F_v + \overline{v}.F_{\overline{v}}$. Branches of the tree that lead to a 1-leaf are labelled with the wanted disjoint products.

Pivotal decomposition procedures have been extensively studied in the automated reasonning and artificial intelligence literature (see e.g. [KL99]). Unfortunately, they are almost never considered in the reliability engineering framework, but as a pedagogical tool. Few articles however have considered methods derived from the factoring theorem in the case of reliability networks, e.g. [PP88, PP89]. This theorem is basically equivalent to the above decomposition. Despite of their apparent simplicity, pivotal decomposition procedures are interesting w.r.t. the three above criteria:

- They work in constant space (for they just need a backtracking stack): the probability of the original formula can be assessed on the fly, thanks to Shannon decomposition $p(F) = p(v).p(F_v) + [1 - p(v)].p(F_{\overline{v}})$.
- Branching heuristics help to keep as small as possible the sizes of the binary trees (and therefore the number of products of the result).
- Their running times compare often favourably with those of other methods.

The pivotal decomposition is also the basis of Bryant's Binary Decision Diagrams [Bry86, BRB90]. Binary Decision Diagrams have proved to be the most efficient tool to assess fault trees. A key issue for the efficiency of this technology is to find a good variable ordering. In this article, we propose a factorization algorithm (together with a variable ordering heuristic) to be used as a preprocessing of sums of products. This algorithm is derived from the heuristic developped for our pivotal decomposition procedure.

In order to compare all of these methods, we considered mainly three kinds of formulae.

- Sums of minimal cutsets of real life fault trees.
- Sums of st-paths of reliability networks of the literature.
- Randomly generated sum-of-products.

This benchmark covers, at least to a certain extent, the formulae one may encounter in practice.

The remainder of this article is organized as follows. Section 2 presents the problem and examines its complexity. Section 3 gives a brief overview of the classical methods proposed in the literature. Section 4 presents pivotal decomposition procedures and discusses two key issues to make them efficient: data structures and branching heuristics. Section 5 recalls basic about Binary Decision Diagrams and proposes a preprocessing method dedicated to the assessment of sums of products. Finally, section 6 provides some experimental results.

2 Sums of Products

2.1 Presentation

Let $S = \sum_{i=1,\ldots,n} \pi_i$ be a Boolean expression in disjunctive normal form (DNF), or a sum of products, i.e. where each product π_i is a conjunct of literals and each literal is either a variable v_j or its negation $\overline{v_j}$ $(1 \leq j \leq n)$. Throughout the paper we keep the same meaning for m and n, i.e. the number of products and the number of variables of the formula under study. v_j and $\overline{v_j}$ are said opposite and we denote by $\overline{p}$ the opposite of a literal p $(\overline{\overline{p}} = p)$.

In order to assess the probability $p(S)$ of such a formula S from the probabilities of the v_j's a first solution consists in applying the Sylvester-Poincare development:

$$p(S) = \sum_{1 \leq i \leq n} p(\pi_i) - \sum_{1 \leq i < j \leq n} p(\pi_i \pi_j) + \sum_{1 \leq i < j < k \leq n} p(\pi_i \pi_j \pi_k) - \ldots + (-1)^{m+1} p(\pi_1 \ldots \pi_m)$$

where $p(\pi)$ is product of the probabilities of its literals if π contains no two opposite literals and 0 otherwise. The full computation of the Sylvester-Poincare suffers from the exponential blow-up of the number of product probabilities it requires. This is the reason why most of the fault tree assessment tools restrict themselves to the very first terms of this development, applying the so-called Boole-Bonferroni bounds.

Let $W_k(S) = \sum_{1 \leq i_1 < i_2 < \ldots < i_k \leq n} p(\pi_{i_1} \pi_{i_2} \ldots \pi_{i_k})$, then the following inequalities hold.

$$\begin{aligned} W_1(S) - W_2(S) &\leq p(S) \leq W_1(S) \\ W_1(S) - W_2(S) + W_3(S) - W_4(S) &\leq p(S) \leq W_1(S) - W_2(S) + W_3(S) \\ &\vdots \end{aligned}$$

This technique gives often acurate results, especially when the formula under study is monotone (without negation) and the probabilities of its variables are low. Note that it is not true in general that the difference between two consecutive bounds decreases.

The other solution to assess the probability of S consists in rewriting S in order to make its products disjoint, i.e. that for any two products π_i and π_j there is at least a variable that appears positively in π_i and negatively in π_j, or vice-versa. If the products

are pairwisely disjoint, the probability of S is the sum of the probability of its products. It is therefore assessed linearly w.r.t. the size of S.

2.2 Complexity Issues

In 1979, Valiant introduced a new complexity class, the class #P, to describe problems that consist in counting the number of solutions of a decision problem [Val79b, Val79a]. The class #P contains all the counting problems whose decision counterpart is NP-complete. Preliminary results by Valiant himself, completed with a theorem by Toda [Tod89] showed that counting is surprisingly hard. Moreover, there are cases in which the decision problem is trivial while the corresponding problem is #P-hard. As an illustration, deciding whether a monotone Boolean formula (e.g. a sum of products without negation) is satisfiable is trivial (it suffices to set all the variables to 1), but it is hard to count its number of solutions.

Determining the probability of a formula is indeed at least as hard as counting its solutions for the latter problem is reduced to the former just by setting the probabilities of all variables to 2^{-n}.

These theoretical considerations have several practical consequences for the problem we are dealing with here. First, it is hopeless to find a polynomial time algorithm to compute the probability of a sum of products. Second, any attempt to rewrite a sum of products into a sum of disjoint products is subject to combinatorial explosion. Third, algorithms can therefore be compared only from a practical viewpoint.

3 Methods to Make Disjoint Products

3.1 Principle

A lot of algorithms has been developed to rewrite the sum of products in order to make disjoint products [Mis93, RVT95]. There are based on comparisons between products. These comparisons lead to multiply the compared products by well chosen products of variables in order to disjoint them to others. KDH [Hei89] is a well known algorithm to assess disjoint products, improved recently by the DA algorithm [Bou93, CDRB99]. The ordering of the comparison of the products has a significant effect on the final sum of disjoint products –number of products and their length [SR93, CDRB99]. In [CDRB99], a preprocessing module (PPM) was proposed to improve the ordering before the use of a disjonction algorithm –the whole method PPM and DA is called Semi-Optimal Disjonction Approach, SODA. This module may improve KDH and DA performances for many examples with a competitive processing time. Nevertheless PPM is not efficient in the general case –we show below some examples for which PPM decreases KDH and DA performances.

3.2 K.D. Heidtmann algorithm: KDH

The KDH88 algorithm is an improvement of the Abraham algorithm [Abr79] which is based on single variables complementation in order to disjoint products. The main idea is to compare at each stage $i = 1, ..., n$ a fixed product π_i to others π_j, $j < i$, considering the set $T_{i,j}$ of π_j uncompleted variables which don't exist in π_i.

In the Abraham algorithm, π_i is replaced by $\bar{x}_1.\pi_i + x_1.\bar{x}_2.\pi_i + ... + x_1.x_2...\bar{x}_k\pi_i$ at each comparison from $j = i + 1$ to n, with $T_{i,j} = \{x_1, ..., x_k\}$.

In the KDH algorithm, a multi-variable complementation rule is proposed: π_i is replaced by $\overline{x_1.x_2...x_k}\pi_i$. But, for each complemented set of variables Y in π_i, for which a sub-set Z contains variables of π_j, the complementation has to be rewritten before the replacement rule of π_i as follows: $\overline{\prod_{y_k \in Y} y_k} = \overline{\prod_{z_k \in Z} z_k} + \prod_{z_k \in Z} z_k \overline{\prod_{y_k \in Y \setminus Z} y_k}$

This last formula is needed to ensure the complementation using the multi-variable complementation rule proposed in the KDH algorithm. It increases the number of products only in particular cases –when $Z \neq \emptyset$– which lead to better performances than Abraham algorithm and some others as [LW92, LYL93].

3.3 Disjonction Approach: DA

The Disjonction Approach (DA) is based on the same principle as KDH algorithm. The improvements are based on the ordering of products. First, the products are ordered according to their length. The algorithm starts the comparison from one of the longest products to one of the shortest's for $i = n, ..1$. At each step, among products with a given length, the KDH comparison rule is applied choosing π_j such as its set of selected uncompleted variables (which don't exist in π_i) is smallest as possible. In practice, this new rule leads to order the π_j to be compared to a given π_i for each i.

This disjonction algorithm is more efficient than KDH one and others developed in 1990's [CDRB99]. But, the initial ordering of products has also a significative effect to the disjoint products obtained with DA. For this reason, a preprocessing module (PPM) has been proposed to improve DA results.

3.4 Semi-Optimal Disjonction Approach: SODA

DA efficiency has been improved in classical examples using PPM –to see [CDRB99] for details. The principle is to calculate r_i coefficients to characterize the ability of each product which has to be compared, to be disjoint from other products –or to reduce the number of additional products induced. The main idea is to compare each others the differences of the variable sets to be evaluated in the disjonction procedure. Because of the high number of the whole possible differences, the r_i coefficients calculated by PPM are only based on the two set differences.

Let $N = \text{card}[\bigcup_{j \neq k} T_{i,j} \cap T_{i,k}]$ and $D = \text{card}[\bigcup_j T_{i,j}]$. It can be verified that the disjunction approach produces relatively few terms when N/D is close to 0 or to 1, and

consequently the products with r_i's close to $1/2$ are placed at the end of the comparison sequence. Then, r_i coefficient was chosen equal to $|1/2 - N/D|$, the sequence being in decreasing order of r_i.

4 An Efficient Pivotal Decomposition Procedure

In this section, we propose an efficient implementation of the so-called pivotal decomposition method. This kind of algorithms have been widely studied by the Artificial Intelligence community in a slightly different context: the resolution of the well-known SAT problem (which is the cornerstone on complexity theory and has been the first problem to be shown NP-complete [Coo71]). The recent monography [KL99] gives a large overview of this topics.

4.1 Presentation of the Method

The pivotal decomposition method builds implicitly a binary tree. It may either produce an equivalent sum of disjoint products or compute the probability of the sum of products under study. For the sake of simplicity, we present below both variations at once. At each step, the algorithm has two parameters: a sum-of-products S and a product π. It starts with the sum-of-products under study and the empty product. It works as follows.

Satisfaction rule: If S contains an empty product, then π is a cutset of S. π is added to the sum of disjoint products under construction. If one computes a probability, the value 1 is returned.

Falsification rule: Else if S is empty, then π falsifies S. If one computes a probability, the value 0 is returned.

Unit product rule: Else if S contains a unit product, i.e. a product with only one literal x, then the algorithm is called on $S_{\bar{x}}$ and $\pi.\bar{x}$. The obtained products together with the product $\pi.x$ are added to the sum of disjoint products under construction. If one computes a probability, then $p(S) = p(x)+p(\bar{x}).p(S_{\bar{x}})$. This rule is actually borrowed from the Davis-Putnam procedure [DLL62].

Branching rule: Else, a literal x is selected, the algorithm is called first on S_x and $\pi.x$ and second on $S_{\bar{x}}$ and $\pi.\bar{x}$. The products obtained by both recursive calls are added to the sum of disjoint products under construction. If one computes a probability, then $p(S) = p(x).p(S_x) + p(\bar{x}).p(S_{\bar{x}})$.

As an illustration, let us consider the sum of products $S = ab + bcd + dei + acei$, taken from reference [CDRB99]. The binary tree traversed by the algorithm for S and the branching heuristic described in the next section is pictured Fig. 1. On this figure, internal nodes are labelled with the current sum of products as well as the rule applied to this formula. Leaves are labelled either with 1 or 0, depending on whether the satisfaction rule

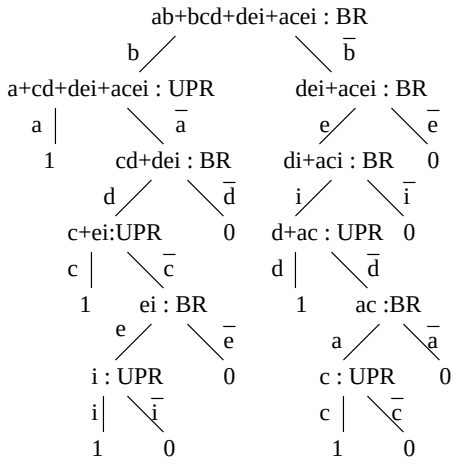


Figure 1: An example of binary tree built by the pivotal decomposition method.

or the falsification rule is applied. Branches are labelled with literals. The sum of disjoint products built by the algorithm is (from left to right): $ba + b\bar{a}dc + b\bar{a}d\bar{c}ei + \bar{b}eid + \bar{b}eiac$.

By construction, all of the produced products are pairwisely disjoint and their sum is equivalent to the formula under study. Alternatively, the algorithm computes the exact probability of the formula. Note that the binary tree can be a more concise representation for the sum of disjoint products than the sum of disjoint products itself (thanks to the sharing of prefixes of products).

The efficiency of the above algorithm relies on two points. First, the data structure used to encode sum-of-products, to compute S_x from S and x, and to detect unit products. Second, the branching heuristic. This latter point will be discussed in the next section.

The best data structure to encode sum-of-products consists in a sparse matrix. Each row of the matrix encodes a product. Each column encodes the occurrences of a variable. Such a data structure makes it possible to compute S_x and to detect unit product in linear amortized time (see, for instance, [Rau95] for a discussion). The main idea is that S_x is computed just by traversing the list of occurrences of x in S and by updating some counters (e.g. the numbers of satisfied and falsified literals in each product). When the algorithm backtracks, the same traversal is used to restore the counters. Therefore, the whole algorithm works in constant space (it needs only the recursivity stack).

The above scheme can be adapted to compute lower and upper approximations of the probability of the formula under study. The idea is to add a threshold on the probability of the product π (or alternatively on the number of positive/negative literals). If, at a given step of the algorithm, this probability goes below the threshold then the subtree rooted by the current node is not explored. The returned probability is either 0 or 1 depending on whether one wants a lower or an upper approximation.

4.2 Heuristics

Branching heuristics for pivotal decomposition methods have been widely discussed in the litterature, e.g. [HV95]. Conversely to BDDs for which heuristics are used to determine a variable order once for all, here the heuristic is invocated each time the branching rule is applied. The role of such heuristic is to reduce the size of the search tree. Two rules of thumb are applied to do so:

- One tries to produce as many unit products as possible. Each application of the unit product rule eliminates a variable for free.
- One tries to balance the sizes of the two subtrees. Assume for instance that we have two heuristics A and B such that A eliminates two variables on each branch (after its application and an additional application of the unit product rule), and B eliminates one variable on the first branch and three variables on the second one. Simple combinatorial arguments show that the tree built by A is much smaller than the tree built by B.

The design of a branching heuristic must result from a tradeoff between the accuracy of the result and the computational cost.

For the purpose of this study, we used the following heuristic. The selected variable is the most frequent variable in the shortest products. In order to break the ties, one considers the frequence in second shortest products (and then just takes the first variable in the instance order).

This heuristic is easy to apply and gives rather good results. Its design results from an extensive study over a benchmark made of dozens of formulae (some of which are presented section 6). We considered in turn several heuristics. From this study, we drew several conclusions.

- Static heuristics (i.e. those that select a variable order once for all) give very poor results.
- In order to be efficient, a heuristic must select the next variable in the shortest products. The idea is basically that once the algorithm has chosen to explore a part of the formula, it must stay on this part until it is fully explored.
- More elaborated selection mechanisms can be designed. However, the one we presented above is a good tradeoff. It is worth noticing that it is often the case that brute force exploration performs better than clever exploration schemes.

5 Binary Decision Diagrams

5.1 A Brief Presentation

Bryant's Binary Decision Diagrams (BDDs) [BRB90] are the state-of-the-art data structure to encode and to manipulate Boolean functions. The BDD associated with a formulae is a

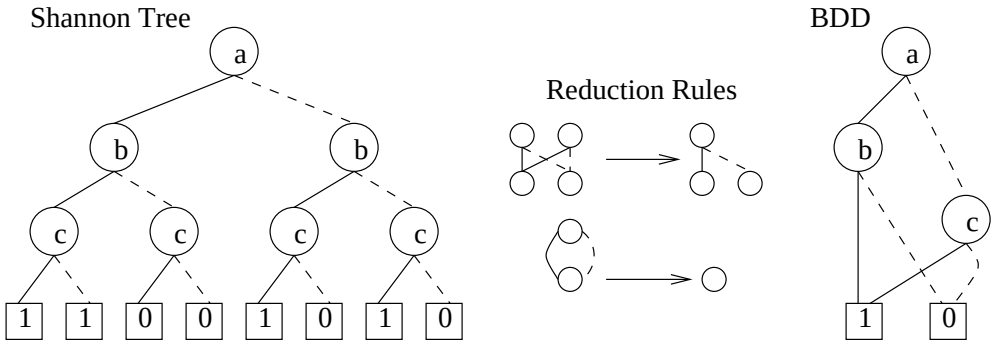


Figure 2: From the Shannon tree to the BDD encoding $ab + \bar{a}c$.

compact encoding of the truth table of this formula. This representation is also based on the pivotal decomposition: Let F be a Boolean formula that depends on the variable v, then $F = v.F_v + \bar{v}.F_{\bar{v}}$. By choosing a total order over the variables and applying recursively the pivotal decomposition, the truth table of any formula can be graphically represented as a binary tree. The nodes are labeled with variables and have two outedges (a *then*-outedge, pointing to the node that encodes F_v, and a *else*-outedge, pointing to the node that encodes $F_{\bar{v}}$). The leaves are labeled with either 0 or 1. The value of the formula for a given variable assignment is obtained by descending along the corresponding branch of the tree. The Shannon tree for the formula $ab + \bar{a}c$ and the lexicographic order is pictured Fig. 2 (dashed lines represent *else*-outedges).

Indeed such a representation is very space consuming. It is however possible to shrink it by means of the following two reduction rules.

- Isomorphic subtrees merging. Since two isomorphic subtrees encode the same formula, at least one is useless.
- Useless nodes deletion. A node with two equal sons is useless since it is equivalent to its son ($v.F + \bar{v}.F = F$).

By applying these two rules as far as possible, one get the BDD associated with the formula. A BDD is therefore a directed acyclic graph. It is unique, up to an isomorphism. This process is illustrated on Fig. 2.

Logical operations (and, or, xor, ...) can be directly performed on BDDs. This results from the orthogonality of usual connectives and the Shannon decomposition:

$$(v.F_1 + \bar{v}.F_0) \odot (v.G_1 + \bar{v}.G_0) \quad = \quad v.(F_1 \odot G_1) + \bar{v}.(F_0 \odot G_0)$$

where $\odot$ is any binary connective.

Among other consequences, this means that the complete binary tree is never built and then shrunk: the BDD encoding a formula is obtained by composing the BDDs encoding its subformulae. Moreover, a caching principle is used to store intermediate results of computations. This makes the usual logical operations (conjunction, disjunction) polynomial

in the sizes of their operands. A complete implementation of a BDD package is described in [BRB90]. The reader interested in more details should thus refer to this article.

5.2 Specific Heuristics to Handle Sums of Products

It is widely known, since the very first uses of BDDs, that the chosen variable ordering has a great impact on the size of BDDs, and therefore on the efficiency of the whole methodology. Finding the best ordering (or even a reasonnably good one) is a very hard problem (see [Weg00] for a recent survey on this topics). Two kinds of heuristics are used to determine which variable ordering to apply. Static heuristics are based on topological considerations and select the variable ordering once for all (see for instance [FFK88]). Dynamic heuristics change the variable ordering at some points of the computation. The latters are thus more versatile than the formers, but the price to pay is a serious increase of running times. Sifting is the most widely used dynamic heuristics [Rud93].

In order to handle sum-of-products, we designed a two steps preprocessing algorithm. First, the variables are ordered according to a heuristic presented below. Second, the sum of products is factorized as follows.

Let S be the sum of products under study and let v be the first variable in the order. Let S^v, $S^{\bar{v}}$ and $S^{\not v}$ be respectively the subsets of S of the products that respectively contain v, contain $\bar{v}$ and contain neither v nor $\bar{v}$. Then S is rewritten as $v.S^v_v + \bar{v}.S^{\bar{v}}_{\bar{v}} + S^{\not v}$ and the factorization is applied recursively on S^v_v, $S^{\bar{v}}_{\bar{v}}$ and $S^{\not v}$. This factorization is cheap, since at each step the current sum of products is splitted into three disjoint subsets. It simplifies the BDD computation because none of the formulae S^v_v, $S^{\bar{v}}_{\bar{v}}$, $S^{\not v}$ contains v.

The total order among the variables is determined as follows. The first variable v_1 is the most frequent one in the shortest products. In case of ties, one considers the frequence in the second shortest products. And so on. Once v_1 is selected, one considers the formula $S_{\not v_1}$ obtained from S by removing the occurrences of v_1 (and $\bar{v_1}$) and the empty products, if any. The second variable is selected by applying the same method on $S_{\not v_1}$, and so on.

The factorization does not influence the size of the final BDD. It has however a dramatic influence on the sizes of intermediate computations. This has been already pointed out by one the authors in [NR98]. The design of the variable ordering heuristic follows from the conclusions drawn for the design of branching heuristics for the pivotal decomposition method (see section 4). The only difference is that here the heuristic determines the variable order once for all. Therefore, it is interesting to pay a bit more to be more clever.

6 Experimental Results

In order to compare the algorithms presented in the previous sections, we considered mainly three kinds of formulae: sums of minimal cutsets of real fault trees, sums of products that encode s,t-paths of networks and sums of products generated at pseudo-random according to the fixed length model. Finding a good benchmark is always a difficult task. The first two categories of formulae cover, at least to a certain extent, the kind of formulae one may

			Poincaré		Algorithms					
Fault Trees	#var.	#MCS	1st	2nd	KDH	DA	SODA	ESOP	Fact.	BDDs
baobab1	61	46188	0.14	3770	_	_	_	_	21.96	0.63
baobab2	32	4805	0.01	25.42	8192	2092	3303	6.59	0.81	0.00
baobab3	80	24386	0.07	978	_	_	_	_	10.88	0.27
chinese	25	392	0.00	0.09	12.45	2.98	4.16	0.02	0.05	0.00
das9201	122	14217	0.02	143	_	_	_	1537	4.88	0.02
das9202	49	27778	0.10	1398	_	_	_	1055	12.05	0.01
das9205	51	17280	0.04	330	188740	31275	221255	3.00	4.18	0.00
das9208	103	8060	0.02	53	_	_	_	15.89	2.33	0.14
edf9205	165	21308	0.04	411	_	_	_	2135	11.08	0.02
edfpa15r	88	26549	0.08	1079	_	_	_	9979	13.41	0.38
ftr10	152	305	0.00	0.02	_	_	_	0.49	0.10	0.00
isp9603	91	3434	0.01	8.57	_	_	_	661	0.87	0.04
isp9605	32	5630	0.02	38.10	26832	3634	7376	12.50	1.24	0.01
isp9606	89	1776	0.01	1.47	14464	3635	7969	105	0.35	0.00
jbd9601	532	14007	0.02	139	_	_	_	_	22.27	4.00

Table 1: Running times to assess minimal cutsets of fault trees

expect to encounter in practive. Randomly generated formulae have also some interesting properties that will be discussed below.

6.1 Minimal Cutsets of Real-Life Fault Trees

The first set of formulae we considered consists of the disjuncts of the minimal cutsets of 15 real-life fault trees. The number of basic events and minimal cutsets for each of these trees is given table 1.

The fourth and fifth columns give the running times in seconds to compute the first two terms of the Sylvester-Poincaré development (on a PC pentium III biprocessor 733MHz running Linux). The remaining columns give the running times of respectively, the algorithms KDH, DA, SODA, ESOP (the pivotal decomposition method described section 4), the factorization algorithm and the computation of the BDD from the factorized formula (a symbol "_" in a cell indicates that the algorithm is unable to handle the formula within a reasonable running time).

The table 2 gives the sizes of the BDDs as well as the running times (still in seconds) to compute the probability of the formula from its BDD. It is worth to mention that the computed probability is the exact one, for — thanks to the Shannon decomposition — no approximation is required.

The following facts can be observed from results presented tables 1 and 2.

Fact 1. Running times for BDDs are by orders of magnitude smaller than those of other methods. Moreover there are examples (e.g. jbd9601) for which BDDs are the only

	baobab1	baobab2	baobab3	chinese	das9201	das9202	das9205	das9208
ESOP tree size	–	66486	–	2341	8271107	350890	17973	311538
BDD size	18786	204	10169	67	696	353	272	11490
CPU Pr	0.07	0.00	0.02	0.00	0.01	0.00	0.01	0.03

	edf9205	edfpa15r	ftr10	isp9603	isp9605	isp9606	jbd9601
ESOP tree size	3417097	8232340	238995	32388182	104586	8083369	–
BDD size	935	11310	302	4331	401	346	342190
CPU Pr	0.01	0.03	0.00	0.00	0.00	0.00	0.58

Table 2: More statistics about minimal cutsets of fault trees

successful method.

Fact 2. Not only running times of BDDs are better, but also BDDs are much smaller than the sums of disjoint products produced by KDH, DA or SODA and the binary trees produced by ESOP. As a consequence, the computation of the probability of the formula is much more efficient using BDDs than any of the other methods.

Fact 3. In the Table 1, the DA computing time is smaller than SODA and KDH ones. One can notice that the obtained number of disjoint products –and total number of operations to calculate the final probability– in increasing order is smaller with DA, KDH and SODA, except for "isp9605" example –the increasing order is then SODA, DA and KDH.

Fact 4. The ESOP computing time is largely smaller than SODA, DA and KDH ones. The total number of operations to calculate the final probability is also largely smaller with ESOP than with SODA, DA or KDH, even if it is not the case for the number of disjoint products.

Fact 5. For these "not too small" formulae the computation of Boole-Bonferonni bounds is very expensive (the third term is not computable for most of these formulae within reasonable time limits). Moreover, it is always faster to compute the BDD and then the probability than to compute the second term of the Sylvester-Poincaré development.

6.2 Reliability Networks

Sums of products arise naturally in the study of reliability networks. In order to determine the probability that there is an operating s,t-path, it is often convenient to determine

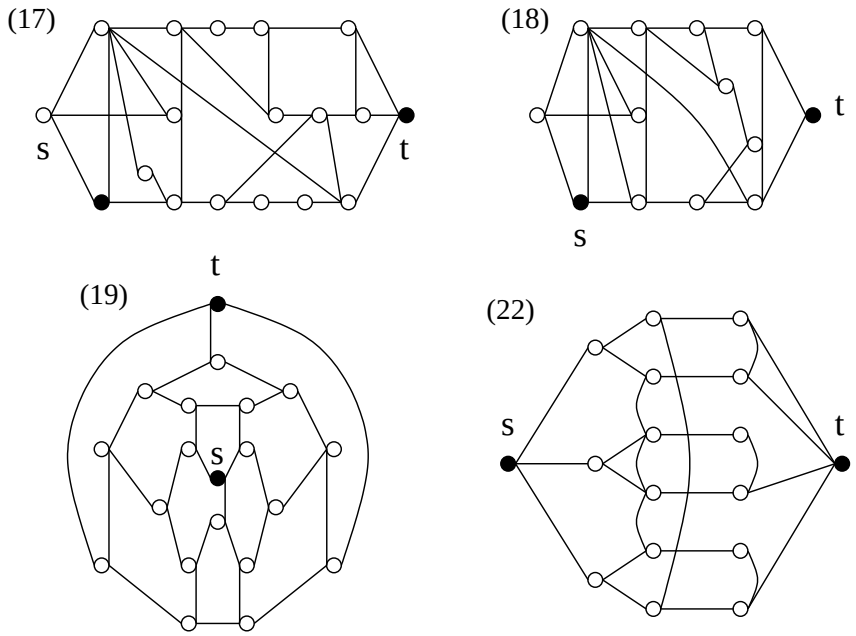


Figure 3: Four networks taken from reference [KLY99]

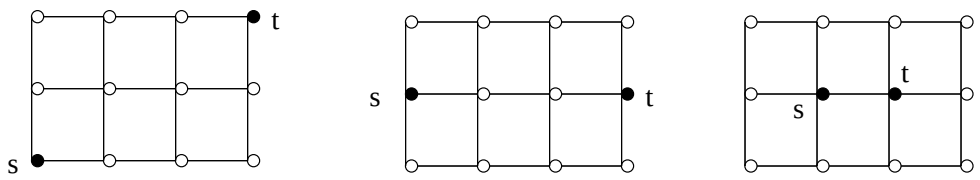
minimal s,t-paths, to encode them as a sum of products and to assess the probability this formula. More direct compilation schemes exist [MCFB94, KLY99, Rau01]. However, the above methodology is still of interest for it is simple and robust (and does not require the use of quantifiers, conversely to the cited methods).

In references [PP88, SR93, KLY99], a collection of more or less realistic networks is given as a benchmark for reliability network assessment methods. Unfortunately, most of these networks are too small to make any valuable comparison. We report here only results for four large ones that are pictured Fig. 3.

A good benchmark should contain some scalable test cases. Fig. 4 presents three networks that differ only from the location of their source and target nodes. Source and target nodes of (n, m) grid, rescom and taxi networks are respectively located on nodes $(1, 1)$ and (n, m), $(1, \lfloor m/2 \rfloor)$ and $(n, \lceil m/2 \rceil)$, and $(\lfloor n/2 \rfloor, \lfloor m/2 \rfloor)$ and and $(\lceil n/2 \rceil, \lceil m/2 \rceil)$. These three families of networks have hovewer a drawback: they are in some sense too regular and give a great advantage to methods that capture these regularities, such as BDDs.

The tables 3 gives the running times of the different algorithms considered so far for the four networks issued from [KLY99] and some networks of the Grid, Rescom and Taxi families.

It is clear from tables 3 and 4 that the facts we observed for minimal cutsets of fault trees remain valid for s,t-paths of reliability networks.



(a) Grid (4,3) (b) Rescom (4,3) (c) Taxi (4,3)

Figure 4: Grid networks.

			Poincaré		Algorithms					
Networks	#var.	#paths	1st	2nd	KDH	DA	SODA	ESOP	Fact.	BDDs
KLY17	25	145	0.00	0.02	0.96	0.58	0.25	0.09	0.02	0.01
KLY18	22	269	0.00	0.07	5.46	2.25	1.10	0.06	0.03	0.01
KLY19	30	780	0.00	1.18	629	282.7	33.7	14	0.18	0.02
KLY22	26	192	0.01	0.03	6.01	3.74	0.89	0.29	0.03	0.00
Grid (2,11)	31	1024	0.01	2.57	14.57	6	11.43	2.08	0.70	0.11
Grid (2,12)	34	2048	0.01	12	117	36.86	83	10.92	1.67	0.28
Grid (2,13)	37	4096	0.02	53	875	226	564	60	4.01	0.73
Rescom (4,5)	31	395	0.00	0.21	5.71	3.43	1.71	1.20	0.13	0.01
Rescom (5,4)	31	613	0.00	0.62	13.43	8	5.14	3.06	0.29	0.09
Rescom (5,5)	40	3915	0.02	47	4806	2366	918	675	2.21	0.10
Taxi (3,8)	37	801	0.00	1.61	48.57	25.14	10	1.55	0.41	0.00
Taxi (3,9)	42	1217	0.00	0.09	211	85.71	27.43	11.67	0.66	0.01
Taxi (3,10)	47	4803	0.00	0.09	11790	3527	963	155	3.62	0.01
Taxi (3,11)	52	6999	0.00	0.09	_	_	3106	2042	5.35	0.01
Taxi (4,5)	31	339	0.00	0.15	4.29	2.86	1.14	0.52	0.14	0.01
Taxi (4,6)	38	1756	0.01	8.18	715	275	82.57	44.62	0.87	0.01
Taxi (4,7)	45	6677	0.04	156	_	_	3219	2283	4.29	0.02

Table 3: Running times to assess disjuncts of s,t-paths of networks

	KLY17	KLY18	KLY19	KLY22	Grid (2,11)	Grid (2,12)	Grid (2,13)	Rescom (4,5)	Rescom (5,4)
ESOP tree size	13395	5780	348874	31149	37861	98840	258019	70892	120608
BDD size	1070	333	1546	242	7704	15810	31845	934	7992
CPU Pr	0.00	0.00	0.00	0.00	0.03	0.05	0.10	0.00	0.01

	Rescom (5,5)	Taxi (3,8)	Taxi (3,9)	Taxi (3,10)	Taxi (3,11)	Taxi (4,5)	Taxi (4,6)	Taxi (4,7)
ESOP tree size	3585841	39130	203465	559062	2896349	34423	532884	5417036
BDD size	7082	1296	300	1936	410	4239	1698	2061
CPU Pr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01

Table 4: More statistics about s,t-paths of reliability networks

6.3 Randomly Generated Formulae

Formulae generated at pseudo-random have two interesting properties. First, they provide an inexhaustible source of scalable formulae. Second, the generated formulae have almost no structure and therefore no regularities. The more there are regularities the easier the treatments are (or at least could be), and vice versa. In other words, pseudo-random generation is a source of concise and hard formulae. This latter point has been demonstrated for the SAT problem both theoretically [CS88], and experimentally [MSL92]. It should be noticed however that formulae encountered in practice do have regularities for they derive from human-made artefacts. Results of experiments on pseudo-randomly generated formulae must therefore be considered with care.

For the purpose of the present study, we generated monotone sums of products according to the fixed-length model. The principle is as follows. One chooses a number of variables n, a number of products m and a size of products k. Then, each product is generated independently by drawing k variables out of the n's (each variable has the same probability to be drawn). It is easy to verify that if k and m are small w.r.t. respectively n and $\binom{k}{n}$ then the probability to draw the same product twice is neglictible. Each instance is characterized by four numbers: n, m, k that are defined as above and s the seed of the random number generator.

Tables 5 and 6, which report performances of the algorithms considered so far, shows a totally different picture that what was observed for minimal cutsets and s,t-paths.

Fact 6. BDDs still outperform KDH, DA and SODA. However, ESOP is now clearly the best algorithm. Moreover, the larger the formula, the greater the advantage of ESOP over BDDs.

Fact 7. Since m and k are small, the computation of the first two terms of the Sylvester-

			Poincaré		Algorithms					
Networks	#var.	#prd.	1st	2nd	KDH	DA	SODA	ESOP	Fact.	BDDs
FL20-50-5-1111	20	1000	0.00	0.60	49.14	15.71	27.43	0.69	0.10	0.13
FL20-50-5-1234	20	1000	0.00	0.59	48.86	16.29	27.71	0.70	0.10	0.13
FL25-40-4-1111	25	1000	0.00	0.47	543.4	93.43	105.1	0.99	0.08	0.36
FL25-40-4-1234	25	1000	0.00	0.46	88.86	32.57	105.4	1.00	0.09	0.35
FL25-40-5-1111	25	1000	0.00	0.61	287.4	114.9	268	5.61	0.12	2.75
FL25-40-5-1234	25	1000	0.00	0.65	1045	241.7	256.9	5.61	0.12	2.62
FL30-40-3-1111	30	1200	0.00	0.47	76.86	20	140.6	0.23	0.08	0.09
FL30-40-4-1111	30	1200	0.00	0.72	579.7	208.6	_	6.55	0.12	7.40
FL30-40-4-1234	30	1200	0.00	0.67	611.7	216.9	_	6.41	0.13	8.08
FL30-40-5-1111	30	1200	0.00	0.98	_	_	_	55.60	0.17	182
FL30-40-5-1234	30	1200	0.00	0.94	_	_	_	54.70	0.17	167
FL35-40-3-1111	35	1400	0.00	0.68	458.6	106.9	_	0.78	0.10	0.47
FL35-40-4-1111	35	1400	0.00	1.03	_	_	_	42.36	0.16	280
FL35-40-4-1234	35	1400	0.00	0.98	_	_	_	41.94	0.17	243
FL40-30-3-1111	40	1200	0.00	0.50	2140	595.1	_	3.49	0.11	6.13
FL45-20-3-1111	45	900	0.00	0.27	_	_	_	20.88	0.09	285

Table 5: Running times to assess pseudo-randomly generated formulae

	FL20-50-5-111145	FL20-50-5-1234	FL25-40-4-1111	FL25-40-4-1234	FL25-40-5-1111	FL25-40-5-1234	FL30-40-3-1111	FL30-40-4-1111
ESOP tree size	26834	27712	56509	55892	253187	252105	15804	344540
BDD size	12151	11890	32147	30578	123717	124482	11442	209245
CPU Pr	0.02	0.02	0.05	0.05	0.21	0.22	0.02	0.36

	FL30-40-4-1234	FL30-40-5-1111	FL30-40-5-1234	FL35-40-3-1111	FL35-40-4-1111	FL35-40-4-1234	FL40-30-3-1111	FL45-20-3-1111
ESOP tree size	336282	2346268	2303536	51531	2076400	2053771	271878	2101086
BDD size	213167	1241981	1185211	42286	1417501	1449130	257671	2179400
CPU Pr	0.36	2.38	2.26	0.06	2.55	2.64	0.40	3.67

Table 6: More statistics about pseudo-randomly generated formulae

Poincaré development is much faster than any of the considered algorithms (subsequent terms are however still hard to compute).

Fact 8. DA performs better than KDH which in turn performs better than SODA.

Fact 9. Due to the lack of regularities, the sizes of ESOP trees are only about twice the sizes of the corresponding BDDs for most of the considered instances. For the two last instances (that are the largest ones) the sizes of ESOP trees and BDDs are even approximately the same.

7 Conclusion

In this article, we reported experimental results we obtained with different algorithms on a large benchmark of sums-of-products. We considered five algorithms:

- KDH, DA and SODA that rewrite the sum of products under study into an equivalent sum of disjoint products. These algorithms can be considered as among the most efficient in their category.
- ESOP which is an efficient implementation of the pivotal decomposition method.
- Binary Decision Diagrams, for which we propose a specialized preprocessing technique.

From this large experimental study, we drew the following conclusions.

- KDH, DA and SODA are never competitive with ESOP and BDDs.
- BDDs outperform ESOP when the formula under study has regularities. Most of the "real-life" formulae (minimal cutsets of fault trees, s,t-paths of reliability networks) do have such regularities.
- ESOP performs better than BDDs when the formula under study has no structure (this is the case for formulae generated at random).
- ESOP computes the probability of the formula in constant space. Moreover, it can be modified to give only lower and upper approximations of this probability. This makes ESOP of interest to deal with formulae for which the computation of the BDD is not tractable.

References

[Abr79] J. A. Abraham. An improved Algorithm for Network Reliability. *IEEE Transactions on Reliability*, R-28(1):58–61, April 1979.

[Bou93] T. Bouhoufani. *Contribution à la construction et au traitement des arbres de défaillance*. Thèse de doctorat, Université Bordeaux I, 1993.

[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.

[CDRB99] E. Châtelet, Y. Dutuit, A. Rauzy, and T. Bouhoufani. An optimized procedure to generate sums of disjoint products. *Reliability Engineering and System Safety*, 65:289–294, 1999.

[Coo71] S.A. Cook. The Complexity of Theorem Proving Procedures. In *Proceedings of the 3rd Ann. Symp. on Theory of Computing, ACM*, pages 151–158, 1971.

[CS88] V. Chvátal and E. Szemerédi. Many Hard Examples for Resolution. *JACM*, 35(4):759–768, october 1988.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A Machine Program for Theorem Proving. *CACM*, 5:394–397, 1962.

[FFK88] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and Improvements of Boolean Comparison Method Based on Binary Decision Diagrams. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'88*, pages 2–5, 1988.

[Hei89] K.D. Heidtmann. Smaller Sums of Disjoint Products by Subproduct Inversion. *IEEE Transactions on Reliability*, 38:305–311, 1989.

[HV95] J.N. Hooker and V. Vinay. Branching Rules for Satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.

[KL99] H. KleineBuning and L. Lettman. *Proposition Logic: Deduction and Algorithms*. Cambridge University Press, 1999. ISBN 0521630177.

[KLY99] Sy-Yen Kuo, Shyue-Kung Lu, and Fu-Min Yeh. Determining Terminal-Pair Reliability Based on Eedge Expansion Diagrams Using OBDD. *IEEE Transactions on Reliability*, 48(3):234–246, September 1999.

[LW92] M. O. Locks and J. M. Wilson. Note on Disjoint Products Algorithms. *IEEE Transactions on Reliability*, 41(1):81–84, March 1992.

[LYL93] H. H. Liu, W. T. Yang, and C. C. Liu. An improved minimizing algorithm for the summation of disjoint products by Shannon's expansion. *Microelectronic Reliability*, 33(4):599–613, 1993.

[MCFB94] J.-C. Madre, O. Coudert, H. Fraïssé, and M. Bouissou. Application of a New Logically Complete ATMS to Digraph and Network-Connectivity Analysis. In *Proceedings of the Annual Reliability and Maintainability Symposium, ARMS'94*, pages 118–123, 1994. Annaheim, California.

[Mis93] K.R. Misra. *New Trends in System Reliability Evaluation*. Fundamental Studies in Engineering, 16. Elsevier, 1993. ISBN 0-444-816607.

[MSL92] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proceedings Tenth National Conference on Artificial Intelligence (AAAI'92)*, 1992.

[NR98] M. Nikolskaia and A. Rauzy. Heuristics for bdd handling of sum-of-products formulae. In Lydersen, Hansen, and Sandtorv, editors, *Proceedings of European Safety and Reliability Association Conference, ESREL'98*, pages 1459–1465. Balkerna, Rotterdam, 1998. ISBN 90 54 10 966 1.

[PP88] L.B. Page and J.E. Perry. A Practical Implementation of the Factoring Theorem for Newtork Reliability. *IEEE Transactions on Reliability*, 37:259–267, 1988.

[PP89] L.B. Page and J.E. Perry. Reliability of directed newtork using the factoring theorem. *IEEE Transactions on Reliability*, 38:556–562, 1989.

[Rau95] A. Rauzy. Polynomial restrictions of SAT: What can be done with an efficient implementation of the Davis and Putnam's procedure. In U. Montanari and F. Rossi, editors, *Proceedings of the International Conference on Principle of Constraint Programming, CP'95*, volume 976 of *LNCS*, pages 515–532. Springer Verlag, 1995.

[Rau01] A. Rauzy. A new methodology to handle boolean models with loops. *IEEE Transactions on Reliability*, 2001. To appear.

[Rud93] R. Rudell. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In *Proceedings of IEEE International Conference on Computer Aided Design, ICCAD'93*, pages 42–47, November 1993.

[RVT95] S. Rai, M. Veeraraghavan, and K.S. Trivedi. A Survey of Efficient Reliability Computation Using Disjoint Products Approach. *Networks*, 25:147–163, 1995.

[Shi91] D.R. Shier. *Network Reliability and Algebraic Structures*. Oxford Science Publications, 1991.

[SR93] S. Soh and S. Rai. Experimental results on preprocessing of path/cut terms in sum of disjoint products technique. *IEEE Transactions on Reliability*, 42(1):24–33, 1993.

[Tod89] S. Toda. On the computational power of PP and $\oplus$P. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science*, pages 514–519, 1989.

[Val79a] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8:410–421, 1979.

[Val79b] L.G. Valiant. On the complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

[Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams - Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000. ISBN 0-89871-458-3.