An Optimized Procedure To Generate Sums of Disjoint Products

E. Châtelet [a], **Y. Dutuit** [b], **A. Rauzy** [c] & **T. Bouhoufani** [d]

[a] *Université de Technologie de Troyes/LM2S. 10010, Troyes cedex, France*

[b] *Université Bordeaux I/LAP-ADS. 33405, Talence cedex, France*

[c] *Université Bordeaux I/LaBRI. 33405, Talence cedex, France*

[d] *Université de Batna/Institut d'Hygiène et Sécurité. 05000, Batna Algérie*

This paper describes an efficient procedure, so-called SODA, to generate a sum of disjoint products (SDP) from a set of minimal products representing either the minimal s-t paths of reliability networks or the minimal cutsets of coherent fault trees. The proposed procedure involves the concept of multiple-variables inversion or grouped-variables inversion. The process is improved by carrying out a preprocessing of the initial formula which reduces the number of terms of the result, and hence the overall computation time. A previous version of this paper was submitted to RESS by the fall of 1993 and was accepted up to minor revisions one year later. Unfortunately, the final redaction has been delayed since. The present version takes into account some other papers published in between and includes new ideas that improve the implementation of the proposed algorithm. At least on classical test examples, SODA produces shorter formulae than the other SDP methods known by the authors.

1 INTRODUCTION

Many methods have been proposed in order to make disjoint the elements of a set of products representing either the minimal s-t paths of a reliability network or the minimal cutsets of a coherent fault tree (see [1] or [2] for a survey on Sum of Disjoint Products methods). The aim of such a rewriting is to make it possible to assess the probability of the formula in an efficient way. It is worth noticing that the determination of the probability of a Boolean formula is a very hard problem — it is #P-complete — even under very strong restrictions (such as monotony) on the type of the formula [3]. Without entering into details of computational complexity theory, this means that it is hopeless to find an efficient algorithm (i.e. an algorithm of polynomial worst case complexity) to solve it. As a consequence, it is hopeless as well to find an efficient algorithm to rewrite a set of products into an equivalent set of disjoint products, for the probability of the later can be determined linearly with respect to the number of variable occurrences it contains. Also hopeless is the aim to compare algorithms and heuristics from a general and theoretical viewpoint.

This does not mean however that nothing can be done in practice. SDP methods should be compared (experimentally) with respect to the following criteria:

(i) The form of the resulting formulae, which is mainly a matter of data structures.
(ii) The size of the results.
(iii) The size of intermediate data structures that are necessary to build the results.
(iv) The computation time.

In this article, we stand in the classical framework of SDP techniques where it is assumed, quite arbitrarily[1] , that the result has to be an explicitely given list of products. Several authors consider however products in which several variables can be negated at once. Algorithms using that technique (see for instance [5–7]) give in general shorter results than algorithms that produce only negative literals (see for instance [8–10]). For all of these algorithms, a preprocessing of the initial set is able to reduce the final number of disjoint products, and consequently the overall running time.

In this paper, we present a new procedure, so-called Disjunction Approach (DA), that uses the multiple variables negation technique. We propose also a preprocessing module, so-called PPM, which is a heuristics to order the products of the initial set. The whole method PPM+DA, is called Semi-Optimal Disjunction Approach (SODA).

The rest of this paper is organized as follows. The principle of the DA method is presented at section 2 by means of a small example. The section 3 describes the preprocessing module PPM by means of another small example. A comparison with already proposed heuristics is provided. The efficiency of SODA is shown in section 4 by comparing its results with those obtained by means of some other methods on a reference test example. Implementation issues are discussed at section 5. Finally, a short conclusion ends the paper.

2 THE DISJUNCTION APPROACH

2.1 Principle

The principle of the disjunction approach (DA) is as follows.

First, the products of the formula F under study are sorted according to any heuristics. Let us denote $P_1, P_2, \dots, P_n$ the obtained sequence of products. The result R is initialized with P_1.

Second, the products are considered in turn, in increasing order (of their indices). For each product P_k ($k = 2, n$), the following sum is built:

[1] This remark is motivated by the fact that there exists methods that do not produce this type of results but that make it possible to assess efficiently the probability of the formula. We think here especially to Binary Decision Diagrams (see for instance [4] for a survey on the use of BDDs in the reliability analysis framework).

$$T_k = \sum_{i=1}^{k-1} P_i \setminus P_k \tag{1}$$

where $P_i \setminus P_k$ denotes the constant 0 if P_i and P_k contains two opposite literals, and the product of the literals of P_i that do not belong to P_k otherwise. For instance, if $P_1 = ab$ and $P_2 = bcd$ then $P_1 \setminus P_2 = a$.

Third, the de Morgan laws are applied on the negation of T_k in order to get an equivalent sum of disjoint products.

$$\overline{T_k} = D_{k,1} + \dots + D_{k,r_k} \tag{2}$$

Fourth, the products $D_{k,1}.P_k$, ... , $D_{k,r_k}.P_k$ are added to the result R. By construction, the $D_{k,j}$'s are pairwisely disjoint and disjoint from the P_i's ($i < k$). Moreover, it is easy to verify that the result R is equivalent to the initial formula F.

Two improvements can be achieved on that process. First, if the term $P_i \setminus P_k$ implies another term $P_j \setminus P_k$, then it can be safely removed from the sum T_k.

Second, if the term $T_{k,i} = P_i \setminus P_k$ does not share any variable with the other terms of the sum T_k, then $\overline{T_k}$ can be rewritten as $\overline{T_k} = \overline{\sum_{j<k, j \neq i} T_{k,j}}.\overline{T_{k,i}}$. This is the principle multiple variables inversion method.

The disjonction approach embeds these two improvements.

2.2 Example

In order to illustrate how the DA method works, let us consider the four products given in tables 1 and 2 of reference [11]. Let f be the sum of these products: $f = ab + bcd + dei + acei$. The problem is to transform f into an equivalent sum of disjoint products with the minimal number of terms.

As presented by one of us in [12], let us construct a table T in which the initial products are arranged in marginal column (P_i) and row (P_j) in increasing order of their lengths (see table 1). The cell $T_{j,i}$ contains the product $P_i \setminus P_j$ if $i < j$, and the constant 0 otherwise.

An example of the first improvement is provided by the cells $T_{4,2}$ and $T_{4,1}$: $T_{4,2} = bcd \setminus acei = bd$ can be removed because it implies $T_{4,1} = ab \setminus acei = b$.

The fifth and sixth columns give respectively the T_i's and the $\overline{T_i}$'s. The last column gives the five

Table 1. The Disjunction Approach applied to the first sequence.

$P_i \backslash P_j$	ab	bcd	dei	$acei$	T_i	$\overline{T_i}$	$DP_i = \overline{T_i}.P_i$
ab	0	0	0	0	0	1	ab
bcd	a	0	0	0	a	$\bar{a}$	$\bar{a}.bcd$
dei	ab	bc	0	0	$ab + bc$	$\bar{b} + \bar{a}b\bar{c}$	$\bar{b}.dei + \bar{a}b\bar{c}dei$
$acei$	b	0	d	0	$b + d$	$\bar{b}\bar{d}$	$a\bar{b}c\bar{d}ei$

resulting disjoint products DP_i and we have $f = \sum_i DP_i$.

The DA method is very simple and slightly similar to the one presented in [11]. However, it is well known that the efficiency of this kind of procedures, in terms of resulting disjoint products, depends strongly on the order in which the products of the initial set are arranged. In order to illustrate this dependency, let us consider the same example as above, but for the order $f = ab + dei + bcd + acei$ (table 2).

With this second order, DA provides a four terms result instead of a five terms one in the former case. It is worth noticing that this improvement is possible because of the use of the multiple variables inversion principle (here applied to DP_2 with ab and to DP_3 with ei). The question is thus to design heuristics that arrange the initial sequence in order to obtain as few disjoint products as possible in the result. This is the role of the preprocessing module PPM.

3 THE PREPROCESSING MODULE

3.1 Principle

A brief examination of the table 1 shows clearly that the term DP_3 involves more products than its corresponding product P_3. This fact is obviously due to the disjunction process which generates additional terms related to the presence of some literals common to several $T_{i,j}$'s. The role devoted to the preprocessing module (PPM) of SODA is to reduce, as far as possible, the number of these additional terms. This is done thanks to discriminative coefficients r_i computed for each product P_i. These coefficients make it possible to sort the initial sequence in a semi-optimal manner (the order is only semi-optimal because the obtention of the optimal sequence cannot be guaranted).

The principle of PPM is as follows.

First, products are grouped according to their lengths. The groups are sorted in increasing order of the length of their products. Therefore, it remains to sort the products inside each group.

Second, for each product P_i, the products $T_{i,j} = P_j \setminus P_i$ are built for all of the products P_j's such that the length of P_j is less or equal to the length of P_i (i.e. each P_j that is potentially before P_i in the final order). The $T_{i,j}$'s are collected into a set T_i. The $T_{i,j}$'s that verify the preconditions of one of the two improvements described in the previous section are not collected. Namely, if $T_{i,j}$ implies another $T_{i,k}$ or if it does not share any variable with the other $T_{i,k}$'s, then it is not collected into T_i.

Third, the P_i's such that $T_i = \emptyset$ are put at the end of their group.

Fourth, the products that remains inside each group are sorted in increasing order of the coefficient r_i defined as follows (the considered $T_{i,j}$'s are those that belong to T_i).

$$r_i = \left| \frac{1}{2} - \frac{\text{card}[\bigcup_{j \neq k} T_{i,j} \cap T_{i,k}]}{\text{card}[\bigcup_j T_{i,j}]} \right| \tag{3}$$

Let $N = \text{card}[\bigcup_{j \neq k} T_{i,j} \cap T_{i,k}]$ and $D = \text{card}[\bigcup_j T_{i,j}]$. The coefficient r_i comes the experimental observation that the disjunction approach produces relatively few terms when N/D is close to 0 or to 1 (and consequently it is of interest to put the P_i's with r_i's close to 1/2 at the end of the sequence).

3.2 Example

As an illustration, let us consider the network pictured Fig. 1, so-called modified ARPANET, that has been already used as a reference case by several authors [2,6,13]. The table 3 summarizes the

Table 2. The Disjunction Approach applied to the second sequence.

$P_i \backslash P_j$	ab	dei	bcd	$acei$	T_i	$\overline{T_i}$	$DP_i = \overline{T_i}.P_i$
ab	0	0	0	0	0	1	ab
dei	ab	0	0	0	ab	$\overline{ab}$	$(\overline{ab})ei$
bcd	a	ei	0	0	$a + ei$	$\bar{a}(\overline{ei})$	$\bar{a}(\overline{ei})bcd$
$acei$	b	d	0	0	$b + d$	$\bar{b}\bar{d}$	$a\bar{b}c\bar{d}ei$

Table 3. Complete analysis of the ARPA net with SODA

P_i	initial rank	final rank	terms from which T_i and DP_i are built	r_i	DP_i
adh	1	1	$\mid bfi, be$	0.25	adh
bfi	3	2	$adh \mid eh$	0.25	$bfi(\overline{adh})$
beh	2	3	$ad, fi \mid$		$beh(\overline{ad})(\overline{fi})$
$acfi$	5	4	$dh, b,$ ~~beh~~ $\mid dg,$ ~~bdh~~, ~~bgh~~, $eh,$ ~~beg~~	0	$acfi\bar{b}(\overline{dh})$
$adgi$	6	5	$h, bf,$ ~~beh~~, $cf \mid$ ~~beh~~, ~~bfh~~, ~~ceh~~, be	0	$adgi\bar{h}\bar{f} + adgi\bar{h}f\bar{b}\bar{c}$
$bcdh$	7	6	$a, fi, e,$ ~~afi~~, ~~agi~~ $\mid fg,$ ~~ae~~, ~~egi~~	0.167	$bcdh\bar{a}\bar{e}(\overline{fi})$
$bfgh$	9	7	$ad, i, e,$ ~~aci~~, ~~adi~~, $cd \mid$ ~~ace~~, ~~ei~~	0.167	$bfgh\bar{e}\bar{i}\bar{d} + bfgh\bar{e}\bar{i}d\bar{a}\bar{c}$
$aceh$	4	8	$d,$ ~~bfi~~, $b, fi,$ ~~dgi~~, ~~bd~~, ~~bfg~~ $\mid$ ~~bgi~~		$aceh\bar{b}\bar{d}(\overline{fi})$
$begi$	8	9	~~adh~~, $f, h,$ ~~acf~~, $ad,$ ~~cdh~~, ~~fh~~, ~~ach~~ $\mid$		$begi\bar{f}\bar{h}(\overline{ad})$
$acegi$	10	10	~~dh~~, ~~bf~~, ~~bh~~, $f, d,$ ~~bdh~~, ~~bfh~~, $h, b \mid$ ~~fh~~, ~~df~~, ~~bd~~		$adegi\bar{b}\bar{d}\bar{f}\bar{h}$
$acfgh$	11	11	$d,$ ~~bi~~, ~~be~~, $i,$ ~~di~~, ~~bd~~, $b, e,$ ~~bei~~, ~~ei~~ $\mid$ ~~dei~~, ~~bdi~~		$acfgh\bar{b}\bar{d}\bar{e}\bar{i}$
$adefi$	12	12	$h, b,$ ~~bh~~, $c, g,$ ~~bch~~, ~~bgh~~, ~~ch~~, ~~bg~~, ~~cg~~, ~~cgh~~ $\mid$ ~~bcg~~		$adefi\bar{b}\bar{c}\bar{g}\bar{h}$
$bcdgi$	13	13	~~ah~~, $f,$ ~~eh~~, ~~af~~, $a, h,$ ~~fh~~, ~~aeh~~, $e,$ ~~ae~~, ~~afh~~, ~~aef~~ $\mid$		$bcdgi\bar{a}\bar{e}\bar{f}\bar{h}$

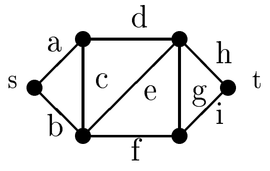

Fig. 1. The modified ARPA network

results obtained by applying PPM to this network. The terms not collected in the T_i because they imply some other term are crossed out. The character | separates the terms that are taken into account to build DP_i (at its left) from those that are not (at its right). An empty cell in the r_i column indicates that the set T_i is actually empty once the improvements have been achieved.

3.3 Comparison between PPM and six other preprocessing techniques

Soh and Rai provided experimental results to compare the performance of six different preprocessing techniques applied to minimal paths of many networks [13]. They used the CAREL (Computer Aided RELiability evaluator) algorithm [14] to obtain SDP terms. The table 4 reports a comparison between these six methods and PPM. The meanings of H, L, C are respectively the decreasing Hamming distance, Lexicographic ordering and increasing Cardinality (more details can be found in the cited papers).

Table 4. Comparison between six preprocessing techniques of [13] and PPM

Heuristics	Random	H	L	C	C + H	C + L	SODA
sequences of paths	adh	adh	acegi	bfi	adh	adh	adh
	adgi	bcdh	aceh	beh	beh	beh	bfi
	adefi	acfgh	acfgh	adh	bfi	bfi	beh
	aceh	aceh	acfi	bfgh	adgi	aceh	acfi
	acegi	adefi	adefi	begi	acfi	acfi	adgi
	acfgh	adgi	adgi	bcdh	begi	adgi	bcdh
	acfi	bfgh	adh	acfi	aceh	bcdh	bfgh
	bcdh	beh	bcdgi	aceh	bcdh	begi	aceh
	bcdgi	bcdgi	bcdh	adgi	begh	bfgh	begi
	beh	acfi	begi	bcdgi	adefi	acegi	acegi
	begi	acegi	beh	acfgh	acegi	acfgh	acfgh
	bfgh	bfi	bfgh	acegi	acfgh	adefi	adefi
	bfi	begi	bfi	adefi	bdfgi	bcdgi	bcdgi
#SDP	23	22	26	16	16	17	15

Table 5. The number of disjoint products given by some SDP techniques

SVI-methods				MVI-methods							
Abraham [8]	ALR [10]	ALW [10]	SLR [15]	CAREL [14]					Heidtman [5]	Singh [7]	SODA
				H	L	C	C+H	C+L			
71	61	59	55	101	76	39	41	41	41	37	35

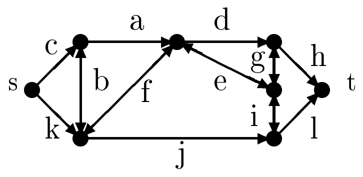

Fig. 2. Network of example 2

4 FURTHER COMPARISON BETWEEN SODA AND OTHER APPROCHES

4.1 A benchmark exercise

The second test example, pictured Fig. 2, is a 12-components system with 24 minimal paths. It is also one of the most frequently cited example [6–10,13,15]. The table 5 summarizes the number of disjoint products obtained by means of several SDP techniques. These techniques are splitted into two categories: those that use single variable inversion (SVI) and those that use multiple variables inversion (MVI). The meaning of the acronyms used in this table is as follows. ALR: Abraham-Locks Revised, ALW: Abraham-Locks-Wilson, SLR: Schneeweiss-Lin Revised. As for the previous example, the SODA procedure appears as the best algorithm to reduce the number of terms in the SDP resulting from the second network analysis.

4.2 The R_ν's networks

The third test example is the generic network R_ν of reference [5]. This network is made of ν squares joined side by side in a linear chain as shown Fig. 3. In that network, each edge corresponds to a component. Therefore, there are $n = 3\nu + 1$ components and $m = 2^\nu$ minimal s-t paths. The generic expres-

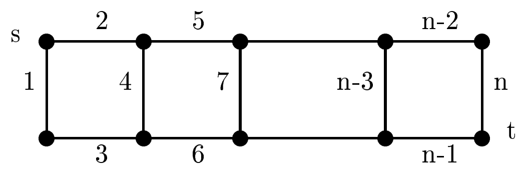

Fig. 3. General structure of the R_ν networks

sion of the number of minimum paths of each length is as follows.

$$L_\nu = \{\binom{\nu+1}{\nu}.[\nu+1], \binom{\nu+1}{\nu-2}.[\nu+3], \dots, \\ \dots, \binom{\nu+1}{p}.[2\nu+1-p]\}$$

where the binomial expressions denote the number of paths of the length given between brackets and $p = \nu - 2.\lfloor \nu/2 \rfloor$ indicates the parity of ν.

We applied the Heidtmann's procedure [5], so-called KDH, as well as SODA on the first R_ν's networks. The results are reported in table 6, where N denotes the number of disjoint products and t(s) the running time in seconds on a IBM computer (series 9021-820).

Some comments can be made above these results:

- As expected, the running times of both procedures increases exponentially as ν increases. This is a drastic limitation of "classical" SDP techniques.
- Due to the computation time needed by the preprocessing module PPM (surrounded with bracket in the last column), the overall running time of SODA is greater than the KDH one for small networks ($\nu < 7$), but it becomes attractive beyond.
- For all of the considered networks, SODA gives better results (in term of number of disjoint products) than KDH.

5 IMPLEMENTATION ISSUES

The implementation of a procedure such as SODA is mainly a matter of data structures. For that kind of methods that are highly combinational, the main problem is memory consumption (as opposed to time consumption). It is in general the case that, among two different implementations of the same method, the most efficient is the one that allocates the less memory. Based on these considerations, we propose in what follows some guidelines to implement SODA.

5.1 Implementation of PPM

We assume that the products of the initial set are encoded by means of lists of literals.

For each product P_i, we have to perform the following operations:

(i) Build, for each P_j $(i \neq j)$, the difference $T_{i,j} = P_j \setminus P_i$.
(ii) Remove the $T_{i,j}$'s that imply another $T_{i,j}$.
(iii) Remove the $T_{i,j}$'s that do not share any variable with the other $T_{i,j}$'s.
(iv) Compute the cardinals of $\cup_{j,k}(T_{i,j} \cap T_{i,k})$ and $\cup_j T_{i,j}$

First, it can avoided to build explicitely the $T_{i,j}$'s: the P_j's can be used for that purpose. It suffices to set a mark on the literals of the P_j's that belong to P_i (in order to shadow them). In this way, one gets a representation of the $T_{i,j}$'s without creating any new data structure. This can be achieved in linear time w.r.t. the size of the formula.

The second operation may remove the unwanted $T_{i,j}$'s just by setting a mark on these products. This operation requires two nested loops over the products. It is thus basically of quadratic complexity.

In order to isolate the $T_{i,j}$'s that do share any variable with the other $T_{i,j}$'s, it suffices to count the number of occurrences of each variable in the surviving products. The $T_{i,j}$'s that contain only variables with only one occurrence are those to be removed. This third operation can be therefore achieved in linear time.

The fourth operation is of linear complexity as well: the cardinal of $\cup_j T_{i,j}$ is just the number of variables that occur in the surviving products, and the cardinal of $\cup_{j,k}(T_{i,j} \cap T_{i,k})$ is the number of variables that occur more than once in the surviving products.

Therefore, the whole PPM method is of cubic complexity (since the four operations described above have to be repeated for each product and that the second one, which is the most costly, is of quadratic complexity). A cubic complexity induces a very quick growth of the running times as the size of the problem increases. However, the implementation we sketched does not require the allocation of additional data structures.

Table 6. Results of KDH [5] and SODA on some R_ν's networks

Characteristics	parameters	KDH	SODA
$\nu = 4$, $n = 13$, $m = 16$	N	34	26
$L_4 = \{5(5), 10(7), 1(9)\}$	t(s)	0.0011	0.0026 (0.0013)
$\nu = 5$, $n = 16$, $m = 32$	N	89	66
$L_5 = \{6(6), 20(8), 6(10)\}$	t(s)	0.0071	0.0148 (0.0067)
$\nu = 6$, $n = 19$, $m = 64$	N	233	181
$L_6 = \{7(7), 35(9), 21(11), 1(13)\}$	t(s)	0.059	0.083 (0.034)
$\nu = 7$, $n = 22$, $m = 128$	N	610	469
$L_7 = \{8(8), 56(10), 56(12), 8(14)\}$	t(s)	0.434	0.463 (0.178)
$\nu = 8$, $n = 25$, $m = 256$	N	1597	1191
$L_8 = \{9(9), 84(11), 126(13), 36(15), 1(17)\}$	t(s)	2.61	2.78 (1.08)
$\nu = 9$, $n = 28$, $m = 512$	N	3756	3133
$L_9 = \{10(10), 120(12), 252(14), 120(16), 10(18)\}$	t(s)	18.60	17.02(6.92)
$\nu = 10$, $n = 31$, $m = 1024$	N	10880	9020
$L_{10} = \{11(11), 3030(13), \ldots, 55(19), 11(21)\}$	t(s)	126.5	111 (47.1)

5.2 Implementation of DA

There are basically two ways of implementing DA. The first one is derived straight from the presentation of the method. It consists in building the $T_{i,j}$'s and then applying the de Morgan laws to get the DP_i's. We do not go further into the discussion of this way, for we gave already some guidelines in the previous section.

The second one is based on the Shannon decomposition. Consider the product P_i to be inserted into the SDP at a given step of the algorithm. There are two cases:

(i) Either P_i is pairwisely distinct from all of the products P_j's $(j < i)$. In that case, P_i can be added straight to the SDP.

(ii) Or P_i is not pairwisely distinct from all of the products P_j's $(j < i)$. In that case, it exists a variable x occurring in the P_j's $(j < i)$ but not in P_i and the problem can be reduced to insert first $x.P_i$ and second $\bar{x}.P_i$ into the SDP.

This means that the principle of DA can be implemented by means of a recursive procedure that explores implicitely a binary tree. Such procedures are extensively studied in the automated demonstration and artificial intelligence frameworks and efficient data structures and heuristics have been designed to implement them (see for instance [16]).

At this point, two remarks may be made. The first improvement is automatically embedded (without any additional cost) into such a recursive procedure. This is of a great importance because we just saw at the last section that it is costly. The second improvement is more difficult to achieve in a cheap way, although the principle of autark (detailed in [16]) could be an interesting alternative.

6 CONCLUSION

In this article, we presented an optimised procedure, so-called SODA, to generate disjoint products from a set of products representing either the minimal s-t paths of a reliability network or the minimal cutsets of a coherent fault tree.

We showed on a number of classical test examples that SODA performs better than other procedures proposed in the literature in terms of the number of terms of the results.

We discussed some implementation issues. We show that a procedure such as SODA can be interpreted in terms of a recursive procedure based on the Shannon decomposition. This opens new perspectives since this kind of procedures is extensively studied in the automated demonstration and artificial intelligence frameworks.

Anyway, we are convinced that the future of SDP methods depends on their ability to compete fairly with Binary Decision Diagrams techniques. Such a comparative study remains to do.

REFERENCES

1. K.R. Misra. *New Trends in System Reliability Evaluation.* Elsevier, 1993.
2. S. Rai, M. Veeraraghavan, and K.S. Trivedi. A Survey of Efficient Reliability Computation Using Disjoint Products Approach. *Networks*, 25:147–163, 1995.
3. L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal of Computing*, 8:410–421, 1979.
4. Y. Dutuit and A. Rauzy. Exact and Truncated Computations of Prime Implicants of Coherent and non-Coherent Fault Trees within Aralia. *Reliability Engineering and System Safety*, 58:127–144, 1997.
5. K.D. Heidtmann. Smaller Sums of Disjoint Products by Subproduct Inversion. *IEEE Transactions on Reliability*, 38:305–311, 1989.
6. M. Veeraraghavan and K.S. Trivedi. An Improved Algorithm for Symbolic Reliability Analysis. *IEEE Transactions on Reliability*, 40(3):347–358, 1991.
7. B. Singh. A procedure for generating sums of disjoint products. *Micro-electronic Reliability*, 33(15):2269–2272, 1993.
8. J.A. Abraham. An improved method for network reliability. *IEEE Transactions on Reliability*, R-28:58–61, 1979.
9. J.M. Wilson. An Improved Minimizing Algorithm for Sum of Disjoint Products. *IEEE Transactions on Reliability*, 39:42–45, 1990.
10. M.O. Locks and J.M. Wilson. Note on Disjoint Products Algorithms. *IEEE Transactions on Reliability*, 41(1):81–84, 1992.
11. S. Rai and J.L.. Trahan. Pseudo-boolean Approach to Solve Reliability Problems. In *Proceedings of the ESREL'97 Conference*, volume 3, pages 2079–2086, 1997.
12. T. Bouhoufani. *Contribution à la construction et au traitement des arbres de défaillance.* Thèse de doctorat, Université Bordeaux I, 1993.
13. 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.
14. S. Soh and S. Rai. CAREL: Computer Aided RELiability evaluator for distributed computing networks. *IEEE Transactions on Parallel and Distributed Systems*, 2:199–213, 1991.
15. H.H. Liu, W.T. Yang, and C.C. Liu. A improved minimizing algorithm for the summation of disjoint products by Shannon's expansion. *Micro-electronic Reliability*, 33(4):599–613, 1993.
16. 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.