A BDD-based fast heuristic algorithm for disjoint decomposition

Tomas Bengtsson
Embedded systems / ING
Jönköping University
Jönköping, Sweden
beto@ing.hj.se

Andrés Martinelli
IMIT / KTH
Royal Institute of Technology
Stockholm, Sweden
andres@imit.kth.se

Elena Dubrova
IMIT / KTH
Royal Institute of Technology
Stockholm, Sweden
elena@imit.kth.se

Abstract— This paper presents a heuristic algorithm for disjoint decomposition of a Boolean function based on its ROBDD representation. Two distinct features make the algorithm feasible for large functions. First, for an n-variable function, it checks only $O(n^2)$ candidates for decomposition out of $O(2^n)$ possible ones. A special strategy for selecting candidates makes it likely that all other decompositions are encoded in the selected ones. Second, the decompositions for the approved candidates are computed using a novel IntervalCut algorithm. This algorithm does not require re-ordering of ROBDD. The combination of both techniques allows us to decompose the functions of size beyond that possible with the exact algorithms. The experimental results on 582 benchmark functions show that the presented heuristic finds 95% of all decompositions on average. For 526 of those functions, it finds 100% of the decompositions.

I. Introduction

The *disjoint decomposition* of a Boolean function is a representation of type $f(X) = h(g(Y),Z)$ with Y and Z being sets of variables partitioning the set X. Disjoint decomposition has many applications in computer science and discrete mathematics, including logic synthesis [1, Chapter 13], combinatorial optimization problems over graphs and networks [2], reliability theory [3] and game theory [4].

This wide range of applications makes it important to have efficient algorithms for finding all, or at least some, decompositions for a given structure. Fast decomposition algorithms are known for binary relations and graphs [5, 6, 7]. For Boolean functions, however, the existing methods either involve the solution of an NP-complete problem (as in [8]) or have exponential running time [9, 10, 11, 12]. More recent ROBDD-based decomposition algorithms, including [13, 14, 15], show much better average-time performance.

This paper presents a heuristic algorithm targeting to find all disjoint decompositions of an n-variable Boolean function represented by a ROBDD. The heuristic is based on two properties: (1) all decompositions of a Boolean function (which can be $O(2^n)$) can be uniquely described by so called *strong* decompositions (which are only $O(n)$); (2) there exist a best variable ordering for a ROBDD in which the variables Y from any strong decomposition $f(X) = h(g(Y),Z)$ are adjacent.

If we had such a best ordering, we could examine all its linear intervals to find which Y results in a decomposition $f(X) = h(g(Y),Z)$. However, computing best orderings is infeasible for large functions. The algorithm presented in this paper is *heuristic* because it starts from a "good" ordering which is not necessarily keeping the variables Y adjacent. The experimental results show that if *sifting* ordering algorithm [16] is used to get a "good" initial order, then our heuristic finds 95% of all decompositions on average.

II. Previous work

The first major investigation on the subject was carried out by Ashenhurst [17]. He studied simple disjoint decomposition $f(X) = h(g(Y),Z)$ for Boolean functions $f,g,h : B^n \to B$, where $B = \{0,1\}$. Ashenhurst's fundamental contribution is a theorem which states that any Boolean function has a unique *disjoint tree-like decomposition* such that all possible simple disjoint decompositions of f are exhibited.

Curtis [18] and Roth and Karp [19] extended Ashenhurst theory to the decomposition of type $f(X) = h(g(Y),Z)$ with g,h being multiple-valued functions of type $g\,B^{|Y|} \to M$ and $h\,M \times B^{|Z|} \to B$, where $M = \{0,1,...,m-1\}$. The function g can be encoded by $k = \lceil \log_2 m \rceil$ Boolean functions $g_1, g_2, \dots, g_k$, giving a decomposition of the form $f(X) = h(g_1(Y),\dots,g_k(Y),Z)$, often referred to as *Roth-Karp* decomposition. Unfortunately Ashenhurst's main theorem does not extend directly to multiple-valued functions (for a counterexample see chapter 4 of [1]). A consequence of this is that there is no unique disjoint tree-like Roth-Karp decomposition. Von Stengel [20] has defined a class of multiple-valued functions for which Ashenhurst's main theorem holds.

Early algorithms for decomposition used *decomposition charts* [17], [18]. The decomposition chart for $f(Y,Z)$ is a two-dimensional table where the columns represent all combinations of the variables from the set Y and the rows represent all combinations of the variables from the set Z. The set Y is a bound set if and only if the chart has *column multiplicity* at most two, i.e. there are at most two distinct columns in the chart [17].

In a short time, decomposition charts were abandoned in favor of *cube* representation [21]. The task of computing column multiplicity on charts was replaced by the task of computing *compatible classes* for a set of cubes. Two assignments $x_1, x_2 \in B^{|Y|}$ are said to be *compatible with respect to the reference function* $f(Y,Z)$ if, for all $y \in B^{|Z|}$ such that $f(x_1,y)$ and $f(x_2,y)$ are defined, $f(x_1,y) = f(x_2,y)$ [21]. The set Y is a bound set if and only if $B^{|Y|}$ can be partitioned into $k \leq 2$ mutually compatible classes [21]. If $f(X)$ is completely specified, then compatibility is an equivalence relation and k is the number of equivalence classes. It is easy to see the one-to-one mapping between a column in a decomposition chart and a compatible class.

Due to the exponential size of decomposition charts and cube representations, early decomposition algorithms were rarely applied to large practical circuits. Instead, *algebraic* methods were used [22]. ROBDDs [23] made possible developing new algorithms for decomposition, feasible for much larger functions than previously possible.

In a ROBDD, the column multiplicity can be easily computed by moving the variables Y to the upper part of the graph and checking the number of children below the boundary line, usually called *cut* line. The decomposition $f(X) = h(g(Y),Z)$ exists if and only if there are only two children below the cut line [24].

This approach has been adopted by a number of BDD-based decomposition algorithms [24, 25, 26, 27]. In [28], a strategy telling which variables are more likely to be in a bound set is used to improve this. Stanion and Sechen [29] used cut to find *quasi-algebraic* decomposition of the form $f(X) = g(Y) \odot h(Z)$, where "$\odot$" is any binary Boolean operation and $|Y \cup Z| = k$ for some $k \geq 0$. This type decomposition is often referred to as *bi-decomposition* [30, 31].

BDD-based decomposition algorithms following cut-strategy proved to be orders of magnitude faster than those based on decomposition charts and cube representations. However, they require reordering of variables of BDD to move the variables on the top or to check bi-decompositions for partitionings which are not consistent with the variable order. As an alternative, a number of methods use the fact that BDDs themselves are a decomposed representation of the function and exploit the structure of BDDs, rather than cut, to find disjoint decompositions. Karplus [32] extended the classical concept of *dominator* on graphs [33] to 0,1-dominators on BDDs. A node v is a 1-dominator (0-dominator) if every path from the root to one (zero) terminal node contains v. If v is a 1-dominator, then the function represented by the BDD possesses a conjunctive (AND) decomposition. If v is a 0-dominator, then the function can be decomposed disjunctively (OR). This idea was extended by Yang et al [34] to XOR-type decompositions and to more general type of dominators. Minato and De Micheli [14] presented an algorithm which computes disjoint decompositions by generating irreducible sum-of-product for the function from its BDD and applying factorization. In [35], Sasao presented a method to reduce the search space for decompositions using BDDs and look-up tables. The algorithm of Bertacco and Damiani [13] makes a single traversal of the BDD to identify the decomposition of the co-factors and then combine them to obtain the decomposition for the entire function. The algorithm is impressively fast; however, as Sasao has observed in [36], it fails to compute some of the disjoint decompositions. This problem was corrected by Matsunaga [15], who added the missing cases in [13] allowing to treat the OR/XOR functions correctly. The algorithm [15] appears to be the fastest of existing exact algorithms for finding all disjoint decompositions.

A strategy using a test pattern generator working on net lists to find bound sets was presented by Sasao [37]. This method works well to detect bound sets containing only few variables or all variables except a few.

III. New heuristic algorithm

The new heuristic algorithm is based on the following two properties.

Property 1 *All disjoint decompositions of an n-variable Boolean function can be uniquely described by a certain subset of strong decompositions of size $O(n)$.*

Property 2 *There exist a best variable ordering for a ROBDD for f in which the variables Y from any strong decomposition $f(X) = h(g(Y),Z)$ are adjacent.*

Property 1 follows from the results of [20][1]. We define strong decomposition and describe the results of [20] in Section A. Property 2 follows from the main theorem of [39].

The presented algorithm examines all linear intervals of variables from a given ordering of a ROBDD and, for each interval Y, checks whether it is a bound set. In this paper we use ROBDDs without complemented edges. The procedure **IntervalCut** described in Section B, is used to perform the checking as well as to compute the functions g and h in the resulting decomposition $f(X) = h(g(Y),Z)$.

A. Properties of the disjoint decomposition

This section describes the properties of the disjoint decomposition from [20], implying Property 1. The formulation of the definitions and theorems is adjusted to the notation of this paper.

Definition 1 *A bound set Y of $f(X)$, $Y \subset X$, is strong if any other bound set of $f(X)$ is either a subset of Y, a superset of Y, or disjoint to Y.*

The partial order induced by set theoretical inclusion between pairs of strong bound sets of f defines a tree.

Definition 2 *The decomposition tree $T(f)$ of $f(X)$ is a tree whose nodes represent all strong bound sets of $f(X)$, related by inclusion. Any node has two labels:*
(a) a type, which is either "prime" or "full",
(b) an associated function.

[1] A brief translation of the main results of [20] to English is given in [38].

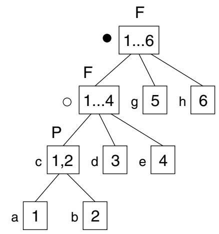

Fig. 1. Example of a decomposition tree. The digits inside the boxes represent the indices of the variables associated with the bound set.

The following Theorem shows how decompositions of a function can be derived from its decomposition tree and characterizes the functions associated with the nodes. It also states that the decomposition tree is unique for a given function (up to isotopy/isomorphy). Remind that two Boolean functions are *isotopic* if they are identical up to complementation of variables or function values. Two binary operations $\circ$ and $\bullet$ are *isomorphic* if there is a bijection $\phi : B \rightarrow B$ such that $\phi(a \circ b) = \phi(a) \bullet \phi(b)$.

Theorem 1 *Let $T(f)$ be the decomposition tree of a Boolean function $f(X)$ with support set X. Let $Y_1,\ldots,Y_k$ be the children of the root X. Then $f(X)$ has a decomposition of type*

$$f(X) = h(g_1(Y_1), g_2(Y_2), \ldots, g_k(Y_k))$$

for functions $g_i : B^{|Y_i|} \rightarrow B$ ($1 \leq i \leq k$) and $h : B^k \rightarrow B$ where
(a) h is non-decomposable if X is labeled “prime”,
(b) h is an associative and commutative Boolean operation if X is labeled “full”,
(c) h is unique up to isotopy in (a) and up to isomorphy in (b).

Example 1 *An example of a decomposition tree is shown in Figure 1. Abbreviations “P” and “F” stand for labels “prime”, and “full”, respectively. Letters a,b,c,d,e,g,h denote the functions associated with the nodes, whereas $\bullet$ and $\circ$ denote operations. In accordance with the tree, the complete disjoint decomposition of the function is*

$$f(x_1,\ldots,x_6) = (c(a(x_1), b(x_2)) \circ d(x_3) \circ e(x_4)) \bullet g(x_5) \bullet h(x_6)$$

with $\bullet$ and $\circ$ being associative and commutative Boolean operations. a,b,c,d,e,g,h are non-decomposable Boolean functions. In this case all those functions except c are unary Boolean functions (identity or complement).

Theorem 1 shows that the decompositions associated with strong bound sets uniquely represent all disjoint decompositions of a function. It was proved in [40] that the number of strong bound sets of an n-variable Boolean function is $O(n)$, while the number of all bound sets is $O(2^n)$.

IntervalCut(G,a,b)
input: ROBDD G of $f(X)$, two cuts $cut(a)$ and $cut(b)$, $a < b$, $a,b \in \{0,\ldots,n\}$.
output: ”not a bound set” if the set of variables Y between $cut(a)$ and $cut(b)$ is not a bound set of $f(X)$; functions g and h if Y is a bound set resulting in $f(X) = h(g(Y),Z)$.

for *all* $v \in cut\_set(a)$
    if $(|cut\_set(b_v)| > 2)$
        return(*”not a bound set”*);
for *all* $v_1, v_2, \ldots, v_k \in cut\_set(a)$
    if $(g_{v_i} \neq g_{v_{i+1}})$ */* up to complementation */*
        return(*”not a bound set”*);
h = substitute each subgraph g_v, $\forall v \in cut\_set(a)$, by a node;
$g = g_v$;
return(g,h);

Fig. 2. Pseudo code of the **IntervalCut** procedure.

B. IntervalCut procedure for finding bound sets

Let V be a set of nodes of a ROBDD G of an n-variable function $f(X)$. Every non-terminal node $v \in V$ has an associated variable index, $index(v) \in \{1,\ldots,n\}$. The index of the root node is 1. In order to have a unified notation in the proof of the main result, we assume that the terminal nodes also have an index, which is $n+1$.

Suppose that all nodes with $index \leq i$ are in the upper part of the graph and all nodes with $index > i$ are in the lower part of the graph, for some $i \in \{1,\ldots,n\}$. The boundary line between the upper and lower parts of the graph is called $cut(i)$. If the number of nodes with $index > i$ which are children of the nodes above the $cut(i)$ is two, then the set of variables $Y = \{x_1,\ldots,x_i\}$ is a bound set [25].

One possibility to check whether a set of variables Y is a bound set is to move the variables Y to the top of the ROBDD and then check the number of children below $cut(|Y|)$, as in [25, 26]. However, re-ordering is computationally expensive. Instead, we have developed a procedure, called **Interval Cut** which checks whether a given linear interval of variables of a ROBDD is a bound set without reordering. To describe the procedure, we first introduce some definitions.

Suppose the variables Y lie between two cuts, $cut(a)$ and $cut(b)$, such that $a < b$, $a,b \in \{0,\ldots,n\}$. Let $cut\_set(a)$ denote a set of nodes $v \in G$ with indices $a < index(v) \leq b$ which are children of the nodes above the $cut(a)$ of G. Let G_v stand for a ROBDD rooted at some $v \in cut\_set(a)$. Then, $cut\_set(b_v)$ is the set of nodes $u \in G_v$ with indices $b < index(u) \leq n+1$ which are children of the nodes of G_v above the $cut(b)$. If $|cut\_set(b_v)| = 2$, then g_v is a Boolean function represented by the subgraph rooted at v whose terminal nodes are obtained by replacing the two nodes of $cut\_set(b_v)$. The resulting g_v is unique up to complementation.

Using this notation, we can describe the pseudo code of the algorithm **IntervalCut**(G,a,b) as shown in Figure 2. Next, we prove that it computes the decompositions correctly.

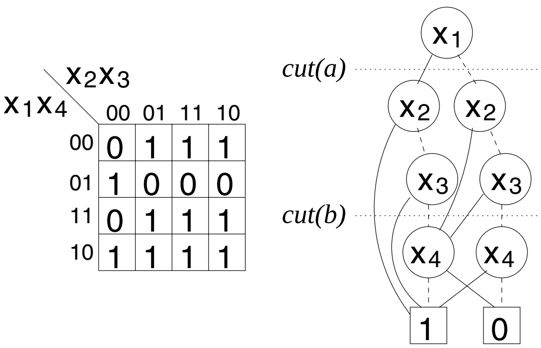

Fig. 3. Example of a four-variable function with a bound set $\{x_2, x_3\}$.

Theorem 2 *Algorithm* **IntervalCut**(G, a, b) *computes the decomposition* $f(X) = h(g(Y), Z)$.

Proof: Let Y be the variables between $cut(a)$ and $cut(b)$, Z_1 be the variables above $cut(a)$ and Z_2 be the variables below $cut(b)$. We have $Z_1 \cup Z_2 = Z$ and $Y \cup Z = X$.

Let $k_v(Z_1)$ be a function which is a sum of all the paths leading to a node $v \in cut\_set(a)$. Then f can be co-factored with respect to k_v as

$$f(X) = \sum_{\forall v \in cut\_set(a)} k_v(Z_1) \cdot f|_{k_v}(Y, Z_2) \qquad (1)$$

If $|cut\_set(b_v)| = 2$, then Y is a bound set for $f|_{k_v}$ so it can be decomposed as

$$f|_{k_v}(Y, Z_2) = h_v(g_v(Y), Z_2) \qquad (2)$$

for some h_v, g_v. Furthermore, if for all $v \in cut\_set(a)$ the functions g_v are equal up to complementation, then we can denote g_v by g and write (2) as

$$f|_{k_v}(Y, Z_2) = h_v(g(Y), Z_2) \qquad (3)$$

From (1) and (3) we can conclude that f can be represented as

$$f(X) = h(g(Y), Z)$$

with $h = \sum_{\forall v \in cut\_set(a)} k_v \cdot h_v$.

□

Let $max(|g_v|)$ be the size of the largest subgraph representing g_v, for some $v \in cut\_set(a)$. Since substitution of a ROBDD by a node is a constant-time operation, the complexity of the pseudo code in Figure 2 is $O(|cut\_set(a)| \cdot max(|g_v|))$.

Example 2 *Figure 3 shows the Karnaugh map and the ROBDD for a four-variable function. The two nodes labeled by* x_2 *belong to* $cut(a)$. *The functions rooted at these nodes have the the same structure down to* $cut(b)$. *For each of these two functions, there are two nodes below* $cut(b)$ *with parents above* $cut(b)$. *These two facts are true if and only if* $\{x_2, x_3\}$ *is a bound set.*

IV. Experimental results

To make a thorough evaluation of the presented heuristic, we have implemented an exact decomposition algorithm[2] from [38] and applied both, exact and heuristic versions, to *iwls*93 benchmark set. For all single outputs, for which the exact algorithm did not time out[3], 582 in total, we have computed the total number of strong bound sets found by each algorithm. In the first set of experiments, we used *sifting* ordering algorithm [16] to get a good initial order for ROBDDs. The heuristic algorithm has succeeded to find 95% of all the decompositions on average. For 526 of those 582 single-output functions, it found 100% of the decompositions. In the second set of experiments, we switched the sifting off, and build ROBDDs using the breadth first traversal order from the benchmark's circuit description. For 191 functions out of 582 the result got worse (by 57% on average). Nevertheless, the heuristic still found all the decompositions for 365 functions.

We have also applied the presented heuristic to the benchmarks reported in [14], [13] and [15]. The results are summarized in Table I. Column 4 shows how many non-trivial strong bound sets are found for each benchmark by our algorithm. Every output is handled as a separate function. The number given in Column 4 is the total sum of bound sets for all the outputs. Columns 5-8 show runtime comparison. Our experiments were run on Sun Ultra 60 operating with two 360 MHz CPU and with 1024 MB RAM main storage. The algorithm [14] uses a SUN Ultra 30, [13] uses a PC equipped with 150 MHz Pentium and 96 MB RAM main storage and [15] uses a PC with Pentium-II 233Mhz processor.

V. Conclusion

This paper presents a heuristic algorithm for finding disjoint decompositions of Boolean functions. Benchmark experiments demonstrate the effectiveness of the described technique. This algorithm can be adopted for ROBDDs with complemented edges with only slight modifications.

Our on-going work includes extension of the presented algorithm to Roth-Karp decomposition [19]. We are also investigating a possibility of combining **IntervalCut** with decomposition algorithms exploiting the structure of BDDs, like [13] and [15].

References

[1] S. Hassoun and T. Sasao, eds., *Logic Synthesis and Verification*. Kluwer Academic Publishers, 2002.

[2] R. H. Möhring and F. J. Radermacher, "Substitution decomposition of discrete structures and connections to combinatorial optimization," *Ann. Discrete Math*, vol. 19, pp. 257–264, 1984.

[3] Z. W. Birnbaum and J. D. Esary, "Modules of coherent binary systems," *SIAM Journal of Applied Math.*, vol. 13, pp. 444–451, 1965.

[2] We have chosen [38] because this algorithm actually builds decomposition trees. It computes only $O(n)$ strong bound sets which are the nodes of $T(f)$.

[3] Time limit 30 min per circuit.

name	in	out	N of bound sets	CPU time, sec presented heuristic	exact alg [14]	exact alg [13]	exact alg [15]
alu2	10	6	3	0.0002	-	0.28	-
alu4	14	8	2	0.0009	-	0.37	0.15
apex1	45	45	83	0.008	59.0	1.01	-
apex2	38	3	16	0.001	5.9	1.14	-
apex3	54	50	23	0.008	44.3	-	-
apex4	9	19	4	0.002	-	0.33	-
apex5	114	88	196	0.032	-	2.34	-
apex6	135	99	258	0.008	13.1	2.62	0.41
apex7	49	37	96	0.006	1.7	1.03	0.37
b9	41	21	49	0.001	-	-	0.02
C432	36	7	10	0.002	415.4	1.23	0.28
C499	41	32	68	5.2	-	83.47	8.80
C880	60	26	45	0.046	-	2.71	0.92
C1355	41	32	0	5.2	-	91.25	8.87
C1908	33	25	15	0.23	-	7.58	1.42
C3540	50	22	18	2.8	-	21.1	3.48
cmb	16	4	4	0.002	-	0.36	-
CM42	4	10	10	0.0006	-	0.15	-
CM85	11	3	15	0.0003	-	0.27	-
CM150	21	1	1	<0.0001	-	0.51	-
comp	32	3	47	0.002	-	0.71	-
count	35	16	47	0.007	-	0.73	0.01
dalu	75	16	42	0.015	>0.8	-	-
des	256	245	688	0.041	-	-	0.36
e64	65	65	63	0.51	-	1.31	-
f51m	8	8	6	0.0004	-	0.26	-
frg2	143	139	532	0.032	19.2	2.86	0.15
k2	45	45	85	0.008	-	1.04	-
lal	26	19	57	0.002	-	0.55	-
misex2	25	18	29	0.003	-	0.57	-
mux	21	1	1	0.0001	-	0.48	-
pair	173	137	725	0.040	-	4.02	7.36
PARITY	16	1	1	0.001	-	0.38	-
rot	135	107	296	0.039	-	22.62	-
seq	41	35	135	0.009	67.8	1.10	-
s298	17	20	15	0.0004	-	0.40	-
s420	35	18	18	0.007	-	0.75	-
s444	24	27	65	0.001	-	0.54	-
s526	24	27	45	0.002	-	0.52	-
s641	54	42	138	0.003	-	1.12	-
s832	23	24	37	0.003	-	0.54	-
s953	45	52	40	0.003	-	20.97	-
s1196	32	32	33	0.002	-	0.71	-
s1238	32	32	33	0.002	-	0.75	-
s1423	91	79	38	0.066	-	12.48	-
s1488	14	25	38	0.002	-	0.36	-
s1494	14	25	38	0.002	-	0.34	-
term1	34	10	65	0.002	-	0.75	-
too_large	38	3	17	0.001	>1.0	-	0.09
ttt2	24	21	44	0.002	-	0.55	-
vda	39	17	30	0.003	>0.5	0.4	-
x4	94	71	180	0.008	-	1.90	-

TABLE I

EXPERIMENTAL RESULTS; "−" INDICATES THAT INFORMATION FOR THE BENCHMARK IS NOT PROVIDED; ">" INDICATES THAT INFORMATION IS ONLY PROVIDED FOR ONE OF THE OUTPUTS.

[4] L. S. Shapley, "Solutions of compound simple games," in *Advances in Game Theory*, no. 52 in Ann. of Math. Study, pp. 267–280, Princeton University Press, 1964.

[5] Z. W. Birnbaum and R. H. Möhring, "A fast algorithm for the decomposition of graphs and posets," *Math. Oper. Res*, pp. 170–177, 1984.

[6] W. H. Cunningham, "Decomposition of directed graphs," *SIAM Journal of Algebraic and Discrete Methods*, vol. 3, pp. 214–221, 1982.

[7] M. Habib and M. C. Maurer, "On the x-join decomposition for undirected graphs," *Journal of Appl. Discr. Math.*, vol. 3, pp. 198–205, 1979.

[8] L. J. Billera, "On the composition and decomposition of clutters," *Journal of Comb. Theory*, vol. 11, pp. 234–241, 1971.

[9] J.-P. Deschamps, "Binary simple decomposition of discrete functions," *Digital Processes*, vol. 1, pp. 123–130, 1975.

[10] V. Y. Shen and A. C. McKellar, "An algorithm for the disjunctive decomposition of switching functions," *IEEE Trans. Computers*, vol. C-19, pp. 239–245, 1970.

[11] V. Y. Shen, A. C. McKellar, and P. Weiner, "A fast algorithm for the disjunctive decomposition of switching functions," *IEEE Trans. Computers*, vol. C-20, pp. 239–246, 1970.

[12] A. Thayse, "A fast algorithm for the proper decomposition of boolean functions," *Philips Res. Rep.*, vol. 27, pp. 140–147, 1972.

[13] V. Bertacco and M. Damiani, "The disjunctive decomposition of logic functions," in *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 78–82, 1997.

[14] S. Minato and G. D. Micheli, "Finding all simple disjunctive decompositions using irredundant sum-of-products forms," in *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pp. 111–117, 1998.

[15] Y. Matsunaga, "An exact and efficient algorithm for disjunctive decomposition," in *Proceedings of SASIMI'98*, pp. 44–50, 1998.

[16] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," in *Proceeding of IEEE/ACM International Conference on Computer-Aided Design*, vol. 29, pp. 42–47, 1993.

[17] R. Ashenhurst, "The decomposition of switching functions," in *Proceedings International Symp. Theory of Switching*, vol. 29, pp. 74–116, 1959.

[18] H. A. Curtis, *A New Approach to the Design of Switching Circuits*. Princeton, New Jersey: D. van Nostrand company, 1962.

[19] J. P. Roth and R. M. Karp, "Minimization over Boolean graphs," *IBM Journal*, vol. 6, pp. 227–238, April 1962.

[20] B. von Stengel, "Eine dekompositionstheorie für mehrstellige funktionen," in *Mathematical Systems in Economics*, vol. 123, Anton Hain, Frankfurt, 1991.

[21] R. M. Karp, "Functional decomposition and switching circuit design," *Journal of Soc. Indust. Appl. Math.*, vol. 11, pp. 291–335, June 1963.

[22] R. K. Brayton and C. McMullen, "The decomposition and factorization of Boolean expression," in *Proceedings of the IEEE International Symposium of Circuits and Systems*, pp. 49–54, IEEE, 1982.

[23] R. E. Bryant, "Graph-based algorithm for boolean function manipulation," *Transactions on Computers*, vol. C-35, pp. 677–691, 1986.

[24] T. Sasao, *FPGA design by generalized functional decomposition*, pp. 233–258. Kluwer Academic Publishers, 1993.

[25] Y.-T. Lai, K.-R. Pan, and M. Pedram, "BDD-based function decomposition: algorithms and implementation," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 977–990, 1996.

[26] S.-C. Chang, M. Marek-Sadowska, and T. Hwang, "Technology mapping for TLU FPGA's based on decomposition of binary decision diagrams," *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1226–1235, 1996.

[27] H. Sawada, S. Yamashita, and A. Nagoya, "Restructuring logic representations with easily detectable simple disjunctive decompositions," in *Proceedings of Design Automation Conference*, pp. 755–759, IEEE, 1998.

[28] T. Sasao and K. Kurimoto, "Three parameters to find functional decompositions," in *Proceedings of Asia and South Pacific Design Automation Conference*, pp. 259–264, 2000.

[29] T. Stanion and C. Sechen, "Quasi-algebraic decompositions of switching functions," in *Proceedings of Sixteenth Conference on Advanced Research in VLSI*, pp. 358–367, IEEE, 1995.

[30] S. Yamashita, H. Sawada, and A. Nagoya, "New methods to find optimal non-disjoint bi-decompositions," in *Proceedings of Design Automation Conference*, pp. 59–68, IEEE, 1998.

[31] A. Mishchenko, B. Steinbach, and M. Perkowski, "An algorithm for bi-decomposition of logic functions," in *Proceedings of Design Automation Conference*, pp. 103–108, IEEE, 2001.

[32] K. Karplus, *Using if-then-else DAGs for multi-level logic minimization*. University of California Santa Cruz: Technical Report UCSC-CRL-88-29, 1988.

[33] T. Lengauer and R. E. Tarjan, “A fast algorithm for finding dominators in a flowgraph,” *Transactions of Programming Languages and Systems*, vol. 1, pp. 121–141, July 1979.

[34] C. Yang, V. Singhal, and M. Ciesielski, “Bdd decomposition for efficient logic synthesis,” in *Proceedings of International Conference on Computer Design*, pp. 626–631, 1999.

[35] T. Sasao, “Totally undecomposable functions: Applications to efficient multiple-valued decompositions,” in *Proceedings of IEEE International Symposium on Multiple-Valued Logic*, pp. 59–65, IEEE, 1999.

[36] T. Sasao and M. Matsuura, “DECOMPOS: An integrated system for functional decomposition,” in *Proceedings of ACM/IEEE International Workshop on Logic Synthesis*, 1998.

[37] T. Sasao and S. Kajihara, “Functional decompositions using an automatic test pattern generator and a logic simulator,” in *Proceedings of ACM/IEEE International Workshop on Logic Synthesis*, ACM/IEEE, 1999.

[38] E. V. Dubrova, J. of C. Muzio, and B. von Stengel, “Finding composition trees for multiple-valued functions,” in *Proceedings of 27th International Symposium on Multiple-Valued Logic*, pp. 19–26, IEEE, 1997.

[39] E. Dubrova, “Composition trees in finding best variable orderings for ROBDDs,” in *Proceedings of Design, Automation & Test in Europe Conference*, p. 1084, 2002.

[40] R. H. Möhring, “Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and Boolean functions,” *Annals of Operations Research*, vol. 4, pp. 195–225, June 1985.