2001-08

Bi-partition of shared binary decision diagrams

Sasao, Tsutomu; Butler, Jon T.; Matsuura, Munehiro;
Iguchi, Yukihiro

T. Sasao and J. T. Butler, "Worst and best irredundant sum-of-products expressions,
" IEEE Transactions on Computers, Vol. 50, No. 9, Sept. 2001, pp. 935-948.
https://hdl.handle.net/10945/35863

Bi-Partition of Shared Binary Decision Diagrams

Munehiro Matsuura[1], Tsutomu Sasao[1,2], Jon T. Butler[3], and Yukihiro Iguchi[4]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology
[3]Department of Electrical and Computer Engineering, Naval Postgraduate School
[4]Department of Computer Science, Meiji University

August 24, 2001

Abstract

A shared binary decision diagram (SBDD) represents a multiple-output function, where nodes are shared among outputs. A partitioned SBDD usually consists of two or more SBDDs that often share nodes. The separate SBDDs are optimized independently, often resulting in a reduction in the number of nodes over a single SBDD. We show a method for partitioning a single SBDD into two parts that reduces the node count. Among the benchmark functions tested, a node reduction of up to 16% is realized.

Keyword: Shared binary decision diagram, SBDD, bi-partition, multiple-output function, decomposition.

1 Introduction

Various methods exist to represent multiple-output functions [17, 18, 19]. Among them, shared binary decision diagrams (SBDDs) are most commonly used, since their sizes are usually smaller [19] than other types of BDDs, such as multi-terminal binary decision diagrams (MTB-DDs) [17] and BDDs for characteristic functions (BDDs for CFs) [1, 21]. However, for some applications, SBDDs are still large and more compact representations are required.

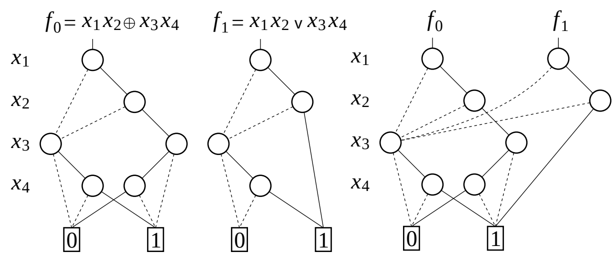
In this paper, we propose a method to represent multiple-output functions, partitioned SBDDs. Each part represents a set of outputs, and is optimized independently. Such BDDs are considered as a special case of partitioned BDDs [13, 14, 6] and free BDDs (FBDDs) [7, 8].

Applications of partitioned SBDDs are similar to that of partitioned BDDs and FBDDs.

1) Hardware synthesis. Replace each non-terminal node of an SBDD by a multiplexer (MUX), forming a network for F. This is used to design multiplexer-type FPGAs [4] and pass-transistor logic [23].

2) Software synthesis [2, 19]. Replace each non-terminal node by an *if then else* statement, forming a branching program for F.

3) Verification [13, 14, 6]. In verification, a monolithic BDD may be too large to be stored in a computer memory. So, in this case, small BDDs are generated sequentially, and each component is checked one by one.



(a) Before sharing (14 nodes). (b) After sharing (10 nodes).

Figure 2.1: Shared BDD.

2 Partition of SBDDs

An SBDD is considered as a compact BDD representation of a multiple-output function, since nodes can be shared among many outputs.

Example 2.1 *Consider the two-output function:*

$$f_0 = x_1x_2 \oplus x_3x_4,$$
$$f_1 = x_1x_2 \vee x_3x_4.$$

In this case, $\pi = (x_1, x_2, x_3, x_4)$ is a good ordering of the input variables for both f_0 and f_1. Note that some nodes can be shared between f_0 and f_1, as shown in Fig. 2.1. In the figures, dotted lines denote 0-edges, while solid lines denote 1-edges. *(End of Example)*

In an SBDD, there can be only one ordering of input variables for all output functions. Thus, the size tends to be large when the individual functions have different optimal orderings of the input variables.

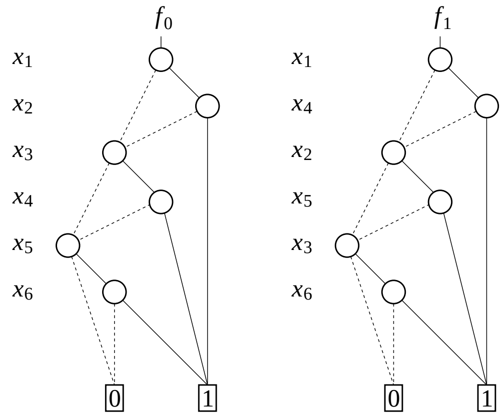


Figure 2.2: Pair of BDDs, which is smaller than SBDD.

Example 2.2 *Consider the BDDs of functions:*

$$f_0 = x_1x_2 \vee x_3x_4 \vee x_5x_6,$$
$$f_1 = x_1x_4 \vee x_2x_5 \vee x_3x_6.$$

In this case, $\pi_0 = (x_1, x_2, x_3, x_4, x_5, x_6)$ *is an optimal ordering for* f_0*, while* $\pi_1 = (x_1, x_4, x_2, x_5, x_3, x_6)$ *is an optimal ordering for* f_1*. Fig. 2.2 shows the corresponding BDDs. Together, they require a total of* $8 \times 2 = 16$ *nodes. On the other hand, a minimum SBDD for* $\{f_0, f_1\}$ *requires 17 nodes. In this case, the pair of separately optimized BDDs is smaller than the optimized monolithic SBDD for* $\{f_0, f_1\}$*. This is an example of a partitioned BDD that is smaller than the monolithic SBDD. (End of Example)*

From these examples, we can formulate the following:

Problem 2.1 *(Partitioned SBDD)*
Given a multiple-output function F*, represent* F *by a set of SBDDs so that the total number of nodes is minimized, where each SBDD is optimized independently.*

3 Bi-partition of SBDDs

Before considering a general partitioning problem, we start with a simpler problem, i.e., the *bi-partition problem*. We can obtain a general partition by applying bi-partitions recursively.

Definition 3.1 *Let* $F = \{f_0, f_1, \ldots, f_{m-1}\}$ *be the set of the output functions.* $size(SBDD, F, \pi)$ *denotes the number of nodes in the SBDD for* F*, where* π *is the ordering of the input variables.* $size(SBDD, F)$ *denotes the number of nodes in the minimum SBDD for* F *over all orderings.*

Then, we can formulate,

Problem 3.2 *(Bi-partition of an SBDD)*
Given a multiple-output function $F = \{f_0, f_1, \ldots, f_{m-1}\}$*, represent* F *by a pair of SBDDs so that* $size(SBDD, F_1) + size(SBDD, F_2)$ *is minimized, where* $F_1 \cup F_2 = F$*,* $F_1 \cap F_2 = \phi$*, and* $F_1 \neq \phi$*.*

It is possible that, for all non-trivial bi-partitions, the total number of nodes in the partitioned SBDD is greater than in the original one. In this case, we accept the original given SBDD as the best we can do. This is represented as the trivial partition $F = (F, \phi)$, where ϕ is the null set. For example, any non-trivial partition of the SBDD in Fig. 2.1(b) will increase the node count.

Algorithm 3.1 *(Bi-partition of an SBDD: Exact method)*

1. $min\_size \leftarrow \infty$.
2. *Enumerate a bi-partition* $\{F_1, F_2\}$ *of* $F = \{f_0, f_1, \ldots, f_{m-1}\}$*, where* $F_1 \cup F_2 = F$ *and* $F_1 \cap F_2 = \phi$*. If done, stop.*
3. $size \leftarrow size(SBDD, F_1) + size(SBDD, F_2)$.
4. *If* $(size < min\_siz)$*, then* $min\_siz \leftarrow size$.
5. *Go to 2.*

Although Algorithm 3.1 produces an exact minimum solution, it requires $T = 2^{m-1}$ minimizations of pairs of SBDDs, since T is the number of bi-partitions on F. So, this method is only practical for functions with small n and m. The following is a heuristic algorithm that can be used for functions with large BDDs.

Algorithm 3.2 *(Bi-partition of an SBDD: Heuristic method)*

1. *Simplify the SBDD for* F *by using the heuristic method [15]. Let* π_t *be an ordering of the input variables that simplifies the SBDD for* F*.*
2. *Simplify the BDD for each component function* f_i $(i = 0, 1, \ldots, m-1)$ *by using the method of [15]. Let* $r_i = \dfrac{size(BDD, f_i)}{size(BDD, f_i, \pi_t)}$.
3. $r_{av} = \dfrac{1}{m}\sum_{i=0}^{m-1} r_i$, $F_1 \leftarrow \phi$, *and* $F_2 \leftarrow \phi$.
   For each f_i
   if $(r_i < r_{av})$ *then*
   $F_1 \leftarrow F_1 \cup \{f_i\}$
   else
   $F_2 \leftarrow F_2 \cup \{f_i\}$.
4. $e_1 \leftarrow size(SBDD, F_1) + size(SBDD, F_2)$.
5. *Let* f_h *be a function that has the minimal* r_i *in* F_2*.*
6. $e_2 \leftarrow size(SBDD, F_1 \cup \{f_h\}) + size(SBDD, F_2 - \{f_h\})$.
   If $(e_1 > e_2)$ *then*
   $e_1 \leftarrow e_2$
   $F_1 \leftarrow F_1 \cup \{f_h\}$
   $F_2 \leftarrow F_2 - \{f_h\}$
   go to 5

7. *Let f_h be a function that has the maximal r_i in F_1.*

8. $e_2 \leftarrow size(SBDD, F_2 \cup \{f_h\}) + size(SBDD, F_1 - \{f_h\})$.
   If $(e_1 > e_2)$ *then*
   $e_1 \leftarrow e_2$
   $F_1 \leftarrow F_1 - \{f_h\}$
   $F_2 \leftarrow F_2 \cup \{f_h\}$
   go to 7
   else
   stop

4 Node Sharing

In this part, we consider the case where nodes are shared across SBDDs. Such applications exist for hardware and software synthesis.

Example 4.1 *In Fig. 2.2, the non-terminal nodes labeled x_6, and constant nodes can be combined yielding a BDD with 13 nodes. However, the resulting BDD is not an SBDD because of different orderings for input variables across middle level nodes.* *(End of Example)*

As shown in the above example, *node sharing* can produce a BDD that is not an SBDD. Thus, we cannot use existing BDD packages. Therefore, we perform this operation as a separate process.

Proposition 4.1 *Let v_0 and v_1 be nodes of two SBDDs: SBDD0 and SBDD1, respectively. If v_0 and v_1 represent the same logic function, then one can be removed.*

This is a sufficient condition to share a node between two SBDDs. The following example shows a case where two nodes representing different functions can be shared.

Example 4.2 *Consider the two functions:*

$$f_0 = (\bar{x}_1 x_2 \vee x_1 \bar{x}_2) x_3,$$
$$f_1 = x_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3.$$

Fig. 4.1 shows a pair of BDDs representing f_0 and f_1. Note that v_0 represents f_0, and v_1 represents the function $x_1 x_3$. Fig. 4.2 shows the BDD after node sharing. Note that v_0 is used instead of v_1 to represent f_1. Indeed, v_2 represents the function f_1 since,

$$\bar{x}_2 f_0 \vee x_2 \bar{x}_1 \bar{x}_3 = x_1 \bar{x}_2 x_3 \vee x_2 \bar{x}_1 \bar{x}_3 = f_1.$$

Note that the BDD in Fig. 4.2 is not an SBDD, since x_2 occurs twice in paths from the node for v_2 to the constant nodes. *(End of Example)*

In the interest of an efficient algorithm, we ignore the above case, which involves a complex analysis, and we only use Proposition 4.1 for node sharing between two SBDDs.

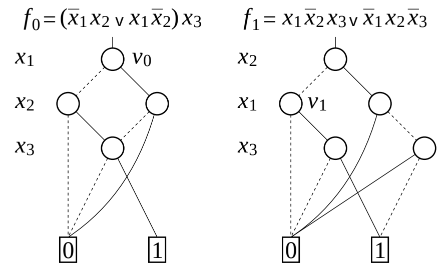


Figure 4.1: Pair of BDDs.

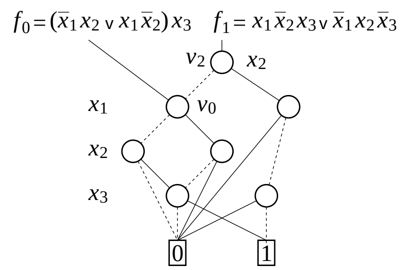


Figure 4.2: Sharing nodes that represent different functions.

Algorithm 4.1 *(Node sharing between two SBDDs)*

1. *For each node, assign two weights as follows:*

$$weight\_a = |\psi|$$
$$weight\_b = \sum_{i=1}^{n} 2^i \; depend(\psi, i),$$

   where ψ is the function represented by the node, $|\psi|$ is the number of 1's in its truth table, and

$$depend(\psi, i) = 1 \quad \text{if } \psi \text{ depends on } x_i$$
$$= 0 \quad \text{otherwise.}$$

2. *Select a node v_0 from SBDD0 and a node v_1 from SBDD1. For each pair of nodes (v_0, v_1) that have the same values for both weight_a and weight_b, check if they represent the same function. If so, remove the node v_i (not the subtree) that has fewer successor nodes that are shared by other parts of the SBDD. All edges leading to the eliminated node v_i, are redirected to the other node v_{1-i}.*

3. *Remove un-referenced nodes (e.g., a node may have no incoming edges because its predecessors were eliminated in Step 2).*

4. *Repeat Steps 2 and 3 until all pairs of nodes are considered.*

Table 5.1: Size of bi-partitioned SBDDs.

Name	In	Out	Monolithic	Bi-partitioned	Time (sec)
apex6	135	99	711	674	2010.4
apex7	49	37	302	253	2043.1
C499	41	32	27876	27862	472.4
C3540	50	22	34710	34543	369.8
C880	60	26	4166	4012	12.1
clip	9	5	111	110	0.1
ex5	8	63	342	339	3.0
exep	30	63	675	605	80.9
frg2	143	139	1117	1116	27.9
i10	257	224	24054	24031	19133.8
intb	15	7	608	607	1.6
jbp	36	57	467	444	2.9
rckl	32	7	198	188	0.6
signet	39	8	1472	1326	13.4
too_large	38	3	329	324	2.2
x2dn	82	56	243	234	27.6

IBM PC/AT compatible, PentiumIII 1GHz, Linux 2.2.16

5 Experimental Results

5.1 Performance of heuristic method

We implemented Algorithms 3.2 and 4.1, and partitioned BDDs for many benchmark functions. Table 5.1 lists the functions where the bi-partitioned SBDDs are smaller than monolithic SBDDs. In these cases, node sharing was not performed (i.e. Algorithm 3.2 only was applied). In the case of apex7, a reduction of 16% is achieved.

Table 5.2 lists the functions where the non-terminal nodes were reduced by node sharing (i.e. Algorithm 3.2 and 4.1 were used). Unfortunately, the number of nodes reduced by Algorithm 4.1 is not so large.

We applied Algorithm 3.2 to the results of Table 5.1, recursively. Table 5.3 lists the functions where the recursive application of Algorithm 3.2 reduced the total node count. In about one-half of the cases, recursive application resulted in significant node reduction over one application.

Table 5.2: Number of non-terminal nodes reduced by node sharing.

Name	In	Out	Node reduction
apex6	135	99	1
apex7	49	37	1
C3540	50	22	2
C499	41	32	1
C880	60	26	1
ex5	8	63	2
frg2	143	139	1
i10	257	224	41
intb	15	7	2
signet	39	8	2
x2dn	82	56	0

Table 5.3: Sizes of SBDDs after recursive application.

Name	In	Out	Monolithic	Bi-partitioned	
				Once	Recursive
apex6	135	99	711	674	671
apex7	49	37	302	253	244
C880	60	26	4166	4012	3975
exep	30	63	675	612	550
frg2	143	139	1117	1116	1114
i10	257	224	24054	24031	19981
intb	15	7	608	607	567
rckl	32	7	198	188	178
signet	39	8	1472	1326	1226

5.2 Comparison of heuristic and exact method

To see the quality of the bi-partitions obtained by Algorithm 3.2, we compared Algorithm 3.2 with an exhaustive method. The exhaustive method produced all the partitions of the outputs. Table 5.4 compares the sizes of BDDs for benchmark functions by using Algorithm 3.2 and the exhaustive method. *Alg3.2* denotes the size of bi-partitioned BDDs obtained by Algorithm 3.2; *Max* denotes the maximum size of bi-partitioned BDDs; *Min* denotes the minimum size of bi-partitioned BDDs; *Average* denotes the average size of bi-partitioned BDDs for all the partitions. Table 5.4(a) shows the case where the BDDs were optimized by an exact method [11]. Table 5.4(b) shows the case where the BDDs were optimized by a heuristic method

Table 5.4: Comparison of the heuristic method and exact method.

(a) When BDDs are minimized by an exact algorithm [11].

Name	In	Out	Monolithic	Bi-partitioned			
				Alg3.2	Max	Min	Average
alu2	10	8	70	75	87	69	80.2
clip	9	5	111	104	113	97	106.7
ex1010	10	10	1423	1493	1577	1486	1560.2
ex7	16	5	91	92	94	92	93.1
intb	15	7	608	595	636	560	601.2
max512	9	6	184	192	210	189	200.0
newtpla	15	5	54	55	61	55	57.5
t3	12	8	66	71	84	69	77.7
t4	12	8	44	51	53	46	49.5
x2	10	7	43	44	50	44	47.1

(b) When BDDs are minimized by a heuristic algorithm [15].

Name	In	Out	Monolithic	Bi-partitioned			
				Alg3.2	Max	Min	Average
i3	132	6	139	140	140	140	140.0
rckl	32	7	198	188	216	188	209.2
signet	39	8	1472	1326	1478	1316	1377.4
vg2	25	8	90	102	134	102	127.0
x1dn	27	6	139	140	174	140	164.0
x9dn	27	7	139	140	189	140	177.8

[15]. Table 5.4(a) shows that Algorithm 3.2 often produces solutions that are larger than minimum but smaller than the average. Unfortunately, bi-partitioned BDDs are often larger than monolithic ones. Table 5.4(b) shows that Algorithm 3.2 obtained the minimum solution in five out of six functions. Also in this case, bi-partitioned BDDs are often larger than monolithic ones.

In Tables 5.1–5.4, our SBDD do not use complemented edges.

6 Conclusions and Comments

In this paper, we showed a new method to represent a multiple-output function, partitioned SBDDs. Partitioned SBDDs represent a multiple-output function by a set of SBDDs, where each SBDD is optimized independently. The partitioned SBDD is more canonical than partitioned BDDs and free BDDs (FBDDs). We developed a heuristic bi-partition algorithm for SBDDs, and showed cases where the total numbers of nodes in bi-partitioned SBDDs are smaller than in monolithic SBDDs.

The advantages of partitioned SBDDs are

1) For each group of outputs, the orderings of the input variables are the same. So we can use well-developed tools for SBDDs [22].

2) When no node sharing among SBDDs is allowed, they can be evaluated in parallel for logic simulation [19].

In this paper, we consider applications as useful when the partitions have the property

$$size(SBDD, F_1) + size(SBDD, F_2) < size(SBDD, F).$$

However, for verification, the criteria for usefulness is different. Each SBDD is stored in computer memory one at a time, and the partition is used to reduce the peak memory size. In such a case, the bi-partitions are used to reduce $\max\{size(SBDD, F_1), size(SBDD, F_2)\}$.

Partitioned BDDs are considered in [13]. Their application is verification, in which case, extremely large BDDs are needed. Partitioning is a means of reducing BDD size so that each part fits into memory. Their experimental results show that the total sizes over all parts of a partitioned BDD are less than the size of the original un-partitioned BDD in 13 out of 20 benchmark functions. That is, partitioning results in a reduction in size in 65% of the benchmark functions.

Acknowledgments

This work was supported in part by a Grant in Aid for Scientific Research of the Ministry of Education, Culture, Sport, Science, and Technology of Japan.

References

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *ICCAD'95*, pp. 408-412, Oct. 1995.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E. M. Sentovich, and K. Suzuki, "Synthesis of software programs for embedded control applications," *IEEE Trans. CAD*, Vol. 18, No. 6, pp.834-849, June 1999.

[3] J. Bern, C. Meinel, and A. Slobodova, "Some heuristics for generating tree-like FBDD types," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems 15*, pp. 127-130, 1996.

[4] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic, *Field Programmable Gate Arrays*, Kluwer Academic Publishers, Boston, 1992.

[5] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Trans. on Comput.*, Vol. C-35, No. 8, pp. 677-691, Aug. 1986.

[6] G. Cabodi, P. Camurati, and S. Quer, "Improving the efficiency of BDD-based operators by means of partitioning," *IEEE Trans. on CAD*, Vol. 18, No. 5, pp. 545-556, May 1999.

[7] J. Gergov and C. Meinel, "Efficient analysis and manipulation of OBDDs can be extended to FBDDs," *IEEE Trans. on Comp.*, Vol. 43, No. 10, pp. 1197-1209, Oct. 1994.

[8] W. Gunther and R. Drechsler, "Minimization of free BDDs," *Proc. of Asia and South Pacific Design Automation Conference*, Jan. 1999, pp. 323-326.

[9] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASP-DAC'2000)*, Jan. 26-28, Yokohama, Japan.

[10] Y. Iguchi, T. Sasao, M. Matsuura, "Implementation of multiple-output functions using PQMDDs," *International Symposium on Multiple-Valued Logic*, pp. 199-205, May 2000.

[11] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of binary decision diagrams based on exchange of variables," *ICCAD-91*, pp. 472-475, Nov. 1991.

[12] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD'95*, pp. 402-407, Nov. 1995.

[13] A. Narajan, J. Jain, M. Fujita and A. Sangiovanni-Vincentelli, "Partitioned ROBDDs - A Compact, canonical and efficient manipulable representation of Boolean functions," *Proc. IEEE/ACM ICCAD'96*, San Jose, CA, USA pp. 547-554, Nov. 1996.

[14] A. Narajan, A. Isles, J. Jain, R. K. Brayton and A. Sangiovanni-Vincentelli, "Reachability analysis using partitioned-ROBDDs," *Proc. IEEE/ACM ICCAD'97*, San Jose, CA, USA, pp. 547-554, 1997.

[15] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 42-47, Santa Clara, CA, November 1993.

[16] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[17] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[18] T. Sasao and J. T. Butler, "A method to represent multiple-output switching functions by using multi-valued decision diagrams," *International Symposium on Multiple-Valued Logic*, pp. 248-254, Santiago de Compostela, Spain, May 29-31, 1996.

[19] T. Sasao, M. Matsuura, and Y. Iguchi, "Cascade realization of multiple-output function and its application to reconfigurable hardware," *International Workshop on Logic and Synthesis*, Lake Tahoe, June 2001, pp. 225-230.

[20] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *International Symposium on Multiple-Valued Logic*, Warsaw, Poland, May 2001, pp. 207-212.

[21] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," *ICCAD'97*, pp. 8-12, Nov. 1997.

[22] F. Somenzi, "CUDD: CU decision diagram package," Public software. University of Colorado, Boulder, CO, April 1997. http://vlsi.colorado.edu/ fabio/.

[23] M. Tachibana, "Heuristic algorithms for FBDD node minimization with application to pass-transistor-logic and DCVS synthesis," *Proc. of SASIMI Workshop*, pp. 96-101, Nov. 1996.

[24] S. Yang, *Logic synthesis and optimization benchmark user guide version 3.0*, MCNC, Jan. 1991.