Nonlinear Sifting of Decision Diagrams

Christoph Meinel and Rustam Mubarakzjanov
Dept. of Theoretical Computer Science and New Applications,
University of Trier, Trier, 54296, Germany

Abstract

We propose a new algorithm for the optimization of decision diagrams that combines the efficiency of sifting and the power of transformations. We show that the new algorithm is applicable to large examples, and that in many cases it leads to more compact diagrams when compared to linear sifting algorithm known to be more efficient than simple variable reordering.

Key words. OBDD, sifting, switching functions, circuits.

1 Introduction

Ordered binary decision diagrams (*OBDD*s) [2] are widely used in formal verification and provide an efficient graph-based structure to represent switching functions. The main optimization parameter of *OBDD*s is the underlying variable order that could change the size of the representation sometimes dramatically. Therefore variable reordering [8] is the most known and intensively investigated. But some important functions, e.g. Boolean multiplication [2] have an unmanageable large *OBDD*s for any variable order. A very promising extension of the basic *OBDD* structure is based on transformations of input variables ([1], [6], [7], [5]). After such a transformation, that should be bijective and easily constructible, a transformed function may have a more compact *OBDD* representation.

One of such transformations being studied intensively in last years is linear sifting based on transformations of the form

$$x_i \mapsto (x_i \equiv x_j). \tag{1}$$

That means the variable x_i is replaced by the equivalence function, i.e. the complement of the exclusive or, of x_i and x_j. Linear sifting is most suitable among the set of all previous known encoding transformation. The algorithm runs very fast, since the basic operation (very similar to variable reordering [8]) is a local operation. By this, it is possible to obtain much smaller *OBDD* sizes than with sifting alone.

We present in this paper a new type of transformations, nonlinear transformations. Our transformations are based on elementary operations of the form

$$x_i \mapsto (x_i \equiv (x_j x_k)), \tag{2}$$

i.e. the variable x_i is replaced by the equivalence function of x_i and the conjunction of variables x_j and x_k. Independently, a generalization of linear transformations has been proposed in [3], where some theoretical aspects of this generalization have been discussed.

2 Preliminaries

A (shared) *OBDD* on a set of variables $X = \{x_0, ..., x_{n-1}\}$ is a multi-rooted directed acyclic multi-graph which distinguished sink nodes labeled 1 and 0. The out-degree of each non-sink node is exactly 2. Each node is labeled by some variable x_i, $1 \le i \le n$, and is called an x_i-node. The two arcs outgoing from x_i-node are labeled 0 and 1. The label "δ" indicates that only inputs satisfying $x_i = \delta$ may follow this arc in the computation. An *OBDD* respects some ordering π of variables, i.e. the variables occurrences on each path from a root to a sink have to be consistent with π. An *OBDD* computes functions, a one for each root, in the obvious way: for each $\mathbf{x} \in \{0,1\}^n$, the label of a sink b defines the value of the function corresponding to a root a if and only if the path starting at a such that all labels along this path are consistent with $x = x_0 x_1 \ldots x_{n-1}$ leads to b. The *size* of an *OBDD* is the number of its nodes. It is well known that for each *OBDD* P, the unique (*reduced*) *OBDD* P' of minimal size exists so that P and P' compute the same function and have the same variable order. A device commonly employed, with which we shall concern ourselves, is a *complement edge*. A complement edge is an edge in the *OBDD* with an attribute set to indicate that the function pointed by the edge should be complemented to yield the desired function. To preserve canonicity, the edge labeled 1 cannot carry the complement attribute and *OBDD*s contain only 1-sink.

The size of *OBDD*s depends, sometimes, critically, on the variable order. A one of famous algorithms proposed to determine a good variable order is Rudell's *sifting* algorithm [8]. In this algorithm, each variable is considered in turn and is moved up and down in the order so as to take all positions successively. The variable is then returned to a position where the minimum size of the *OBDD* was recorded. The process continues then with another variable.

Let an *OBDD* P computing a function $f(x)$ read variables in the natural order $x_0, x_1, \ldots, x_{n-1}$. The exchange of variables x_i and x_j in the order could be considered as a computation of another function $f'(y) = f'(\rho(x)) = f(x)$ with

$$\rho(\ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots) =$$

$$(\ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots) =$$

$$(y_0, \ldots, y_{i-1}, y_i, y_{i+1}, \ldots, y_{j-1}, y_j, y_{j+1}, \ldots, y_{n-1}).$$

The *OBDD* P is transformed after this exchange into an *OBDD* P' computing a function $f'(y)$, again with the natural order of variables $y_0, y_1, \ldots, y_{n-1}$. The function ρ is an encoding transformation (or an automorphism) of input variables vectors, i.e. ρ is a bijective function: $\{0,1\}^n \to \{0,1\}^n$. It is evident that a proper automorphism may have a large impact of the *OBDD* size. Moreover there exist exponential many possible automorphisms therefore one should restrict him self with some set of transformations. An automorphism is completely characterized by the images of the *projection functions*, $x_0, x_1, \ldots, x_{n-1}$. Linear transformations are those automorphisms in which the images of the projection functions are obtained by composing a sequence of elementary transformations of the form (1). Even in the case of linear transformations it is impossible to consider all possible transformations in order to find an optimal one. To find a better transformation one uses local transformations recursively. *Linear sifting* proposed in [6] combines sifting algorithm with the transformation (1) in the following way: First, shift a variable x_i to a new position in the order, then perform the operation (1) to the variables x_i and x_j, where x_i and x_j are adjacent in the order. If the size of the *OBDD* increases after this transformation the same operation (1) rebuilds the *OBDD* to the previous one. Then the process continues to shift a variable x_i according to sifting algorithm.

It is easy to see ([6]) that if x_i immediately precedes x_j in the order and f is the function of x_i-node then we have:

$$f = x_i x_j f_{11} + x_i \overline{x_j} f_{10} + \overline{x_i} x_j f_{01} + \overline{x_i}\ \overline{x_j} f_{00}, \quad (3)$$

exchanging x_i and x_j and rearranging terms yields:

$$f' = x_i x_j f_{11} + x_i \overline{x_j} f_{01} + \overline{x_i} x_j f_{10} + \overline{x_i}\ \overline{x_j} f_{00}, \quad (4)$$

and the same process for the operation (1) yields:

$$f' = x_i x_j f_{11} + x_i \overline{x_j} f_{00} + \overline{x_i} x_j f_{01} + \overline{x_i}\ \overline{x_j} f_{10}. \quad (5)$$

In words, the effect of a swap and of an elementary linear transformation is to interchange f_{10} with f_{01} and f_{00} respectively. Note that one uses in the linear transformation (1) the equivalence function, rather than exclusive or, because *OBDD*s use complement arcs; hence we cannot swap f_{11} with one of $\{f_{00}, f_{01}, f_{10}\}$, all of which could be complemented.

3 Non-linear transformations

Mult-transformations are those automorphisms in which the images of the projection functions are obtained by composing a sequence of elementary transformations of the form (2). A transformation of the form (2) generalizes the exchange of two variables and the operation (1) if we allow that some of variables in (2) may be equal. We show that there exist functions for which mult-transformations can yield *OBDD* of polynomial size what is impossible for reordering variables.

We define the following function f on a set V of n variables, $n = k^2 + 2k$,

$$CON_k(x_1, \dots, x_k, y_1, \dots, y_k, z_{11}, \dots, z_{kk}) :=$$

$$\bigwedge_{i,j=1,1}^{k,k} (z_{ij} \equiv x_i y_j).$$

This function has an *OBDD*-representation with an *OBDD* P of the size $k^2 < n$ if one applies first a sequence of elementary mult-transformations $z_{ij} \mapsto (z_{ij} \equiv (x_i y_j))$, for all $i, j = 1, \dots, k$. Then P tests if all variables from $\{z_{ij} | i, j = 1, \dots, k\}$ are equal to 1.

Theorem 1 *An OBDD computing the function CON_k has at least 2^k nodes.*

Proof. Let *OBDD* R_k compute the function CON_k. Consider all paths from the root of R_k to a sink. The ordering π of variables in R_k introduces a partition of variables in two subsets V' and V'' such that any variable in V' is read before any variable from V'' and V' is the smallest subset containing at least one variable of each column or at least one variable of each row of the $(k \times k)$-matrix $[z_{ij}]$. Let there exist $\{z_{i_1 1}, \dots, z_{i_k k}\} = Z' \subseteq V'$, i.e. V' contains at least one variable of each column of the matrix. If z_{ij} is the last variable in π belonging to V' then z_{ij} is the unique variable of i-th row of the matrix belonging to V' or no variable of some row of the matrix belongs to V'. Thus there exists j such that for $Z'' = \{z_{j1}, \dots, z_{jk}\}$, it hold $|Z'' \cap V''| \geq (k-1)$.

We fix the following values of variables. All x's are set to 1. If $z_{jl} \in V' \cap Z''$ then we set $z_{tl} = 1$ for all $t, 1 \leq t \leq k$. We consider only such assignments of variables that $y_j = z_{i_j j} \in Z'$. Then different assignments to variables in $Z' - \{z_{i_j l}\}$ correspond to different nodes of R_k, because the function CON_k is equal 1 only if the values of the variables $Z'' \setminus V'$ are equal to the values of corresponding variables in $Z' - \{z_{i_j l}\}$. There are 2^{k-1} possible assignments to variables in $Z' - \{z_{i_j l}\}$. It easy to see that therefore R_k has at least 2^k nodes.

For such an ordering π, that V' contains at least one variable in each row of the matrix, the proof is analogous: We can find a column of the matrix $[z_{ij}]$ that determines a set Z'' and set then all y's to 1 and x's equal to corresponding values of z's. ∎

Like in (3)-(5) one can see that an elementary mult-transformation is local if x_i immediately precedes x_j in the order and x_j immediately precedes x_k. Then for the function f of x_i-node, we have:

$$f = x_i x_j x_k f_{111} + x_i x_j \overline{x_k} f_{110} + x_i \overline{x_j} f_{10} + \overline{x_i} x_j x_k f_{011} + \overline{x_i} x_j \overline{x_k} f_{010} + \overline{x_i}\ \overline{x_j} f_{00}, \quad (6)$$

the operation (2) and rearranging terms yields:

$$f = x_i x_j x_k f_{111} + x_i x_j \overline{x_k} f_{010} + x_i \overline{x_j} f_{00} + \overline{x_i} x_j x_k f_{011} + \overline{x_i} x_j \overline{x_k} f_{110} + \overline{x_i}\ \overline{x_j} f_{10}. \quad (7)$$

In words, the effect of an elementary mult-transformation is to interchange f_{10} and f_{110} with f_{00} and f_{010} respectively. As for linear transformations, we use in the mult-transformation (2) the equivalence function, rather than exclusive or. Our *mult-sifting* algorithm proceeds analogously to linear sifting algorithm as follows.

Each variable is considered in turn, and, as in sifting, it is moved up and down in the order. Let x_i be the chosen variable, and let x_j be the variable immediately following it in the order, and let x_k be the variable immediately following x_j in the order. One basic step of mult-sifting consists of the following three phases:

1. Variables x_i and x_j are swapped; let the size of the *OBDD* after the swap be s_1.
2. The elementary mult-transformation to three variables x_j, x_i, and x_k is applied.

Because after the swap x_i locates between x_j and x_k in the order, the mult-operation has the form: $x_j \mapsto (x_j \equiv (x_i x_k))$. Let the resulting *OBDD* have the size s_2.

3. If $s_1 \leq s_2$ then the same mult-transformation is applied. Since the operation (2) is its own inverse the mult-transformation will be undone in this case.

The algorithm keeps track of all transformations produced by moving x_i and returns to a position where the minimum size of the *OBDD* was recorded. Then the process is continued with another variable x_i.

Our algorithm for mult-sifting has to maintain the images of the projection functions (the transformed $x_0, \ldots, x_{n-1}$) to be able, e.g., to compare different circuits or switching functions. The most expedient way to do that is to maintain the projection functions in the *OBDD* package. Applications of the transformations to the *OBDD*s of the projection functions yield the *OBDD*s of their transforms, which can be used to build functions directly in the transformed space. We have implemented mult-sifting in the CUDD package [9]. The basic goal was to obtain a smaller size of *OBDD*. The *OBDD* of a projection function is represented by at most one additional node during a reordering in the original CUDD package and does not influence much on the size of the whole *OBDD*. Note that it is not the case for mult-sifting, since the operation (2) to the *OBDD* of the projection function x_i yields *OBDD* with three non-sink nodes.

Mult-sifting yields in average smaller *OBDD*s as by simple sifting. But linear sifting is often more efficient in the comparison with mult-sifting. To get better algorithm (that we shall call *multlin-sifting*) we have combined linear sifting with mult-sifting in the following way. After the swap of x_i and x_j, the linear transformation of the form $x_j \mapsto (x_j \equiv x_i)$ is applied. An elementary mult-transformation is applied then only if the linear transformation is unsuccessful (yields *OBDD* of the bigger size) and is undone.

4 Experiments

In this section we describe experimental results that have been carried out on an Intel Pentium III 500 MHz based Linux system. Our algorithms have been integrated in "nanotrav" of *CUDD* package [9] (version 2.3.0) and have been tested on the LGSynth'93 [4] library that contains 110 error free benchmarks. One uses two methods to build *OBDD*s. The first one is used while building a benchmark's *OBDD* and is applied whenever the size of built *OBDD* doubles. This method is called *automethod*. If the automethod is not given in a nanotrav option explicitly sifting is used as the automethod. The second method (a *final* method) is applied to the complete benchmark's *OBDD*. Tables 1 and 2 summarize results of our two series of experiments. The benchmarks for which compared transformations yield *OBDD*s of the same size are omitted: Table 1 and 2 contain 61 and 73 benchmarks respectively. In Table 1, sifting is considered as only a reordering method. In the second series of experiments, we take all reordering methods of *CUDD*-package except annealing, random, and genetic. The columns 2-3 of both Tables show results for reordering methods: Table 1 for sifting and Table 2 for the best among all the methods.

The columns corresponding to linear sifting present results when linear sifting has been used as the final method. The last columns of Tables show results when mult-sifting or multlin-sifting have been used as the final method. In the first series of experiments (Table 1), only the same methods can be used as an automethod, e.g. linear sifting for linear sifting columns. In the second series (Table 2), every reordering method and linear sifting have been considered as possible automethods for final linear sifting. For last columns, mult- and multlin-sifting have been considered as additional possible automethods. Tables show the best results. In Table 2, the best combination of methods for a benchmark is marked in the columns 5 and 9 respectively (the notations of methods are given below). If the best result is obtained with an automethod then used methods are denoted with the dash:

Table 1

circuit	sifting size	sifting time	linear sifting size	linear sifting ratio	linear sifting time	multlin- and mult-sifting size	multlin- and mult-sifting ratio	multlin- and mult-sifting time
squar5	34	0.02	37	1.09	0.01	37	1.00	0.02
apex4	913	0.25	983	1.08	0.39	981	1.00	0.31
z4ml	24	0.02	14	0.58	0.03	16	1.14	0.02
x2	37	0.02	37	1.00	0.01	41	1.11	0.02
5xp1	42	0.02	42	1.00	0.02	42	1.00	0.01
pm1	41	0.03	40	0.98	0.02	42	1.05	0.02
cordic	71	0.52	41	0.58	0.75	44	1.07	0.57
clip	112	0.04	61	0.54	0.04	78	1.28	0.02
my_adder	141	0.21	61	0.43	0.44	66	1.08	0.75
comp	125	0.16	70	0.56	0.27	71	1.01	0.23
example2	361	0.11	307	0.85	0.20	313	1.02	0.26
C1355	38761	86.59	648	0.02	3.69	1323	2.04	11.25
C499	32303	84.82	565	0.02	4.69	1079	1.91	7.64
ex4p	528	0.35	596	1.13	0.42	522	0.88	0.52
table5	707	0.23	693	0.98	0.24	695	1.00	0.32
table3	751	0.22	782	1.04	0.22	751	0.96	0.26
apex3	936	1.04	931	0.99	2.30	932	1.00	5.07
apex1	1311	0.71	1294	0.99	0.88	1297	1.00	1.31
seq	1414	2.05	1364	0.96	2.26	1414	1.04	2.48
C1908	8628	9.35	1597	0.19	3.65	1613	1.01	5.37
C5315	2081	4.53	1658	0.80	12.85	1681	1.01	6.21
des	3032	3.51	2095	0.69	7.36	2136	1.02	10.15
C2670	3673	11.61	2495	0.68	16.47	3401	1.36	16.58
pair	4088	3.82	4087	1.00	5.03	3965	0.97	10.57
C3540	171930	270.65	42895	0.25	238.81	42101	0.98	211.86
rd53	17	0.02	15	0.88	0.01	15	1.00	0.02
cm85a	36	0.01	22	0.61	0.02	22	1.00	0.03
rd73	31	0.02	28	0.90	0.02	28	1.00	0.02
sao2	84	0.03	85	1.01	0.04	83	0.98	0.03
lal	93	0.04	89	0.96	0.03	89	1.00	0.06
ttt2	121	0.04	120	0.99	0.06	120	1.00	0.06
vg2	244	0.08	193	0.79	0.11	193	1.00	0.16
i2	452	2.59	334	0.74	4.97	334	1.00	6.13
duke2	364	0.07	373	1.02	0.09	364	0.98	0.11
x3	608	0.44	603	0.99	0.59	603	1.00	1.21
k2	1266	0.53	1265	1.00	0.69	1265	1.00	1.14
i8	1282	1.63	1282	1.00	1.93	1280	1.00	2.15
b1	7	0.00	6	0.86	0.00	5	0.83	0.01
cm82a	16	0.01	14	0.88	0.01	12	0.86	0.01
cu	32	0.02	34	1.06	0.03	32	0.94	0.03
rd84	42	0.03	39	0.93	0.03	38	0.97	0.04
f51m	48	0.02	50	1.04	0.01	31	0.62	0.02
frg1	107	0.20	107	1.00	0.24	88	0.82	0.56
pcler8	89	0.04	90	1.01	0.06	88	0.98	0.06
term1	94	0.15	93	0.99	0.17	92	0.99	0.20
bw	104	0.01	106	1.02	0.02	101	0.95	0.02
alu2	162	0.02	165	1.02	0.03	159	0.96	0.07
i6	217	0.10	218	1.00	0.16	216	0.99	0.20
too_large	487	2.61	500	1.03	2.76	379	0.76	8.27
vda	511	0.08	511	1.00	0.10	509	1.00	0.20
apex2	590	2.59	600	1.02	6.35	566	0.94	13.23
misex3	601	1.25	601	1.00	1.29	582	0.97	1.27
dalu	1100	3.88	659	0.60	3.67	631	0.96	6.62
apex6	657	0.45	655	1.00	0.60	648	0.99	1.13
alu4	727	0.54	742	1.02	0.58	664	0.89	1.71
frg2	1051	1.00	1025	0.98	1.48	1012	0.99	3.19
apex5	1289	1.16	1196	0.93	3.23	1185	0.99	4.69
C880	5505	4.52	5512	1.00	6.07	5423	0.98	15.62
rot	8439	4.55	8239	0.98	6.64	7223	0.88	15.66
C7552	16395	59.44	9011	0.55	65.75	8018	0.89	187.77
i10	28977	28.82	27688	0.96	84.28	24966	0.90	57.79
Totals	343847	597.85	125621	0.37	493.15	121663	0.97	621.30

Table 2

	reordering		linear sifting				multlin- and mult-sifting			
circuit	size	time	size	methods	ratio	time	size	methods	ratio	time
cm163a	26	0.02	27	g-l	1.04	0.02	27	g-m	1.00	0.01
cmb	28	0.01	36	g-l	1.29	0.02	36	csy-cm	1.00	0.02
cm162a	30	0.03	31	l	1.03	0.02	31	g-m	1.00	0.02
x2	31	0.04	36	cw3-cl	1.16	0.02	33	cw2-cml	0.92	0.05
squar5	33	0.02	33	cw2-cl	1.00	0.01	37	csi-ml	1.12	0.00
cc	46	0.04	49	l	1.07	0.02	49	g-m	1.00	0.02
sao2	81	0.03	81	w2-cl	1.00	0.03	82	sy-cm	1.01	0.04
pcler8	86	0.08	87	w2-cl	1.01	0.12	88	m	1.01	0.05
b9	97	0.08	98	l-cl	1.01	0.11	98	cm	1.00	0.12
i4	211	2.11	228	w2-cl	1.08	7.09	228	cw2-cm	1.00	7.00
ex5p	242	0.12	296	w3-l	1.22	0.10	292	m-cml	0.99	0.15
too_large	302	2.87	337	cw2-cl	1.12	1.41	329	g-cm	0.98	3.55
vda	484	0.20	488	cl	1.01	0.29	491	cg-cm	1.01	0.39
x3	498	2.86	528	csy-cl	1.06	1.92	513	cml-cml	0.97	4.72
i9	908	0.64	912	l	1.00	0.50	912	sy-m	1.00	0.47
apex5	1053	2.06	1077	g-cl	1.02	3.78	1083	cw2-cm	1.01	1.77
z4ml	17	0.01	11	cw4-cl	0.65	0.01	15	cm-cml	1.36	0.00
rd73	31	0.01	23	sy-cl	0.74	0.02	26	csy-cml	1.13	0.02
clip	75	0.04	61	l	0.81	0.03	72	cml	1.18	0.06
comp	95	0.17	66	cw2-l	0.69	0.36	67	w2-cml	1.02	0.82
alu2	158	0.06	142	cl	0.90	0.06	148	cw2-cml	1.04	0.11
duke2	346	0.10	344	cl	0.99	0.23	346	cg-cm	1.01	0.23
x1	434	1.44	456	cg-cl	1.05	1.57	456	cw3-cm	1.00	1.82
ex4p	461	0.63	474	w3-cl	1.03	0.61	462	w3-cml	0.97	0.86
misex3	521	1.30	519	g-cl	1.00	1.54	521	w2-cm	1.00	0.64
alu4	627	0.64	513	w2-cl	0.82	0.41	528	cw2-cml	1.03	0.71
table5	704	0.29	663	g-cl	0.94	0.43	665	g-cml	1.00	0.69
apex4	889	0.30	887	g-cl	1.00	0.44	889	cw4-cm	1.00	1.17
seq	1355	2.42	1135	cw4-cl	0.84	3.37	1147	w4-cml	1.01	5.31
rot	4353	11.22	2808	cg-cl	0.65	20.17	2845	cg-cml	1.01	24.13
C1908	8613	11.15	794	cw4-cl	0.09	28.46	844	g-cml	1.06	13.52
C499	26408	95.29	486	l-cl	0.02	5.51	496	cl-cml	1.02	13.94
C1355	36772	138.44	471	l-cl	0.01	5.15	507	cml-cml	1.08	33.86
b1	7	0.00	5	g-cl	0.71	0.00	5	g-cml	1.00	0.00
rd53	17	0.01	15	csy-cl	0.88	0.00	15	g-ml	1.00	0.01
t481	21	0.18	21	w4-l	1.00	0.20	21	w4-m	1.00	0.20
cm85a	28	0.03	22	g-l	0.79	0.01	22	g-ml	1.00	0.02
rd84	42	0.03	38	cl	0.90	0.04	38	w3-ml	1.00	0.03
cordic	43	0.51	32	l-cl	0.74	0.77	32	w4-cml	1.00	0.55
lal	67	0.11	63	g-cl	0.94	0.11	63	g-cml	1.00	0.16
frg1	76	0.29	74	cg-cl	0.97	0.81	74	cg-cm	1.00	0.90
my_adder	82	0.26	50	g-l	0.61	0.24	50	g-ml	1.00	0.28
ttt2	107	0.07	106	g-cl	0.99	0.12	106	w2-cm	1.00	0.08
i2	206	6.84	205	csy-cl	1.00	19.02	205	cw2-cm	1.00	13.48
example2	265	0.37	262	cg-cl	0.99	0.50	262	cml	1.00	0.86
misex3c	543	0.21	538	cw2-cl	0.99	0.22	538	w2-cml	1.00	0.33
frg2	895	3.44	871	g-cl	0.97	3.95	871	g-cml	1.00	4.80
apex3	931	1.89	836	w3-cl	0.90	2.17	836	w3-cml	1.00	4.06
k2	1255	1.31	1234	w2-cl	0.98	1.10	1234	w2-cml	1.00	2.25
C432	1500	0.77	1368	cw2-cl	0.91	5.17	1368	cw2-cm	1.00	6.43
cm82a	12	0.00	14	cl	1.17	0.00	12	ml	0.86	0.01
f51m	39	0.02	47	cg-cml	1.21	0.02	27	g-cml	0.57	0.03
5xp1	42	0.01	42	w4-l	1.00	0.01	40	w2-cm	0.95	0.02
count	81	0.12	82	cg-cl	1.01	0.15	81	cm	0.99	0.23
term1	92	0.21	74	w3-cl	0.80	0.27	73	w3-cml	0.99	0.37
bw	101	0.02	100	cw4-cl	0.99	0.02	99	cm	0.99	0.03
apex7	292	0.30	216	g-cl	0.74	0.48	215	g-cml	1.00	0.65
x4	349	0.48	350	cl	1.00	0.75	349	sy-cm	1.00	0.79
apex2	494	3.09	401	cg-cl	0.81	13.46	338	cg-cm	0.84	13.71
apex6	520	1.84	511	csy-cl	0.98	2.52	498	csy-cml	0.97	3.41
table3	751	0.19	741	w2-cl	0.99	0.37	740	cw4-cm	1.00	0.72
cps	1014	0.55	976	g-cl	0.96	0.88	974	g-cml	1.00	1.68
dalu	1067	4.29	596	cg-cl	0.56	11.08	586	cml-cml	0.98	24.98
apex1	1247	1.47	1245	g-cl	1.00	2.49	1240	w2-cml	1.00	3.94
i8	1277	2.20	1277	cl	1.00	3.44	1274	w4-cm	1.00	3.36
C5315	1989	8.73	1491	g-cl	0.75	10.83	1397	g-cml	0.94	16.88
des	2999	5.57	2076	cl	0.69	10.87	2073	cml	1.00	17.03
C2670	3417	13.78	1029	cl-cl	0.30	83.27	991	cl-cml	0.96	90.45
pair	3603	8.62	2979	g-cl	0.83	15.91	2955	cm-cml	0.99	49.59
C880	4082	6.05	4086	cl	1.00	9.83	4035	cml	0.99	15.72
C7552	12239	95.01	4468	cl-cl	0.37	268.23	3499	cml-cml	0.78	993.80
i10	28165	94.93	22751	cg-cl	0.81	414.33	21788	cml	0.96	269.37
C3540	36177	394.76	23941	cg-cl	0.66	133.36	23789	cg-cml	0.99	172.12
Totals	192149	933.28	89430		0.47	1100.83	87176		0.97	1829.60

the automethod is written before dash, the final method is written after dash. We denote the used methods as following: "si" for sifting, "g" for group sifting, "sy" for symmetric sifting, "wi" for exchanges of variables with window equal to i, $i \in \{2,3,4\}$, "l" for linear sifting, "m" for mult-sifting, and "ml" for multlin-sifting. Moreover, each method in the second series is used in two variants, i.e. in its converging version too, that means until no further improvement is achieved. In the later case, the letter "c" is written at the beginning of the method notation.

The row-wise structure of Tables is the same. They have 4 subtables. The first rows are those where just a reordering yields the *OBDD*s of the minimal size. The second subtable contains benchmarks where linear sifting was better than multlin-sifting (but sometimes worse than mult-sifting in Table 1). Next benchmarks have the *OBDD*s of the same size for both types of transformations. In the last part (24 and 23 circuits, resp.), the best results are obtained by applying mult-transformations. In the "ratio" columns for linear sifting and for mult-transformations, ratio of the sizes of *OBDD*s in comparison to reordering and to linear sifting respectively is computed.

We left all the cases in our suite because we insisted on using the entire validation test suite of *CUDD*, so that no bias in favor or against mult-sifting were introduced by the selection process. We conclude this section with the following observation: Neither sifting, nor linear sifting, nor mult-sifting compute the optimal *OBDD*-representation of a function. The applying of a method while building *OBDD* does not guarantee that the obtained *OBDD* will be smaller. In fact, the representation of C880 in the first series of experiments is smallest after simple mult-sifting. If one applies to C880 multlin-sifting or mult-sifting while building *OBDD*s, the size of resulting *OBDD*s is up to 5 times bigger.

Acknowledgments

The authors would like to thank Christian Stangier for his helpful suggestions during the implementation of methods in *CUDD* package.

References

[1] J. Bern, Ch. Meinel, and A. Slobodova, "OBDD-based Boolean manipulation in CAD beyond current limits", in *Proc. 32nd ACM/IEEE Design Automaton Conference*, San Francisco, CA, 1995: 408–413.

[2] R. E. Bryant, "On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication", *IEEE Trans. on Comput.*, 40: 205–213, 1991.

[3] P. Kerntopf, "Nonlinear transformation of decision diagrams", in *Proc. 10th IEEE/ACC Int. Workshop on Logic Synthesis*, Granlibakken, CA, 2001: 173–178.

[4] *LGSynth93 Benchmarks,* http://www.cbl.ncsu.edu/CBL_Docs/lgs93.html

[5] Ch. Meinel and T. Theobald, *Algorithms and Data Structures in VLSI Design,* Springer, 1998.

[6] Ch. Meinel, F. Somenzi, and T. Theobald, "Linear sifting of decision diagrams and its application in synthesis", *IEEE Trans. on Computer-Aided Design*, 19(5): 521–533, May 2000.

[7] Ch. Meinel and T. Theobald, "Local encoding transformations for optimizing OBDD-representations of finite state machines", *Formal Methods in System Design*, 18: 285–301, 2001.

[8] R. Rudell, "Dynamic variable reordering for ordered binary diagrams", in *Proc. IEEE Int. Conf. Computer-Aided Design*, Santa Clara, CA, 1993: 42–47.

[9] F. Somenzi, *CUDD: CU Decision Diagram Package*, ftp://vlsi.colorado.edu/pub/