Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams*

Shipra Panda Fabio Somenzi
Dept. of Electrical and Computer Engineering
University of Colorado at Boulder

Bernard F. Plessier
Motorola Inc.
Austin, TX

Abstract

Knowing that some variables are symmetric in a function has numerous applications; in particular, it can help produce better variable orders for Binary Decision Diagrams (BDDs) and related data structures (e.g., Algebraic Decision Diagrams). It has been conjectured that there always exists an optimum order for a BDD wherein symmetric variables are contiguous. We propose a new algorithm for the detection of symmetries, based on dynamic reordering, and we study its interaction with the reordering algorithm itself. We show that combining sifting with an efficient symmetry check for contiguous variables results in the fastest symmetry detection algorithm reported to date and produces better variable orders for many BDDs. The overhead on the sifting algorithm is negligible.

1 Introduction

Interest in symmetric boolean functions has been keen since the early days of logic design [15]. In the presence of symmetric variables, several design problems simplify considerably. For instance, in recent times, attention has been paid to the comparison of functions with unknown input correspondence [4]. The comparison can be carried out more efficiently if some variables are symmetric. Another problem that benefits from the knowledge of symmetric variables is the ordering of variables for Binary Decision Diagrams. It is well-known that the relative order of symmetric variables is immaterial. In addition, it has been empirically observed in [8] that symmetric variables are adjacent in optimum orders for BDDs without complement arcs.

In this paper we take a synergistic approach to symmetry detection and variable ordering. We present a new algorithm for symmetry detection that outperforms previously published methods [12, 16] in terms of speed and capacity. Our algorithm is based on dynamic reordering of variables; specifically, it is based on sifting [14]. We show that symmetry detection, when combined with the sifting algorithm, produces higher-quality variable orders.

The symmetry detection method in [16], based on spectral analysis, differs substantially from ours; however, our work has a few ideas in common with [12]. (Theorem 3 in Section 4 is the criterion for neighboring variables found in [12].) However, our method is not based on a series of filters of increasing cost, but on a single algorithm that identifies symmetric variables and groups them together in the order to produce a better BDD. In addition, it deals with multiple output functions.

Although we present our algorithm for BDDs [2] and ADDs [1], it is also applicable, with minor modifications, to Edge-Valued BDDs [9] and Zero-Suppressed BDDs [11].

2 Preliminaries

A set $\{b_1, \ldots, b_p\}$ of n-variable boolean functions is orthonormal if $\sum_{i=1}^{p} b_i = 1$ and $b_i \cdot b_j = 0$, $i \neq j$. Given an orthonormal set $\{b_1, \ldots, b_p\}$, we can expand a function f with respect to it in the form $f = \sum_{i=1}^{p} f_i \cdot b_i$. When the b_i's are product terms, the f_i's are the familiar *cofactors* or *residues* of f with respect to the product terms.

Binary Decision Diagrams (BDDs) [2] are an efficient data structure for the representation of logic functions. A BDD representing a set of functions $\{f_1, \ldots, f_n\}$ is a directed acyclic graph (DAG) with n roots—one for each function—and two leaves. One leaf represents the constant 1 and the other represents the constant 0. An internal node N of the DAG is labeled with a variable v and has two children, T and E. The function represented by N, denoted by f_N is given by $f_N = v \cdot f_T + v' \cdot f_E$. It is customary to impose the restriction that the variables be ordered along all paths in the DAG and that no isomorphic subgraphs exist. Under these restrictions, BDDs provide a canonical representation of logic functions. It is also customary to attach a complementation attribute to arcs in the DAG. Proper use of complementation attributes insures that complementary functions are represented by the same DAG and that canonicity is preserved.

During the typical execution of a BDD-based program, nodes are created and disposed of. An efficient scheme to address the ensuing memory management problem uses a *reference count* for each node. The reference count of a node is the number of arcs in the DAG pointing to it. In particular, the arcs from the sources of the DAG are the *external* references, while the other arcs are the *internal* references. The reference counts play an important role in our symmetry detection algorithm. More details on the efficient implementation of a BDD package that is quite similar to ours can be found in [2].

The success of BDDs in solving seemingly intractable problems has motivated researchers to consider variants of the basic data structure that support a wider variety of applications [1, 9] or that are very efficient for some classes of problems [11]. Algebraic Decision Diagrams (ADDs), for instance, extend BDDs by allowing an arbitrary number of leaves. The leaves may store, for instance, real numbers. ADDs are formally defined as boolean functions over a boolean algebra whose carrier is larger than the number of leaves. Thanks to this definition, all theorems of boolean algebra (in particular the results of Section 3) apply to ADDs. Though our presentation is in terms of the more familiar BDDs, our methods have been implemented also for ADDs. We shall comment on Edge-Valued BDDs [9] and Zero-Suppressed BDDs [11] as the opportunity arises.

Finding a good variable order for a BDD is a non trivial problem which has attracted lively interest in the last few years. Many heuristic algorithms have been devised for the problem (e.g., [10]). Most heuristics derive an order by inspection of the circuit for which BDDs are to be built. This approach is obviously of limited usefulness when there is no circuit to start with. Furthermore, even the best heuristics may occasionally fail. For these reasons *reordering* algorithms have been developed [6, 7, 3, 13, 5].

Reordering algorithms are based on successive improvements of an existing order according to some local search

*This work was supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 93-DJ-206.

strategy. Of particular interest to us is the sifting algorithm [14], which can be implemented very efficiently. Sifting reorders variables by considering each of them in turn. A variable is moved up and down in the order by a series of swaps of adjacent variables. The best position is recorded, and at the end the variable is returned to that position.

The effectiveness of sifting depends on the efficiency with which adjacent variables can be swapped. Rudell has shown how this can be done in time that only depends on the number of nodes labeled by the two variables. The key device is a dictionary of the nodes—called the *unique table*—with a sub-table for each variable. Our implementation of sifting is similar to the one described in [14].

3 Some Results on Symmetric Variables

We study symmetric functions as originally defined by Shannon [15]. That is, we consider also the case of symmetry with complementation.

Definition 1 *A boolean function $f(x_1, \ldots, x_n)$ is* symmetric *in x_i and x_j (x_j') if the interchange of x_i and x_j (x_j') leaves the function identically the same.*

A multiple-output function is symmetric in x_i and x_j if and only if all outputs are symmetric in x_i and x_j.

Lemma 1 *A boolean function $f(x_1, \ldots, x_n)$ is symmetric in x_i and x_j if and only if $f_{x_i x_j'} = f_{x_i' x_j}$; f is symmetric in x_i and x_j' if and only if $f_{x_i x_j} = f_{x_i' x_j'}$.*

It is easily seen that interchanging symmetric variables in the order does not change the size of the BDD of f. On the other hand, if one considers an extended definition of symmetry, such that x_i and x_j are symmetric also if either $f_{x_i x_j'} = f'_{x_i' x_j}$ or $f_{x_i x_j} = f'_{x_i' x_j'}$ [16], then the invariance of the size of a BDD when x_i and x_j are interchanged is not guaranteed. Therefore, in this paper we follow Definition 1. Notice however, that our algorithm can be easily augmented to detect symmetries of the extended type.

Theorem 1 *If a boolean function f is symmetric in x_i and x_j, it depends on x_i if and only if it depends on x_j.*

Lemma 2 *If f and g are symmetric in x_i and x_j, then $f + g$, $f \cdot g$, and f' are also symmetric in x_i and x_j.*

Lemma 3 *If a boolean function f is symmetric in variables x_i and x_j, then its cofactors with respect to variables other than x_i and x_j are symmetric in x_i and x_j.*

Lemma 4 *If the cofactors of a boolean function f with respect to an orthonormal basis that does not depend on either x_i or x_j are all symmetric in x_i and x_j, then f is symmetric in x_i and x_j.*

In particular, we are interested in orthonormal bases composed of all the cubes formed with the variables that precede x_i and x_j in the order.

Theorem 2 *Let f be a boolean function symmetric in variables x_i and x_j. Let π be a variable order in which x_i precedes x_j. Let F be the BDD for f under order π. Then there are no arcs in F entering a node labeled x_j and coming from a node labeled by a variable preceding x_i in π. Furthermore, there are no nodes labeled x_i such that both their children are either internal nodes labeled by variables that come after x_j in π, or constant nodes.*

The properties studied so far only depend on the support of a function and its cofactors. Therefore, they apply regardless of the use of complement arcs.

4 Sifting-Based Symmetry Check

Checking for symmetry of distant variables (distant in the order) may be expensive, because it may involve the construction of the BDDs for the cofactors $f_{x_i x_j'}$ and $f_{x_i' x_j}$. However, checking for symmetry of adjacent variables is easy.

Theorem 3 *Let f, π, and F be as in Theorem 2. Let x_i and x_j be adjacent variables. Then x_i and x_j are symmetric in f if and only if: 1) For all nodes labeled x_i, the condition $g_{x_i x_j'} = g_{x_i' x_j}$ is verified, where g is the function rooted at the node labeled x_i. 2) All arcs into nodes labeled x_j come from nodes labeled x_i.*

Checking for negative symmetry is similar and only requires replacing $f_{x_i x_j}$ for $f_{x_i x_j'}$ and $f_{x_i' x_j'}$ for $f_{x_i' x_j}$. For both conditions of Theorem 3, the organization of the unique table makes checking efficient. The nodes labeled x_i and x_j can be accessed by simply scanning the appropriate subtables. The second condition, in particular, can be checked by computing the sum of the reference counts of the nodes labeled x_j and comparing it to the number of arcs out of nodes labeled x_i and into nodes labeled x_j. If the former is larger, then there are references, either external or internal, that do not come from nodes labeled x_i.

For Edge-Valued BDDs, it is also necessary to check that the value on the *then* arc of the node labeled x_i equals the value on the *then* arc of the *else* child of that node.

The criterion for symmetry in x_i and x_j, and the one for symmetry in x_i and x_j' are slightly different in Zero-Suppressed BDDs (ZDDs). In a ZDD, a node is suppressed if its *then* child is 0. A node with identical *then* and *else* children, on the other hand, is not suppressed.

The test for symmetry of a ZDD in x_i and x_j is exactly the same as the one for BDDs: For every node labeled x_i, the condition $f_{x_i x_j'} = f_{x_i' x_j}$ must hold; all edges into nodes labeled x_j must come from nodes labeled x_i. The first condition is obvious; for the second, observe that an arc coming from above x_i implies the presence of a suppressed node labeled x_i with $f_{x_i} = f_{x_i x_j'} = 0$. For symmetry, $f_{x_i' x_j}$ should be 0, but that would imply the suppression of the node labeled x_j.

For symmetry in x_i and x_j', the second condition of Theorem 3 is modified as follows: If a node labeled x_j has references from above x_i, its *else* child must be 0.

In the sifting algorithm, every variable that is moved up and down is at some point adjacent to any other variable. Hence, if we combine the test for adjacency with sifting, we can identify all symmetric pairs, without ever computing a cofactor, as long as each variable is sifted.

Once two variables are identified as symmetric, we "lock" them, so that their relative position never changes from that point on. This leads to an algorithm that sifts groups of variables of varying size. There are a few differences from the original sifting algorithm that derive from this characteristic. First of all, if we want to make sure that the locally optimum position of a group of variables is found, we may have to sift the group twice instead of once.

The requirement for a single variable is that it takes all positions in the order. This is obtained by initially sifting it to the closer extreme of the order, and by then sifting it all the way to the other extreme. If symmetries are considered, though, the variable that is sifted may collect other variables along the way. The information on the best position is therefore invalidated and one additional pass of sifting is needed. If the size of the group does not change during the first complete sifting, the second can be skipped.

The other observation is that sifting a group of m variables from one extreme of the order to the other requires

$m(n-m)$ swaps, where n is the total number of variables. This is clearly more expensive than sifting one variable only, but is roughly equivalent to sifting the m variables one at the time. In summary, the number of pairwise swaps does not increase appreciably when symmetric variables are clustered and moved as a group.

Group sifting may be desirable not only to keep symmetric variables together, but also, for instance, to impose proximity constraints on variables for which the analysis of the circuit suggests that they should be kept close.

5 Experimental Results

In this section we present experiments conducted on several circuits from the IWLS benchmark set and on some additional symmetric circuits. Table 1 summarizes the runs of different combinations of ordinary sifting and sifting combined with symmetry check. In all experiments, BDDs are built for the circuits while applying dynamic reordering. Once the BDDs for all primary outputs are built, the BDDs for the internal nodes of the circuit are freed and reordering is applied again.

The column headings give the combinations of methods. Specifically, *sift-sift* means ordinary sifting while building the BDDs, followed by ordinary sifting. *Sift-symm* means ordinary sifting while building the BDDs, followed by sifting with symmetry check. Similarly for the other sets of columns. *Cosift* indicates ordinary sifting to convergence; *cosymm* indicates sifting with symmetry check to convergence. Also displayed for each circuit are the number of inputs, the number of inputs that are symmetric to some other input, and the total number of symmetry groups containing more than one variable.

In all cases except for the adders and *dpath32*, the standard *misII* ordering [10] was used as initial order. For the other circuits, the orders were intentionally chosen so as to generate exponentially sized BDDs. The intent was to study the ability of the reordering algorithm to recover from bad initial orders. Sizes are given in BDD nodes and times are in seconds on a DECstation 5000/200 with 80 MB of memory.

We first compare the new sifting with symmetry check to ordinary sifting when they are used for the final reordering. The data in the tables shows that for examples that have no symmetry the results are the same, as expected. More interestingly, the times are quite similar, indicating that the overhead for symmetry check is very small. For the circuits in the remaining two groups, the data show that symmetry checking almost always produces better or equally good results. Obviously, the biggest wins come from circuits with lots of symmetry.

It should be noted that ordinary sifting in many cases puts symmetric variables close to each other. Therefore, the advantage of our symmetric sifting lies in being able to sift groups of variables at once, rather than in placing symmetric variables close to each other.

In ordinary sifting, when two symmetric variables are adjacent, it is unlikely that their position in the order will change, because pulling them apart is likely to increase the size of the BDD, even though one variable is going towards the "right" position. Group sifting does not have this problem, because it moves both variables at once.

The data for the case when sifting with symmetry is used while building the BDDs exhibits much larger variance than the other two sets of data. The reason is that a function is symmetric in all variables on which it does not depend. When the BDDs are being built, many symmetries of this type may occur. Those variables are then grouped together, even though they are not related in the final circuit. This sometimes provides good results, and sometimes it does not.

Finally, in the case of sifting to convergence, one can see that the advantages of symmetry checking are larger in terms of nodes. However, this comes at the expense of an increase in CPU time. The larger differences are due to different numbers of iterations to reach convergence.

Table 2 gives the times required to run the symmetry check once the BDDs are built. For our method, this corresponds to the time required by the one pass of sifting done after the BDDs are built for all primary outputs. We compare our times to those reported by [12, 16]; it should be kept in mind that our method considers all outputs simultaneously. It is quite clear, though, that our method is much faster, especially for large circuits.

6 Conclusions

We have presented an algorithm that combined the detection of symmetric variables with the dynamic reordering of the variables of a BDD. This method handles multiple-output functions and is the fastest method devised so far for large functions. It is also very memory-efficient. No additional data structures are required, except for a few bytes per variable (not per node) to keep track of the symmetry information. The size of the BDD may grow during sifting. However, in practice this has not been found to be a problem. In addition, a limit can be imposed of the allowable growth of the DAG. Even though in theory this may lead to missing some symmetries, in our experiments we have observed that very few symmetries escape detection, even when the maximum growth is limited to 20%.

We have shown that the detection of symmetric variables improves the effectiveness of the sifting algorithm: The BDDs produced are smaller and the overhead, when no symmetries are present, is negligible.

The fact that a function is symmetric in all the variables on which it does not depend may cause some problems when using symmetry detection while building the BDDs for a circuit. We are currently investigating ways of mitigating the problem, while retaining the advantages that symmetry detection affords in many cases.

Symmetry detection represents one example of structural analysis of a BDD that may help produce a better order. Given the sensitivity of dynamic reordering algorithms to the initial conditions, such methods are highly desirable and are the focus of our current research.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Proceedings of the International Conference on Computer-Aided Design*, pages 188–191, Santa Clara, CA, November 1993.

[2] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proceedings of the 27th Design Automation Conference*, pages 40–45, Orlando, FL, June 1990.

[3] N. Calazans, Q. Zhang, R. Jacobi, B. Yernaux, and A.-M. Trullemans. Advanced ordering and manipulation techniques for binary decision diagrams. In *Proceedings of the European Conference on Design Automation*, pages 452–457, Brussels, Belgium, March 1992.

[4] D. I. Cheng and M. Marek-Sadowska. Verifying equivalence of functions with unknown input correspondence. In *Proceedings of the European Conference on Design Automation*, pages 81–85, Paris, France, February 1993.

[5] E. Felt, G. York, R. K. Brayton, and A. Sangiovanni-Vincentelli. Dynamic variable reordering for BDD minimization. In *Proceedings of the European Design Automation Conference*, pages 130–135, Hamburg, Germany, September 1993.

[6] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proceedings of the European Conference on Design Automation*, pages 50–54, Amsterdam, February 1991.

name	in	sym-vars	sym-groups	sift-sift		sift-symm		symm-symm		sift-cosift		sift-cosymm	
				size	time	size	time	size	time	size	time	size	time
C1908	33	0	0	6072	112.54	6072	111.76	6062	108.37	6034	135.04	6034	135.53
C1355	41	0	0	29589	739.58	29589	740.02	29589	779.16	29589	809.31	29589	807.41
C432	36	0	0	1241	12.08	1241	11.89	1241	14.10	1241	12.69	1241	13.13
C3540	50	0	0	24561	573.02	24561	569.88	23850	412.60	23842	861.43	23842	862.59
C499	41	0	0	30113	384.11	30113	381.24	40629	561.15	26657	561.34	26657	553.77
des	256	0	0	3066	491.77	3066	492.40	2997	365.87	3066	491.83	3066	493.73
s9234.1	247	0	0	4384	335.81	4384	336.06	3584	254.06	3977	459.33	3977	465.54
s1423	91	0	0	2141	51.73	2141	51.45	2070	63.68	2133	61.48	2133	61.99
accpla	50	0	0	1764	34.18	1764	34.20	1781	33.77	1726	40.18	1726	40.72
mm9a	39	0	0	2042	18.20	2042	18.13	3095	25.08	2042	18.65	2042	18.24
mm9b	38	0	0	2022	27.11	2022	27.04	2022	27.78	2022	28.80	2022	28.64
mm30a	123	0	0	22189	1026.25	22189	1029.10	293148	6476.69	19211	1161.58	19211	1184.67
alupla	25	2	1	969	3.84	969	3.76	669	3.89	969	4.39	969	4.31
frg2	143	2	1	1643	109.00	1643	108.47	1292	72.94	1297	219.45	1229	199.23
apex6	135	2	1	627	31.07	622	31.08	623	33.72	615	49.62	610	50.55
dalu	75	2	1	766	28.11	766	28.25	766	24.90	766	30.80	766	30.88
seq	41	4	2	1496	62.90	1497	62.89	1497	60.58	1382	66.77	1382	69.11
vg2	25	4	2	190	3.69	190	3.71	190	3.55	188	4.11	188	4.22
C5315	178	4	2	2429	157.02	2429	159.91	2462	100.65	2416	209.16	2420	189.83
s15850.1	611	4	2	37539	5059.00	37539	5044.97	13518	2196.81	35341	7394.46	35275	9033.31
C880	60	6	3	4648	50.94	4648	50.62	5417	56.07	4202	66.49	4193	67.09
too_large	38	9	4	352	75.72	352	75.66	356	81.22	352	78.91	352	76.93
C2670	233	12	3	2464	228.83	2389	229.59	2440	235.67	2402	267.25	2182	352.04
i10	257	13	6	31981	1661.68	30855	1652.21	24024	527.43	31626	2316.37	30387	2654.41
cordic	23	17	5	89	0.47	89	0.48	89	0.51	89	0.58	62	0.82
s5378	199	23	7	2710	170.11	2702	169.97	2539	134.96	2567	223.40	2565	255.08
C7552	207	41	13	6365	423.96	6342	423.98	10036	751.92	6083	530.87	6022	641.55
t481	16	16	8	35	10.93	35	10.96	35	11.28	35	10.86	35	11.06
adder16	33	33	16	310	2.87	185	2.90	119	2.53	310	3.23	185	3.17
adder16r	33	33	16	377	2.26	113	2.28	113	2.67	377	2.61	113	2.49
adder32	65	65	32	1181	16.62	309	16.76	225	14.00	1181	17.96	309	17.46
adder32r	65	65	32	2509	27.61	225	27.06	225	11.87	2509	27.60	225	27.90
dpath32	93	93	31	4848	270.97	4362	268.78	6913	1043.50	4848	280.74	4362	276.27
i3	132	132	66	258	19.85	258	17.51	258	8.83	258	26.44	258	22.23

Table 1: Reordering Results.

name	Sifting-Based time	Möller *et al.* time	Tsai *et al.* time
C1908	8.054	1126.5	
C1355	59.504		
C432	0.704	23.2	
C3540	36.182		
C499	66.429		
des	28.174	52.5	432.24
s9234.1	34.568		
s1423	5.086		
accpla	1.988		
mm9a	1.168		
mm9b	1.672		
mm30a	54.625		
alupla	0.57	3.9	
frg2	8.445	5.3	1896.10
apex6	6.148	2.2	28.33
dalu	2.289	32.5	
seq	1.64	11.0	
vg2	0.23	0.7	
C5315	14.55	636.7	
s15850.1	599.891		
C880	7.039	7.7	
too_large	0.493	3.8	
C2670	21.475		
i10	177.731		
cordic	0.164	0.2	
s5378	18.663		
C7552	37.99		
t481	0.051		
adder16	0.36		
adder16r	0.422		
adder32	2.043		
adder32r	2.464		
dpath32	11.394		
i3	3.851	0.6	

Table 2: Symmetry Detection Times.

[7] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *Proceedings of the International Conference on Computer-Aided Design*, pages 472–475, Santa Clara, CA, November 1991.

[8] S.-W. Jeong, T.-S. Kim, and F. Somenzi. An efficient method for optimal BDD ordering computation. In *International Conference on VLSI and CAD (ICVC'93)*, Taejon, Korea, November 1993.

[9] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of the Design Automation Conference*, pages 608–613, Anaheim, CA, June 1992.

[10] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 6–9, Santa Clara, CA, November 1988.

[11] S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the Design Automation Conference*, pages 272–277, Dallas, TX, June 1993.

[12] D. Möller, J. Mohnke, and M. Weber. Detection of symmetry of boolean functions represented by ROBDDs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 680–684, Santa Clara, CA, November 1993.

[13] B. F. Plessier. *A General Framework for Verification of Sequential Circuits.* PhD thesis, University of Colorado at Boulder, Dept. of Electrical and Computer Engineering, 1993.

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

[15] C. E. Shannon. A symbolic analysis of relay and switching circuits. *AIEE Trans.*, 57:713–723, 1938.

[16] C. C. Tsai and M. Marek-Sadowska. Detecting symmetric variables in boolean functions using generalized Reed-Muller forms. In *Proceedings of the International Symposium on Circuits and Systems*, London, Britain, May 1994.