

Who Are the Variables in Your Neighborhood*

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

Abstract

Dynamic reordering techniques have had considerable success in reducing the impact of the initial variable order on the size of decision diagrams. Sifting, in particular, has emerged as a very good compromise between low CPU time requirements and high quality of results. Sifting, however, has the absolute position of a variable as the primary objective, and only considers the relative positions of groups of variables indirectly. In this paper we propose an extension to sifting that may move groups of variables simultaneously to produce better results. Variables are aggregated by checking whether they have a strong affinity to their neighbors. (Hence the title.) Our experiments show an average improvement in size of 11%. This improvement, coupled with the greater robustness of the algorithm, more than offsets the modest increase in CPU time that is sometimes incurred.

1 Introduction

With the growing popularity of Binary Decision Diagrams (BDDs) and their derivatives [1, 9, 11, 4], the interest in reliable methods to produce good variable orders has increased. Given the combinatorial nature of the problem, exact algorithms can only be applied to instances of small size. For most applications, heuristic methods are used to either derive an order by inspection of a logic circuit [6, 10], or to improve on a given order [7, 8, 14, 2, 13]. The heuristic improvement algorithms, also called *dynamic reordering* algorithms, are based on various local search schemes.

Sifting [14] has emerged so far as the most successful algorithm for *dynamic reordering* of variables. The neighborhood of a given order is defined in sifting as the set of orders that can be obtained by moving one variable to all possible positions, while keeping the other variables fixed. Although this strategy normally produces good results, a limitation of sifting is that it uses the absolute position of a variable as the primary objective, and only considers the relative positions of groups of variables indirectly.

Consider the case of a function containing two variables, a and b with a strong affinity to each other. When a is sifted, it is likely to end up close to b , because b “attracts” a strongly. However, the overall position assumed by a and b may be far from optimal, and when b is sifted, it is likely to return to its initial position, due to the attraction of a , which effectively acts like a rubber band. This observation is supported from our experiments on symmetric sifting [13], in which we found that regular sifting usually put symmetric variables together in the order, but the symmetric groups tended to be in sub-optimal positions, because sifting was unable to recognize that these variables should be sifted together. Symmetric sifting, on the other hand, was not hampered by the “rubber-band” effect, because it could move the symmetric variables simultaneously.

The advantage of symmetric sifting over conventional sifting is limited, because symmetric variables do not appear too often in practical circuits. We were therefore motivated to devise a more general algorithm, which could provide more consistent advantages over conventional sifting than symmetric sifting. The main challenge in developing a group sifting algorithm consisted in formulating a dependable criterion for the grouping of variables. Criteria of general applicability are more prone to aggregate variables that should not actually be kept together.

The rest of this paper describes our general group sifting algorithm, and the details of our aggregation criterion, which is based on letting sifting itself identify variables with strong attraction, and on applying a relaxation of the symmetry check that we call *extended symmetry* check.

2 Preliminaries

Binary Decision Diagrams (BDDs) [5, 3] are an efficient data structure for the representation of logic functions. Details on the implementation of a BDD package that is quite similar to ours can be found in [3].

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, 4] or that are very efficient for some classes of problems [11]. Though our presentation is in terms of the more familiar BDDs, our methods apply, with minor or no modifications, also to these other types of decision diagrams.

2.1 Variable Reordering

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 and exact algorithms have been devised for the variable ordering problem. 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 or produce inferior results. For these reasons *reordering* algorithms have been developed [7, 8].

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

*This work was supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 94-DJ-560.

2.2 Symmetric Variables

Symmetry is the basis of one of our aggregation criteria; hence we review some basic ideas on symmetry. We consider symmetric functions as originally defined by Shannon. That is, we consider also the case of symmetry with complementation.

Definition 1 A boolean function $f(x_1, \dots, x_n)$ is symmetric in x_i and x_j if the interchange of x_i and x_j leaves the function identically the same; f is symmetric in x_i and x'_j (alternatively, x'_i and x_j), if the interchange of x_i and x'_j (and x'_i and x_j) leaves the function identically the same.

It is easily seen that interchanging symmetric variables in the order does not change the size of the BDD of f . The fundamental properties for an efficient symmetry check on BDDs are as follows: 1) Let x_i and x_j be adjacent variables where x_i precedes x_j . For all nodes labeled x_i , the condition $g_{x_i x'_j} = g_{x'_i x_j}$ is verified (or $g_{x_i x_j} = g_{x'_i x'_j}$ for the case of negative symmetry), 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 . The organization of the unique table makes checking both conditions of Section 2.2 efficient. More details can be found in [13].

2.3 Adjacency and Symmetry

Symmetric variables tend to be adjacent in optimum orders for BDDs without complement arcs. Möller *et al.* have found this to be the case for most, but not all functions with up to five variables [12]. A simple counterexample is as follows: $\xi(x, y, z) = xy + yz' + y'z$ is a symmetric function for which no symmetric order is optimal. The optimal orders are indeed, $y < x < z$ and $z < x < y$, whereas ξ is symmetric in y and z . This particular counterexample works only for complement arcs. However, based on ξ we can build a family of parameterized functions such that there is a linear gap between optimal orders and the best symmetric orders for both cases with and without complement arcs. For instance: $\Xi_d(x_1, y_1, z_1, \dots, x_d, y_d, z_d) = \bigoplus_{i=1}^d \xi(x_i, y_i, z_i)$ forms such a BDD structure¹. In spite of the existence of functions exhibiting this linear gap, as a practical matter it is almost always a good idea to keep symmetric variables together. Furthermore, the group sifting algorithm we discuss in this paper detects the suboptimality of symmetric orders in cases like this one.

3 Group Sifting

Group sifting is an extension of sifting. It moves a group of variables at the time, instead of a single variable. As a special case, a group may contain a single variable.

We consider, in the general case, two types of groups: *Hard* groups are passed to the reordering procedure by the caller. Hard groups can specify constraints related to the correctness of the calling algorithm, or may carry suggestions on the structure of the ordering, derived from knowledge of the BDD-based application at hand.

Hard groups can be nested and can be *fixed*. The subgroups of a fixed group keep their relative positions throughout the reordering process. *Soft* groups are groups

created by the reordering algorithm, when it identifies variables that have strong affinity. These groups are short-lived, not nested, and do not survive between successive invocations of the reordering procedure.

One simple example of group sifting is symmetric sifting [13]. In the sifting algorithm, every variable that is moved up and down is at some point adjacent to any other variable. Hence, by combining the test for symmetry with sifting, we can identify all symmetric pairs, without ever computing a cofactor, if we guarantee that each variable is sifted. Once two variables are identified as symmetric, they are “locked” together, so that their relative position never changes from that point on. This effectively leads to an algorithm that sifts groups of variables of varying size. In the case of symmetric sifting, soft groups are only dissolved at the end of the sifting procedure.

In the sequel we present a new approach to group sifting, which is not restricted to aggregating symmetric variables. Hard groups are organized as a tree. Initially there are no soft groups. At the top level, the group reordering procedure visits this tree in post-order and sifts the children of each non-terminal non-fixed node. While reordering a set of individual variables (as opposed to a set of groups), the group sifting algorithm will try to create new soft groups.

3.1 Variable Aggregation

The key to successful group sifting is the use of effective aggregation criteria. Soft groups are created or merged when adjacent variables pass an aggregation check. There are several possible aggregation checks and several schemes for applying them. In this section we describe two aggregation criteria, while in the next section we discuss the reordering scheme.

3.1.1 Extended Symmetry

One variable aggregation check is based on the notion of *extended symmetry*. Section 2.2 shows that for two variables to be symmetric, all arcs into the layer corresponding to the variable with the higher index (the top of the BDD has index 0) should come from the layer corresponding to the other variable, and the condition $g_{x_i x'_j} = g_{x'_i x_j}$ must be true for positive symmetry, or $g_{x_i x_j} = g_{x'_i x'_j}$ must be true for negative symmetry. We extended this criterion in two ways. First we allow for cases of mixed positive and negative symmetry. Secondly, we allow for a fixed percentage of violations. For instance, we may allow for 10% of the nodes checked to fail both positive and negative symmetry checks. Because the condition is derived from the one of Section 2.2, we call it extended symmetry.

3.1.2 Variable Interaction

When reordering the variables of a multiple-output function, if two variables do not appear in the support of the same output, they are said to be *non-interacting*. Non-interacting variables should not be aggregated in general, because their proximity in an order is typically accidental. Variable interaction is used as a filter applied after some other aggregation criterion.

3.1.3 The Method of the Second Difference

Let $N(i)$ be the number of nodes labeled i . Let $n-1$ be the largest variable index appearing in the BDD. Let i be such that $0 < i < n-1$. We define: $S(i) = \frac{N(i+1)}{N(i)} - \frac{N(i)}{N(i-1)}$.

¹Detlef Sieling of the University of Dortmund has independently come up with similar examples exhibiting linear gaps.

The quantity $S(i)$ is related to the second difference of the sequence $N(i)$.

Because the behavior of $N(i)$ is more or less the same for all BDDs—they have a pear-like shape—the size information is not directly useful to improve the variable order. The same applies to the first difference of the series, because it also has a predictable behavior. However, the second difference starts to filter out the effects of the pear shape. In particular, we can measure the amount of recombination occurring between two variables when they are adjacent in a manner that is relatively independent of their position in the BDD. A negative $S(i)$ indicates a lot of recombination, and is therefore taken to mean that x_i and x_{i+1} should be kept together. The value of $S(i)$ is very easy to compute, because $N(i)$ is maintained by the BDD package.

3.2 Reordering Schemes

Sifting tries to achieve a global optimum by a sequence of local optima—orders in which one variable’s position is best, given the positions of the other variables. The second method we have implemented tries to take advantage of the observation that the locally optimum position of a variable depends on the absolute position in the order, as well as on the position of the variable relative to the other variables. If the globally optimal positions of a variable differ significantly from the local optima, chances are that other variables attract it to the “wrong” position. The Relative Absolute Position (RAP) method is therefore composed of two phases iterated for each variable. At each iteration, a variable that has not been sifted yet is chosen. This variable is then sifted up and down, while checking it against its neighbor for extended symmetry. If extended symmetry is found, a group is formed and the relative position of its variables does not change from that point on.

At the end, the variable (or the group) is returned to the position where the minimum size was encountered. In the process of returning to the best position, the newly formed group may be dissolved if the best size was achieved before the group was formed. This remedies the shortcoming of symmetric sifting for functions such as those discussed in Section 2.3. If extended symmetry has been found, the algorithm goes on to choose another variable to sift; otherwise it enters the second phase.

The position to which the variable is returned is assumed to depend, at least in part, on the presence of affine variables in adjacent positions. Therefore, the variable is checked against the two adjacent variables, to see if a group should be created. The aggregation test is the second difference check of Section 3.1.3, followed by the check for variable interaction of Section 3.1.2. Since we are not checking until after the variable has found its best position, we can use a more liberal aggregation test than in the first phase. If the test succeeds, a soft group, of size up to three, is formed, and goes through a new phase of sifting. This second phase may bring the soft group to a new position. Once this is done, the new group is dissolved.

Our method is better than iterating sifting until convergence: It normally produces smaller BDDs in less time. This agrees with our observation that a locally optimum position does not usually change much as a result of sifting the other variables. Instrumental to the success of our group sifting approach is the fact that extended symmetry can be checked quickly, and that it fails most of the times.

4 Experimental Results

In this section we present experiments conducted on several circuits from the IWLS benchmark set [15] and on some additional circuits. In all experiments, BDDs are built for the circuits while applying dynamic reordering. At the end, reordering is applied again. No hard groups are specified. Table 1 presents four sets of results. The column headings give the combinations of methods. For instance, *sift-sift* means ordinary sifting while building the BDDs, followed by ordinary sifting. *Symm* means symmetric sifting and *group* means group sifting. *best-ever* indicates the best results we have collected running experiments with different parameters and variants of the algorithms. Times are in seconds on a 275 MHz DEC Alpha workstation with 256 MB of memory.

In all cases except for the adders, adder/subtractors, *comp128*, *alu128*, and *dpath32*, the standard *misII* ordering [10] was used as initial order. For the others, contrived orders were used to test the robustness of the algorithms. The example named *xi30* is the function Ξ_{30} of Section 2.3, showing a linear gap between optimal orders and symmetric orders. The result of 91 nodes is optimal.

Overall, group sifting produces BDDs that are 11% smaller. The CPU times are quite similar, with the exception of a 52% increase in the case of *s38417*. However, even such an increase is easily compensated by the savings that the application that uses the BDDs (e.g., sequential redundancy removal) can achieve. This is especially true of a circuit as large as *s38417* [16].

Ordinary sifting in many cases puts variables with strong affinities close to each other. However, when two strongly attracted variables are adjacent, it is unlikely that their positions 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, on the other hand, does not have this problem, because it moves both variables at once.

5 Conclusions

Group sifting is an extension of sifting that allows a more effective exploration of the different variable orders of a decision diagram. We have presented an algorithm that extends the benefits of symmetry-based sifting to a wider class of functions. Experimental results confirm its effectiveness. The strategy of our group sifting method is to first find a good relative position for a variable—a good neighborhood—and then move the variable and its neighbors around in search of a better neighborhood. In addition, extended symmetry is used during sifting to detect variables that have very strong affinity. This approach avoids local minima where ordinary sifting is often trapped and has a modest overhead in terms of CPU time.

Considerable work remains to be done. We cannot regard the search for better aggregation criteria as terminated. We would also like to explore the application of group moves to search schemes other than sifting. Finally, we hope to report soon on the application of group sifting to problems in sequential verification and synthesis, where constraints on the orders are often found, or the problems exhibit structural features that our algorithm may exploit.

References

- [1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision

		sift-sift		symm-group			group-group			best-ever		
name	in	nodes	CPU	nodes	ratio	CPU	nodes	ratio	CPU	nodes	ratio	CPU
vg2	25	155	0.68	183	1.18	0.69	183	1.18	0.71	80	0.52	0.81
alupla	25	1066	0.43	993	0.93	0.48	993	0.93	0.51	589	0.55	1.32
C1908	33	6380	7.10	6424	1.01	7.70	6428	1.01	10.41	5708	0.89	12.91
C432	36	1221	1.19	1221	1.00	1.28	1257	1.03	1.61	1210	0.99	1.27
too_large	38	412	3.32	497	1.21	3.40	336	0.82	3.38	319	0.77	36.90
mm9b	38	2136	3.34	2646	1.24	3.28	1683	0.79	3.01	1527	0.71	3.31
mm9a	39	2023	3.06	2023	1.00	3.10	2386	1.18	6.61	1112	0.55	6.29
seq	41	1212	5.76	1198	0.99	5.85	1198	0.99	5.89	1190	0.98	7.11
C1355	41	32274	60.72	36127	1.12	70.80	26756	0.83	38.95	25866	0.80	266.41
C499	41	29570	82.29	31517	1.07	86.48	26909	0.91	26.65	25866	0.87	158.26
k2	45	1322	2.31	1319	1.00	2.43	1319	1.00	2.44	1246	0.94	4.18
apex1	45	1300	1.65	1311	1.01	1.76	1311	1.01	1.75	1246	0.96	4.11
accpla	50	1714	1.87	1741	1.02	2.07	1741	1.02	2.06	1626	0.95	10.13
C3540	50	23850	24.48	23947	1.00	24.71	23867	1.00	43.57	23828	1.00	55.25
C880	60	4479	2.51	4568	1.02	2.69	5653	1.26	4.70	4083	0.91	13.51
sbc	68	1043	1.20	1060	1.02	1.34	1060	1.02	1.34	917	0.88	3.49
dalv	75	1211	3.42	1229	1.01	3.46	860	0.71	3.58	689	0.57	4.00
xi30	90	91	0.39	91	1.00	0.38	91	1.00	0.41	91	1.00	0.38
sl423	91	3857	4.74	4168	1.08	5.79	3277	0.85	5.05	1796	0.47	18.32
dpath32	93	702	2.41	702	1.00	2.60	642	0.91	2.35	485	0.69	3.24
mm30a	123	18008	48.17	17797	0.99	48.46	17626	0.98	29.64	11065	0.61	289.39
i3	132	133	0.24	133	1.00	0.28	133	1.00	0.28	133	1.00	0.24
i8	133	2181	2.34	2189	1.00	2.51	2189	1.00	2.53	1276	0.59	7.00
apex6	135	671	0.96	622	0.93	1.05	622	0.93	1.04	498	0.74	3.42
frg2	143	1604	2.84	1434	0.89	2.93	1438	0.90	2.92	963	0.60	4.59
ind1sn	148	3236	4.66	2653	0.82	5.02	2791	0.86	5.41	1796	0.56	7.85
ind2sn	172	5686	10.80	5594	0.98	11.31	4424	0.78	11.62	2989	0.53	24.12
ind3sn	174	5221	14.17	4703	0.90	14.71	4907	0.94	18.14	3201	0.61	20.27
C5315	178	2285	5.32	2332	1.02	5.65	2262	0.99	6.26	2104	0.92	59.24
s5378	199	3842	7.24	2910	0.76	7.90	2010	0.52	6.14	1932	0.50	12.69
C7552	207	10156	29.76	10072	0.99	33.10	11517	1.13	41.71	3730	0.37	68.81
C2670	233	2057	9.03	2113	1.03	8.49	2604	1.27	8.90	2008	0.98	17.94
s9234.1	247	3841	15.16	3997	1.04	16.78	3490	0.91	17.52	3045	0.79	34.93
des	256	3030	8.98	2948	0.97	9.24	2961	0.98	9.73	2945	0.97	11.38
compl28	256	526	16.73	385	0.73	27.83	385	0.73	28.11	385	0.73	17.79
il0	257	40479	136.79	24045	0.59	96.30	23460	0.58	93.29	20660	0.51	211.64
add128	257	771	17.91	642	0.83	15.78	642	0.83	15.87	642	0.83	15.67
add128r	257	2214	25.33	667	0.30	16.12	655	0.30	18.22	642	0.29	16.26
adbs128	257	4222	8.03	4041	0.96	9.32	3866	0.92	9.99	1401	0.33	15.27
adbs128r	257	1401	3.82	1401	1.00	3.50	1401	1.00	3.52	1401	1.00	3.47
alu128	265	2093	114.10	1922	0.92	105.44	1929	0.92	111.65	1918	0.92	108.58
sl5850.1	611	18173	89.96	15322	0.84	71.87	18570	1.02	89.91	10111	0.56	289.29
sl3207.1	700	3189	36.92	3090	0.97	40.36	3064	0.96	47.24	2941	0.92	93.74
s38584.1	1464	16190	99.98	16106	0.99	141.96	14404	0.89	153.61	13303	0.82	382.76
s38417	1664	303636	3060.30	283298	0.93	3112.28	271781	0.90	4660.09	266772	0.88	8380.45
Totals	9749	570863	3982.41	533381	0.93	4038.48	507081	0.89	5558.32	457335	0.80	10707.99

Table 1: Comparison of Different Sifting Schemes.

- diagrams and their applications. In *Proc. ICCAD*, pages 188–191, Santa Clara, CA, Nov. 1993.
- [2] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. Presented at IWLS, Granlibakken, CA, May 1995.
 - [3] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proc. DAC*, pages 40–45, Orlando, FL, June 1990.
 - [4] R. Bryant and Y. Chen. Verification of arithmetic functions with binary moment diagrams. Technical Report CMU-CS-94-160, Carnegie Mellon University, May 1994.
 - [5] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, Aug. 1986.
 - [6] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of boolean comparison method based on binary decision diagrams. In *Proc. ICCAD*, pages 2–5, Nov. 1988.
 - [7] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proc. EDAC*, pages 50–54, Amsterdam, Feb. 1991.
 - [8] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *Proc. ICCAD*, pages 472–475, Santa Clara, CA, Nov. 1991.
 - [9] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proc. DAC*, 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 *Proc. ICCAD*, pages 6–9, Santa Clara, CA, Nov. 1988.
 - [11] S.-I. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proc. DAC*, pages 272–277, Dallas, TX, June 1993.
 - [12] D. Möller, P. Molitor, and R. Drechsler. Symmetry based variable ordering for ROBDDs. In *IFIP Workshop on Logic and Architecture Synthesis*, Grenoble, Dec. 1994.
 - [13] S. Panda, F. Somenzi, and B. F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Proc. ICCAD*, pages 628–631, San Jose, CA, Nov. 1994.
 - [14] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. ICCAD*, pages 42–47, Santa Clara, CA, Nov. 1993.
 - [15] S. Yang. Logic synthesis and optimization benchmarks user guide version 3.0. Technical report, MCNC, Research Triangle Park, NC, Jan. 1991.
 - [16] H. Cho, G. D. Hachtel, E. Macii, M. Poncino, K. Ravi, and F. Somenzi. “Approximate finite state machine traversal: Extensions and new results.” Presented at IWLS95, Lake Tahoe, CA., May 1995.