Speeding up Variable Reordering of OBDDs*

Christoph Meinel
University of Trier, FB IV-Informatik †
54286 Trier, Germany
e-mail: meinel@uni-trier.de

Anna Slobodová
ITWM-Trier, Bahnhofstr. 30-32
54292 Trier, Germany
slobodova@itwm-trier.fhg.de

Abstract

The use of Ordered Binary Decision Diagrams (OBDDs) as a representation of Boolean functions brought essential progress in many different applications. The optimization of the OBDD-size by the choice of the variable ordering is known to be NP-hard. The known heuristics for finding an initial ordering and for reordering still have insufficient performance.

Rudell's sifting is one of the most successful reordering algorithms that is application independent and can be used dynamically. In this paper, we propose a method based on some communication complexity considerations that improves the time performance of the sifting. The main idea is to restrict the reordering of variables to blocks that are determined according to readily computable OBDD-measure. Experimental comparison of the proposed block-restricted sifting with the original algorithm shows a speed-up by factor two without any loss in the final size.

1. Introduction

Ordered Binary Decision Diagrams (OBDDs) are due to their nice algorithmical properties [Bry86] one of the few known representations for Boolean functions that are used in a wide range of different applications. Their success in solving complex industrial problems implicitly involves requirements to minimize their size. A big effort was invested to the generalization of the OBDD structure with purpose of obtaining either a more compact representation or faster algorithms for specific operations. This resulted in application specific data structures with a restricted use. Another approach to minimizing the size of considered OBDDs (e.g., [BMS95b, JLMC, FKB95]) is based on transformations of the functions of interest.

The main source for the optimization of the OBDD size is the choice of a *good* variable ordering, i.e., an ordering under which the OBDDs for the considered functions have manageable size. There are examples that show exponential gaps in the sizes of functionally equivalent OBDDs which differ in their variable orderings. Since the problem of finding an optimal variable ordering is known to be NP-hard [THY93, BW96] and there is a lack of knowledge about the approximability of the problem, there is a strong need for good working heuristics.

The list of the heuristics for an initial variable ordering and reordering developed so far is too long to be presented here. However, none of these heuristics meets the increasing needs of the industrial applications. The form of the input into an ordering heuristic, i.e., representation of the considered functions, depends on an application for which it was developed. It may be a combinatorial circuit, Boolean formulas, graphs, or even OBDDs. A particular representation may implicitly include additional information about the functions of interest, e.g., circuit topology or formula shape, that can be used by the heuristic. In contrast to this case, if we are assumed to start from OBDDs, we have to work 'blind'. Nevertheless, this kind of heuristics is of special importance, since they are *independent on a concrete application* and *can be used dynamically*. The last feature is important, if the functions under consideration are changing during the computation. A typical example is the symbolic simulation of combinatorial circuits, where the representation of some gate exists merely as long as its fanout will be used for the computation of the representation of some other gate. Another example is the reachability analysis of sequential circuits. Here, the set of reachable states is changed throughout the computation. In order to work with a suited ordering at any time, it is useful to have possibility to change it dynamically.

Usually, the heuristics developed for this purpose proceed in greedy manner using certain local, e.g., [Rud93, FYBS93] or global changes, e.g., [BMS95a], within the variable ordering. The most successful

*in Proc. ACM/IEEE ICCD'97, Austin, Texas
†also ITWM-Trier, Bahnhofstr. 30-32, 54292 Trier

of them is the so-called sifting introduced by Rudell [Rud93]. Sifting is based on the idea of searching for the best position for each variable. The single variables are moved through the whole ordering, one at a time. The basic operation used to implement this movement is the exchange of two neighbouring variables (*swap* operation), which can be done in linear time with respect to the number of nodes labelled by the swapped variables. However, the total time for sifting grows very fast with the number of variables. In applications with thousands of variables, sifting of only one variable over the whole ordering takes too much time.

Several attempts have been made to improve the performance of the original sifting.A speed-up can be reached by keeping information about independent variables that can be swapped without any changes in the OBDD structure [S96a]. Another source of improvement is an appropriate setting of the parameter that restricts the allowed growth of the OBDDs. While in the original paper [Rud93], this parameter was set to 2.0, experiments have shown that a substantially lower setting increases the time performance without substantial increase of the final size of the OBDD [S96b]. Another possibility to improve the performance of sifting consists of a pre-estimation of the possible size reduction of the considered OBDDs that could be obtained by sifting of the current variable in a given direction, and interruption of the sifting process if this estimation gives worse values than the optimum found so far. In [PSP94, PS95], the quality of the found variable ordering was improved by grouping variables according to some symmetry properties into blocks that are sifted instead of single variables. Unlike the latter attempts, our goal is to improve the time performance of sifting even if it implies a penalty in the size of the final OBDDs.

In this paper, we propose a method of *block-restricted sifting*. The main idea consists of a reducing the number of used swap operations by restricting their application on variables within certain blocks. Determination of the variable blocks follows from a communication complexity argument: a weak information flow between two parts of an OBDD indicates a candidate for the boundary between two blocks. The information flow can be estimated by an easily computable OBDD characteristic that we call *subfunction profile*. This improves the time performance of the original sifting strategy considerably, without causing a loss in the final size. Similarly as in the case of the original sifting, no additional knowledge about the represented function is used. In our experiments, the block-restricted sifting applied in the final reordering results in about 50,7% time profit and costs less than 0,5% size increase. In the experiments with a dynamic use of the heuristic, the time gain amounts to 52,4%. Interestingly, we ended up with OBDDs that were by 32,4% smaller than those obtained by the original algorithm.

2. Block-Restricted Reordering

In this section, we give the intuition behind the proposed block-restricted sifting strategy and describe its implementation in the CUDD-package [S96a]. We look for an easily computable structural property of an OBDD that could tell us something about the represented function.

Let P be an (multirooted shared) OBDD over n variables ordered with respect to an ordering π. For each i, $1 \leq i \leq n+1$, let us consider a horizontal *cut* of P that divides it into a top part with the nodes labelled by the first $i-1$ variables in the order π and the bottom part with the remaining nodes. The amount of communication between these two parts is reflected by the number of the nodes that are referred (entered via edges) from the top part. These nodes correspond to the subfunctions (cofactors) that can be obtained from the functions represented by the roots by fixing values of the first $i-1$ variables. This set is called *subfunction level* (S_i). Obviously, the first level S_1 consists of all roots and the last one $S_n + 1$ of the sinks. The sequence of sizes of subfunction levels $\{S_i\}_{i=1}^{n+1}$ form a *subfunction profile*.

We use the shape of the subfunction profile to determine certain blocks of variables. Then each variable is sifted within its block only. The underlying idea is that a narrow subfunction level indicates a weak information flow from the corresponding top to the bottom part. An important observation is that reordering of variables in one part does not change the other part. Consequently, the restriction of sifting on the parts that do not communicate too much should not result in an essential loss in optimization, while time could be decreased considerably. This idea can be condensed into four steps:

Block-restricted Sifting.

1. Compute the subfunction profile for the OBDDs.
2. Find local minima in this profile.
3. Define variable blocks by choosing some minima as boundaries.
4. Reorder OBDDs within the chosen blocks by means of sifting.

3. Computing Subfunction Profile

The efficiency of the first step in block-restricted sifting - the computation of the subfunction profile of OBDDs depends on the OBDD package used. The procedure described here was designed for the CUDD-package but it works for all implementations with the following properties:

– each node has direct access to its sons

– for any given i, it is possible to reach all nodes labelled by the i-th variable efficiently.

For each OBDD-node, there is a structure `node` where then-son (`node.then`), else-son (`node.else`) and the variable (`node.var`) of the `node` are stored. Let `x.level` be the index of the variable `x` in the given ordering (the top variable is assumed to be on level 1). `node.var.level` for a sink is defined as `nvars+1` where `nvars` is the number of variables in the support of the OBDDs. The nodes that are labelled by the same variable form *a node level.* Let `table[i]` store the nodes in the i-th node level, `s[j]` be the number of nodes in the j-th subfunction level (initially set to 0), and `roots[0..nroots-1]` be all roots of the OBDDs.

```
ComputeSubfunctions(table,roots) {
  for (i=0; i<nroots; i=i+1) {
    node=roots[i];
    if (NotMarked(node)) {
      for (j=1; j<=node.var.level; j++)
        s[j]=s[j]+1;
      SetMark(node);
    }
  }
  for (i=1; i<=nvars; i=i+1)
    for each node in table[i]
      for both sons of the node
        if (NotMarked(son)) {
         for (j=i+1; j<=son.var.level; j=j+1)
           s[j]=s[j]+1;
         SetMark(son);
        }
}
```

At the first sight, the node and subfunction levels seem to be closely related sets: the i-th node level contains those cofactors from the i-th subfunction level that depend on the i-th variable in the order. However, as the next example shows, the size of these sets and hence, the node and subfunction profiles can be very different.

Example: Consider the node and subfunction profile of OBDDs for functions

$f_i = x_i \wedge x_n$, for $1 \leq i \leq n$

with respect to variable ordering

$x_1 < x_2 < \ldots < x_n$

Let $s[i]$, respectively $v[i]$, be the size of the i-th subfunction, respectively node, level.

i	1	2	3	...	k	...	n	$n+1$
$s[i]$	n	$n-1$	$n-2$		$n-k+1$		1	2
$v[i]$	1	1	1		1		1	2

4. Experiments

The goal of this section is to assess the strategy described above by means of a series of experiments. The strategy is implemented in the CUDD-package.

The implementation of sifting in the CUDD-package follows the description in [Rud93]. Variables are first sorted according to the decreasing size of the node levels, then they are sifted with respect to this priority. Each variable is sifted up and, consequently, down until either the top, respectively the bottom, level is reached, or an interruption criterion is fulfilled. The sifting of a variable can be interrupted if either the size of the OBDDs is increased more than GROWTHFACTOR times the best size reached until now, or if there is no better position for the variable possible anymore. Finally, the variable is sifted to the best position.

Block-restricted sifting starts by computing subfunction profile as described in *ComputeSubfunctions()*. Considering the array `s` as the values of a function s defined on $\{1, 2, ..., n\}$ by $s(i)$ =`s[i]`, the local minima of s are determined. These minima serve as candidates for the boundaries between the blocks. Some of them are excluded from the consideration by using two parameters – MINDOWN and MINUP which values are set by the user. i is chosen if $\frac{s(i-1)-s(i)}{s(i)} \geq$ MINDOWN and $\frac{s(i+1)-s(i)}{s(i)} \geq$ MINUP. In addition, the parameter MINBLKFACTOR prescribe the minimal number of variables in one block. The selection criterion in the case of more local minima with a small distance is their value: the candidate with the minimal value is chosen.

In all experiments, we have set GROWTHFACTOR to 1.2, i.e., at most 20% increase is allowed during the sifting. MINBLKFACTOR is set to 0.1, i.e., each block contains at least 10% of variables. MINDOWN is set to 0.01 and MINUP to 0.02. The OBDDs used in the experiments include the functions of the outputs of all gates (not only outputs! The reason for it is to work with more functions and larger OBDDs) of the circuits from the ISCAS and LGSynth'91 benchmark sets.

All experiments were done on a Sun SPARC 20. The time limit was set to 10,000 s, the memory limit to 100 Mbyte.

4.1. One-Time Block-Restricted Sifting

In the first series of experiments, the effect of the suggested restriction of sifting is investigated for the case of the one-time reordering of the OBDDs. The results are shown in Table 1.

FINAL REORDERING						
	sifting (1)		block-restricted sifting (2)			
Circuit	size (nodes)	time (sec)	size (nodes)	time (sec)	Δsize (%)	Δtime (%)
C1355	121,264	206.30	121,264	74.04	0.00	-64.11
C1908	26,767	28.96	29,993	12.08	12.05	-58.29
C3540	124,926	132.20	125,008	81.95	0.07	-38.01
C499	36,603	107.64	36,836	54.48	0.64	-49.39
C7552	115,474	456.91	108,833	170.42	-5.75	-62.70
C880	35,990	42.49	39,460	32.75	9.64	-22.92
i10	83,028	221.91	88,083	98.83	6.09	-55.46
mm30a	982,326	3,158.50	982,326	1,482.40	0.00	-53.07
s15850.1	39,615	135.65	39,865	44.62	0.63	-67.11
s38584.1	36,311	95.72	36,311	97.71	0.00	2.08
s4863	337,277	555.46	341,436	280.15	1.23	-49.56
s6669	187,249	206.45	187,249	206.90	0.00	0.22
Total	2,126,830	5,348.19	2,136,664	2,636.33	0.46	-50.71

Table 1.

The OBDDs for all gates and outputs of the circuits were reordered by

1. original sifting, and by
2. block-restricted sifting

In all cases, the initial OBDDs were constructed by symbolic simulation of the circuits, with the same dynamical sifting, hence, the compared reorderings have started with the same OBDDs.

The methods are compared with respect to the final size and the time used by the final reordering. The time measured for the original sifting (1) includes only the time for the final reordering. The time for the block-restricted sifting (2) (5th column) includes not only the time for the final reordering, but also the time for computing subfunctions, and setting boundaries between the blocks. This overhead is negligible comparing with the overall time gain: Computing subfunctions and determining blocks takes about one second by all circuits, with two exceptions of s4863 (3 sec) and mm30a (7 sec). The time gain and the possible loss in the final size of the OBDDs in the cases of the block-restricted sifting (last two columns) are expressed relative to the time, respectively final size, of the OBDDs obtained by the original sifting.

4.2. Dynamic Block-Restricted Sifting

In the view of the experiments described in the previous subsection, the block-restricted sifting behaves quite stable – the loss of optimality in the size is negligible, which is a good indication for the possible use of the method in a dynamical reordering process. Since the effect of the proposed strategy used dynamically is hardly predictable by theoretical means, it is interesting to observe it experimentally.

The compared methods: original sifting (1) and the proposed reordering method (2), are used in the process of dynamic reordering during the symbolic simulation of circuits.

DYNAMIC REORDERING						
	sifting (1)		block-restricted sifting (2)			
Circuit	size (nodes)	time (sec)	size (nodes)	time (sec)	Δsize (%)	Δtime (%)
C1355	121,264	464.34	121,268	183.08	0.00	-60.57
C1908	26,767	92.24	25,955	30.05	-3.03	-67.42
C3540	124,926	447.63	146,793	335.78	17.50	-24.99
C499	36,603	293.64	34,797	136.39	-4.93	-53.55
C7552	115,474	2,557.25	292,374	2,044.84	153.19	-20.04
C880	35,990	146.48	41,429	146.97	15.11	0.33
i10	83,028	555.82	83,278	495.44	0.30	-10.86
mm30a	982,326	5,947.84	76,631	294.70	-92.20	-95.05
s15850.1	39,615	383.51	54,363	262.29	37.23	-31.61
s38485.1	36,311	549.62	37,678	669.53	3.76	21.82
s4863	337,277	1,328.27	330,611	1,187.47	-1.98	-10.60
s6669	187,249	581.85	192,104	564.55	2.59	-2.97
Total	2,126,830	13,348.49	1,437,281	6,351.09	-32.42	-52.42

Table 2.

In order to make the results comparable, the same scheme for dynamic reordering is used: the reordering is invoked whenever the size of the currently stored OBDDs grows too much, which is controlled by default parameters of the CUDD-package. When all outputs are computed, one final reordering (always with the respective method) is performed. The results are shown in Table 2. Since the reorderings are called dynamically depending on the growth of the size of the OBDDs, their number is different for the same circuit processed by the different heuristics. Hence, the time shown in the table is the total time needed for the creation of the OBDDs and for their final reordering. The time figures met our expectations motivated by the one-time reordering. Surprisingly, the total size of the final OBDDs decreases too.

4.3. Subfunction vs. Node Profile

The example in Section 3 shows how different the node profile and subfunctions profile for the same OBDD may differ. However, it is not always the case, i.e. it is easy to find an example where the measures are quite close. We wondered how far could node profile approximate subfunction profile of an OBDD.

Our experiments with the block-restricted sifting where the number of subfunctions was approximated by the number of nodes in each level (3) gave us evidence of unpredictability and non-robustness of the dynamic reordering. Having about 1.5% loss in size in the case when the reordering is performed only once (see Table 3), there were several examples of circuits for which the creation of the OBDDs couldn't be finished within the given time bound when heuristic (3) was used for dynamic reordering (Table 4). All experiments were done according the same scheme and with the same parameters setting as in previous two subsections.

FINAL REORDERING						
Circuit	sifting (1)		block-restricted sifting (3) (blocks by node profile)			
	size (nodes)	time (sec)	size (nodes)	time (sec)	Δsize (%)	Δtime (%)
C1355	121,264	206.30	121,264	72.95	0,00	-64.64
C1908	26,767	28.96	30.112	8.86	12.50	-69.41
C3540	124,926	132.20	134,314	45.97	7.51	-65.23
C499	36,603	107.64	36,836	54.43	0.64	-49.43
C7552	115,474	456.91	117,604	101.04	1.84	-77.89
C880	35,990	42.49	39,938	9.90	10.97	-76.70
i10	83,028	221.91	89.237	59.29	7.48	-73.28
mm30a	982,326	3,158.50	982,326	1478.15	0.00	-53.20
s15850.1	39,615	135.65	40.059	52.45	1.12	-61.33
s38584.1	36,311	95.72	38,295	39.94	5.46	-58.27
s4863	337,277	555.46	341,433	183.91	1.23	-66.89
s6669	187,249	206.45	187,465	146.88	0.12	-28.85
Total	2,126,830	5,348.19	2,158,883	2,253.77	1.51	-57.86

Table 3.

DYNAMIC REORDERING				
Circuit	sifting (1)		block-restricted sifting (3) (blocks by node profile)	
	size (nodes)	time (sec)	size (nodes)	time (sec)
C1355	121,264	464.34	121,261	227,28
C1908	26,767	92.24	30,355	45.93
C3540	124,926	447.63	1,940,687	1,537.64
C499	36,603	293.64	34,727	183.19
C7552	115,474	2,557.25	4,695,944	(*)
C880	35,990	146.48	183,796	196.20
i10	83,028	555.82	110,951	521.11
mm30a	982,326	5,947.84	4,193,666	(**)
s15850.1	39,615	383.51	249,596	436.10
s38485.1	36,311	549.62	130,078	804.68
s4863	337,277	1,328.27	356,283	624.88
s6669	187,249	581.85	420,266	922.54

Table 4 (* time exceeded, ** out of memory).

5. Conclusions

In this paper, we showed a way of deriving an information from the structure of OBDDs for speeding-up a variable reordering. The proposed block-restricted sifting runs two times faster on the series of experiments. Besides that, the heuristic used dynamically brought the overall decrease of the final OBDD size as well. The main advantages of the method are

1. its general usability (as no additional information about represented functions is used), and
2. the possibility to use it dynamically

These features make the strategy widely applicable.

Although the proposed strategy was implemented for sifting, the idea is applicable to some other reordering heuristics as well. The first experiments with the heuristics implemented in the CUDD-package show similar behaviour, with the exception of 'window reorderings' where the restriction to blocks is obviously meaningless.

According to the experiments with blocks determined by node profile, subfunction profile seems to be a more adequate measure w.r.t. the information flow in OBDD. This hypothesis was also supported by the following series of experiments. In the original sifting, the choice of a variable for sifting is controlled by the size of the node levels (the variable that corresponds to the widest node level was chosen at first).Table 5 shows the results of experiments with sifting where this choice criterion is changed by replacing node levels by subfunction levels.

Circuit	sifting driven by nodes		sifting driven by subfunctions			
	size (nodes)	time (sec)	size (nodes)	time (sec)	Δsize (%)	Δtime (%)
C1355	121,264	464.34	121,264	470.39	0.00	1.30
C1908	26,767	92.24	25,514	62.66	-4.68	-32.07
C3540	124,926	447.63	130,795	410.67	4.70	-8.26
C499	36,603	293.64	34,506	194.69	-5.73	-33.70
C7552	115,474	2,557.25	88,100	740.88	-23.71	-71.03
C880	35,990	146.48	38,423	145.03	6.76	-0.99
i10	83,028	555.82	78,809	501.86	-5.08	-9.71
mm30a	982,326	5,947.84	158,322	789.18	-83.88	-86.73
s15850.1	39,615	383.51	36,753	335.40	-7.22	-12.54
s38485.1	36,311	549.62	44,322	555.19	22.06	1.01
s4863	337,277	1,328.27	330,612	1,224.92	-1.98	-0.78
s6669	187,249	581.85	187,450	642.42	0.11	10.41
Total	2,126,830	13,348.49	1,274,870	6,073.29	-40.06	-54.50

Table 5.

In all experiments, the same setting of parameters is taken. The tuning of the parameters is a source of possible improvement of the block-restricted sifting.

Some improvement can be achieved also by a combination of the proposed method with some other heuristic. Subfunction profile of an OBDD that is built w.r.t. a 'good' variable ordering determines the blocks much clearer than one for a random variable ordering. Hence, if the OBDDs are reordered dynamically, it is necessary for the success to keep 'good' variable ordering. We can solve it by using a more time consuming heuristic that improves the ordering whenever it seems to be necessary. The experiments in this direction are ongoing.

References

[BMS95a] J. Bern, Ch. Meinel, A. Slobodová: Global Rebuilding of OBDDs – Avoiding Memory Requirement Maxima. *In Proc. of CAV*, (LNCS 939):299–308, 1995.

[BMS95b] J. Bern, Ch. Meinel, A. Slobodová: Efficient OBDD–Based Boolean Manipulation in CAD Beyond Current Limits, *In Proc. 32nd IEEE/ACM DAC*, pages 408–413, 1995.

[Bry86] R.E. Bryant: Graph Based Algorithms for Boolean Function Manipulation. *IEEE Transaction on Computers*, (C-35):677–691, 1986.

[BW96] B. Bollig, I. Wegener: Improving the Variable Ordering of OBDDs is NP–complete. *IEEE Transaction on Computers* 45(9):993–1002, 1996.

[FKB95] M. Fujita, Y. Kukimoto, R.K. Brayton: BDD minimization by truth table permutations. *In. Proc. International Workshop on Logic Synthesis.* 1995.

[FYBS93] E. Felt, G. York, R. Brayton, A. Sangiovanni-Vincentelli: Dynamic Variable Reoredering for BDD Minimization. *In Proc. EURO-DAC*, pages 130–135, 1993.

[JLMC] S. Jha, Y. Lu, M. Minea, E.M. Clarke: Equivalence Checking Using Abstract BDDs. *Technical Report CMU*, October 1996.

[PS95] F. Somenzi, S. Panda: Who are the Variables in Your Neighbourhood? *In Proc. ACM/IEEE ICCAD*, 1995.

[PSP94] S. Panda, F. Somenzi, B.F. Plessier: Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. *In Proc. ACM/IEEE ICCAD*, pages 628–631, 1994.

[Rud93] R. Rudell: Dynamic Variable Ordering for Ordered Binary Decision Diagrams. *In Proc. ACM/IEEE ICCAD*, pages 42–47, 1993.

[S96a] F. Somenzi: CUDD: CU Decision Diagram Package. Release 1.1.1. *Technical Report*, University of Colorado, 1996.

[S96b] F. Somenzi: *Personal communication.*

[THY93] S. Tani, K. Hamaguchi, S. Yajima: The Complexity of the Optimal Variable Ordering Problem of Shared Binary Decision Diagrams. *In Proc. ISAAC, (LNCS 762):389–398, 1993.*