Preprint from Proceedings of IWLS99, Granlibakken, USA, May 1999

Solving the Multiple Variable Order Problem for Binary Decision Diagrams by Use of Dynamic Reordering Techniques

Christoph Scholl Bernd Becker Andreas Brogle

Institute of Computer Science
Albert–Ludwigs–University
D 79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de

Abstract

*Reduced Ordered Binary Decision Diagrams (*ROBDD*s) gained widespread use in logic design verification, test generation, fault simulation, and logic synthesis [16, 7]. Since the size of an* ROBDD *heavily depends on the variable order used, there is a strong need to find variable orders that minimize the number of nodes in an* ROBDD*. In certain applications we have to cope with* ROBDD*s with different variable orders, whereas further manipulations of these* ROBDD*s require common variable orders. In this paper we solve the problem to transform* ROBDD*s with different variable orders into a good common variable order. To do so, we make use of dynamic variable ordering techniques.*

1 Introduction

Binary Decision Diagrams (BDDs) as a data structure for representation of Boolean functions were first introduced by Lee [15] and further popularized by Akers [1] and Moret [18]. In the restricted form of ROBDDs they gained widespread use, because ROBDDs are a canonical representation and allow efficient manipulations [6]. Some fields of application are logic design verification, test generation, fault simulation, and logic synthesis [16, 7]. Most of the algorithms using ROBDDs have run time polynomial in the size of the ROBDDs. The sizes themselves depend on the variable order used. Thus, there is a need to find a variable order that minimizes the number of nodes in an ROBDD.

The existing heuristic methods for finding good variable orders can be classified into two categories: initial heuristics which derive an order by inspection of a logic circuit [16, 12, 13, 11] and dynamic reordering heuristics which try to improve on a given order [14, 20, 10, 3, 9]. Sifting introduced by Rudell [20] has emerged so far as the most successful algorithm for dynamic reordering of variables. This algorithm is based on finding the local optimum position of a variable, assuming all other variables remain fixed. The position of a variable in the order is determined by moving the variable to all possible positions while keeping the other variables fixed.

In this paper we deal with the fact that certain applications have to cope with ROBDDs represented with different variable orders. Then we have to solve the problem to transform ROBDDs with different variable orders into a common variable order. This problem is called *multiple variable order* problem in [23].

One application of this type is reachability analysis and formal verification using partitioned-ROBDDs [19]: ROBDDs are partitioned, i.e. decomposed into sub–ROBDDs. In this way the application can deal with each ROBDD separately and optimize their sizes independently. For image computation however Boolean operations for ROBDDs represented with different variable orders have to be performed. Thus, at first they are transformed into the same variable order.

Moreover, it has been suggested [8] that ROBDDs are used to communicate between different synthesis and verification tools. ROBDDs are dumped to files by one tool and undumped by other tools. If the ROBDDs originate from different tools, it is clear that they can have different variable orders.

Another application for the multiple variable order problem occurs in connection with functional simulation [2, 17, 21] using binary decision diagrams. In these approaches ROBDDs for circuits are computed and then used for compiler-driven simulation. To control the ROBDD sizes intermediate variables are introduced as cut points based on size limits for the ROBDD sizes. The result of this process is a partition of the circuit into clusters. To speed up cycle based functional simulation for the output functions of these clusters (primary outputs or cut points) the ROBDDs of the corresponding characteristic functions χ are computed ($\chi((i_1, \ldots, i_n, o_1, \ldots, o_m) = \bigwedge_{i=1}^{m}(o_i \equiv f_i(i_1, \ldots, i_n))$, where f_i are the output functions and o_i are corresponding output variables). Then the characteristic functions of the clusters are evaluated in topological order.

In the partitioning approach of [21] we use variable reordering to minimize the sizes of the characteristic functions separately. However, to minimize the evaluation time the number of clusters has to be minimized, i.e. it is checked whether pairs of clusters can be merged into one. To do so, the ROBDDs for the characteristic functions are transformed into the same variable orders and then an AND operation is applied to the ROBDDs. The merging is accepted, when the result is smaller than a certain size limit. (In this special application the fact, that it is not possible to transform the ROBDDs for the characteristic functions into a common variable order within a certain node limit for the ROBDDs, can be accepted, since the algorithm still works although we might lose quality of the result.) For reasons of run time efficiency it can make sense to decide early, if the transformation into a common variable order works or should be aborted.

In [23] the problem to transform two ROBDDs into a common variable order is solved by inspection of the two variable orders, computation of an intermediate variable order based on these two variable orders and a transformation of the two ROBDDs into the intermediate variable order by level exchanges. In contrast to this approach we use dynamic reordering techniques [20] to transform the two ROBDDs into a common variable order and thereby dynamically adapt the ordering to the resulting new ROBDDs. Experimental results demonstrate that in our approach time can be traded off for quality of the result by allowing reordering for adaption of the ordering more frequently. Compared to [23], we significantly improve the size of the final ROBDDs within a reasonable amount of runtime.

The paper is structured as follows: In Section 2 we give a brief review of BDDs. In Section 3 we give a theoretical background and we present our heuristic to transform two ROBDDs into a common variable order, in Section 4 we show some experimental results and Section 5 concludes the paper.

2 Preliminaries

BDDs are representations of Boolean functions. In the restricted form of ROBDDs they even provide canonical representations. As defined in [6], ROBDDs are ordered, i.e. on each path from their root to a terminal node each input variable occurs only once and on each path the input variables occur in the same order. Since we work only with ROBDDs in the following we briefly call them BDDs.

Given a variable order π for the input variables of function f there is a unique BDD using variable order π, which is denoted by $BDD_{\pi}(f)$ in this paper. It is well known that the size of a BDD is largely influenced by the choice of the variable ordering [6].

Dynamic reordering [20] allows BDDs to adapt to the changing functions as computation proceeds. When BDD sizes grow too large during the computation of a Boolean operation, the computation is aborted, all BDDs computed so far are minimized by a transformation to another order using a dynamic reordering heuristics like sifting and the operation is tried again. The operation is aborted, when the node number would exceed some reordering limit. Usually, the reordering limit is initialized to some smaller number to reorder also BDDs at the beginning of a series of BDD computations, which are typically smaller, and is increased step by step during the computation until it reaches an absolute node limit [22].

3 The Multiple Variable Order Problem

Suppose we have two Boolean functions f and g, which are represented by BDDs $BDD_{\pi_f}(f)$ and $BDD_{\pi_g}(g)$, respectively. Then the solution of the *Multiple Variable Order* problem (MVO) for $BDD_{\pi_f}(f)$ and $BDD_{\pi_g}(g)$ means the following:

Find a variable order $\pi_{f,g}$, such that the sizes of $BDD_{\pi_{f,g}}(f)$ and $BDD_{\pi_{f,g}}(g)$ as *shared* BDD [5] are minimized.

3.1 Theoretical background

From the NP completeness of the variable ordering problem for *single* BDDs [24, 4] we can easily conclude that the task to solve MVO exactly is a hard problem[1].

Theorem 1 *MVO is an NP complete problem.*

Furthermore it can be shown that there are pairs of Boolean functions, where a blow up of the BDD sizes compared to the BDD sizes of the single BDDs can not be avoided, since it is not possible to find an efficient *common* variable order for the two BDDs. The following theorem gives an example for such a case.

Theorem 2 *Let* $f = \bigvee_{i=1}^{n} \bigwedge_{j=1}^{n} x_{ij}$ *and* $g = \bigvee_{j=1}^{n} \bigwedge_{i=1}^{n} x_{ij}$. *There are variable orders* π_f *and* π_g *such that* $BDD_{\pi_f}(f)$ *and* $BDD_{\pi_g}(g)$ *have (optimal) sizes* $n^2 + 2$, *respectively, but for all variable orders* π $BDD_{\pi}(f)$ or $BDD_{\pi}(g)$ *has a size of at least* $2^{\frac{n}{2}}$.

I.e. f and g in Theorem 2 can be represented efficiently, when different orders for f and g are allowed, but there is no common variable order, which leads to efficient representations for *both* f and g[2].

3.2 Solution of MVO

Here we present a heuristic to solve MVO approximately.

The same problem was already studied by Stangier et al. in [23]. They solve the problem by computation of an intermediate variable order $\pi_{f,g}$ based on π_f and π_g. Then a transformation of $BDD_{\pi_f}(f)$ and $BDD_{\pi_g}(g)$ to $\pi_{f,g}$ by level exchanges is performed. In contrast to this approach we use dynamic reordering techniques [20] to transform the two BDDs into a common variable order $\pi_{f,g}$ which thereby is dynamically adapted to the currently involved BDDs.

First of all, we choose the larger one of the two BDDs to start with. W.l.o.g. assume that $BDD_{\pi_f}(f)$ is the larger BDD. Now we transform *cofactors* of g step by step to the order of the BDD for f.

More precisely, we traverse $BDD_{\pi_g}(g)$ in a depth first manner and transform cofactors of g, which correspond to nodes in $BDD_{\pi_g}(g)$ into the order of the BDD for f. Suppose the current order of the BDD $BDD_{\pi_{old}}(f)$ for f is π_{old} and suppose we have reached node v of $BDD_{\pi_g}(g)$ labeled by variable x_i. Since we traverse $BDD_{\pi_g}(g)$ depth first, we have already computed for low–son $low(v)$ and high–son $high(v)$ $BDD_{\pi_{old}}(g_{low(v)})$ and $BDD_{\pi_{old}}(g_{high(v)})$, which have the same variable order as $BDD_{\pi_{old}}(f)$. Now we simply compute in variable order π_{old} the if–then–else operation $ite(x_i, BDD_{\pi_{old}}(g_{low(v)}), BDD_{\pi_{old}}(g_{high(v)}))$. The result is a representation for the function g_v represented at node v of $BDD_{\pi_g}(g)$, now in same variable order as the BDD for f.

During the computation of the new BDD for g_v by $ite(x_i, BDD_{\pi_{old}}(g_{low(v)}), BDD_{\pi_{old}}(g_{high(v)}))$, we use *dynamic reordering*. If the reordering limit is exceeded during this computation, dynamic reordering (sifting) is applied to simultaneously minimize the BDDs for f and all BDDs computed in variable order π_{old} so far. If dynamic reordering does not give up, after the call of operation ite we have BDDs for f, g_v and all other functions for nodes of g visited so far in a (possibly new) variable order π_{new}.

In this way we compute step by step variable orders, which are good both for f and cofactors of g and finally we have a variable order, which is also good for g. The adaption of the variable orders for the BDDs for f and g proceeds step by step during the computation of the BDD for g based on cofactors of g.

There still remains one point: In many applications dynamic reordering produces good results, but tends to slow down computation times by frequent reorderings.

For this reason we restrict dynamic reordering here. We introduce an upper limit for the number of reordering steps. We count the number of reorderings during the adaption of the variable orders for f and g and if this limit would be exceeded, the operation fails with the parameters currently used. This decision is motivated by our clustering approach for functional simulation [21]: We do not want to spend too much time on the computation of a common variable order for two clusters, which is likely to fail in the end or to produce huge BDDs. Moreover, it is clear, that the introduction of such a limit for the number of reorderings defines a trade-off between run time and the quality of the result in this application.

Finally, we have to adjust the initial reordering limit, if we restrict the number of reorderings. If we have chosen only a small number of reorderings, we do not want to waste the limited number of reordering steps by too early reorderings, which are performed for small BDDs and which are not yet absolutely necessary. Therefore we choose the higher initial reordering limit the smaller the allowed number of reorderings is. The initial reordering limit is chosen based on the allowed number of reorderings $maxreorder$ and on the sizes of the BDDs for which a common variable order has to be computed. For our practical experiments we use $size(BDD_{\pi_f}(f)) + \frac{(size(BDD_{\pi_g}(g))}{maxreorder+1}$ as initial reordering limit.

It it clear that the algorithm can be easily extended to the case that we have to compute common variable orders for sets of BDDs $BDD_{\pi_f}(f_1)), \dots BDD_{\pi_f}(f_n))$ with variable order π_f and BDDs $BDD_{\pi_g}(g_1)), \dots BDD_{\pi_g}(g_m))$ with variable order π_g. Then we simply perform a bottom up construction of BDDs for $g_1, g_2, \dots, g_m$ starting with variable order π_f as described above. Of course, identical nodes/cofactors of g_i have to be visited only once.

4 Experimental Results

To evaluate our heuristic for the MVO problem, we integrated our heuristic in the CUDD package [22]. In a first experiment we use data originating from our approach for functional simulation [21] for larger circuits. We selected the last tries for cluster merging for different circuits (successful or not in our original algorithm), since at the end of the algorithm clusters are getting larger and therefore harder problems must be solved.

The experiments were performed on a SPARC 20 (256MB memory). The CPU time was limited to 2 hours and the node limit for the BDD package was 2000000.

We tried several choices for the maximum number of reorderings during the computation of common variable orders. The results are summarized in Table 1. In the second column the sizes of the two BDDs (number of nodes) are given for which MVO has to be solved. (Note that the BDDs represent not the output functions, but the characteristic functions for the clusters.) Columns dyn$< n >$ show the results for our approach with n as the maximum number of reorderings. dyn0, e.g., is the algorithm, when absolutely no reordering is allowed. The results are compared to the "greedy gradual" heuristic and the "greedy at once" heuristic

[1] To prove that MVO is NP hard, we simply have to add a function, which does not depend on the variable order, e.g. the constant 1 function, to transform an instance of the variable ordering problem for single BDDs into a corresponding instance of MVO

[2] The lower bound for the size of $BDD_{\pi}(f)$ or $BDD_{\pi}(g)$ can be proved by introducing a cut line after the first $\frac{n^2}{2}$ variables. Further details of the proof are omitted due to lack of space.

	sizes	dyn0	dyn1	dyn2	dyn3	dyn5	dyn7	dyn10	dyn15	dyn20	dyn100	gradual	atonce
C2670.ex1	104992 912	space out	145586 0:09:29 118451 0:15:03	122765 0:17:19 109654 0:21:54	125226 0:28:40 101180 0:32:54	125226 0:28:35 101180 0:32:52	125226 0:28:43 101180 0:32:58	125226 0:29:54 101180 0:34:17	125226 0:28:34 101180 0:32:48	125226 0:28:36 101180 0:32:49	125226 0:28:37 101180 0:32:50	time out	space out
C2670.ex2	4127 244	space out	space out	119610 0:00:33 15198 0:01:21	108120 0:00:47 15177 0:01:36	37060 0:01:42 23709 0:02:27	15383 0:02:36 10639 0:02:45	15383 0:02:48 10639 0:03:09	15383 0:02:36 10639 0:02:55	15383 0:02:38 10639 0:02:58	15383 0:02:37 10639 0:02:57	time out	space out
C3540.ex1	196 52756	57001 0:00:32 55004 0:01:03	57001 0:00:31 55004 0:01:02	57001 0:00:31 55004 0:01:02	60722 0:00:31 55051 0:01:04	60722 0:00:31 55051 0:01:03	60722 0:00:31 55051 0:01:03	60722 0:00:35 55051 0:01:09	60722 0:00:31 55051 0:01:03	60722 0:00:31 55051 0:01:04	60722 0:00:31 55051 0:01:04	70090 0:00:31 57563 0:01:03	64530 0:00:05 55086 0:00:41
C3540.ex2	84388 21688	286704 0:01:31 236898 0:06:20	314854 0:03:25 98558 0:06:17	72498 0:05:06 58936 0:05:50	71873 0:04:52 54264 0:05:33	72680 0:05:05 60122 0:05:51	73624 0:04:39 61009 0:05:23	73564 0:04:57 61009 0:05:44	78076 0:04:39 63175 0:05:27	78076 0:04:31 63175 0:05:17	75905 0:04:33 68260 0:05:35	205749 0:31:37 49920 0:32:57	110615 0:00:39 71662 0:01:51
C3540.ex3	98156 21668	447188 0:01:57 145923 0:05:24	239800 0:03:53 144122 0:06:32	163033 0:06:39 147264 0:09:01	155520 0:07:09 148754 0:09:55	155148 0:06:55 143755 0:09:33	155148 0:06:44 143755 0:09:22	155432 0:06:58 143832 0:09:43	155148 0:06:39 143755 0:09:17	153823 0:06:41 146977 0:09:22	155866 0:06:37 144036 0:09:15	192355 0:42:13 157765 0:45:17	204761 0:00:28 144447 0:03:14
C5315.ex1	188920 11751	space out	220903 0:11:05 200839 0:16:55	203993 0:17:43 199912 0:23:12	201828 0:18:06 199598 0:23:34	201735 0:18:30 199760 0:24:00	202288 0:18:08 199797 0:23:37	202288 0:19:00 199797 0:24:47	230055 0:11:18 202378 0:17:20	204426 0:18:12 199444 0:23:43	302024 0:43:12 212236 0:50:33	458547 1:10:29 232780 1:18:35	space out
C5315.ex2	188920 35516	space out	716291 0:14:46 273153 0:27:17	376732 0:29:08 276196 0:42:12	376732 0:29:00 276196 0:42:05	376732 0:28:58 276196 0:42:10	376732 0:28:57 276196 0:42:05	365118 0:28:30 258027 0:40:04	361214 0:28:10 256603 0:39:26	361214 0:27:55 256603 0:39:04	368964 0:29:00 278832 0:41:15	time out	space out
C5315.ex3	18097 35516	space out	space out	space out	space out	286943 0:10:46 153997 0:15:28	182825 0:19:43 151126 0:24:21	182745 0:20:43 151332 0:25:31	156600 0:21:31 138037 0:25:17	168679 0:24:10 145901 0:28:32	184384 0:22:23 149515 0:27:02	time out	560943 0:00:26 180645 0:08:34
C5315.ex4	8637 35516	space out	space out	370998 0:03:32 131756 0:07:09	155056 0:06:17 133691 0:09:24	155408 0:06:18 133886 0:09:26	156389 0:06:04 132412 0:09:08	161587 0:05:43 133563 0:08:49	157407 0:06:01 132418 0:09:06	157403 0:06:00 132418 0:09:04	161820 0:07:13 132699 0:10:16	time out	space out
C5315.ex5	2398 35516	space out	space out	space out	space out	443812 0:15:02 109155 0:19:08	184731 0:27:20 90581 0:29:52	96672 0:31:40 84482 0:33:51	190164 0:20:38 90468 0:23:28	102216 0:25:12 76107 0:26:55	102216 0:25:16 76107 0:26:59	time out	space out
C5315.ex6	18097 8637	space out	135979 0:00:38 32278 0:01:32	58591 0:00:53 28495 0:01:28	32701 0:01:25 29022 0:01:50	33151 0:01:24 29022 0:01:50	32050 0:01:23 27537 0:01:48	33150 0:01:24 29021 0:01:50	33150 0:01:24 29021 0:01:49	33150 0:01:23 29021 0:01:49	32099 0:01:49 28382 0:02:15	489577 1:19:15 42425 1:21:59	space out
C5315.ex7	2398 18097	space out	space out	space out	space out	space out	347442 0:32:11 205731 0:40:02	275216 0:58:47 216581 1:06:00	364031 1:06:54 187474 1:13:52	364031 1:07:13 187474 1:14:14	364031 1:07:02 187474 1:14:01	time out	space out
C5315.ex8	11751 35516	68902 0:00:31 49954 0:01:22	56510 0:01:06 49145 0:01:50	51201 0:02:17 49031 0:03:01	51443 0:02:15 49337 0:03:00	52836 0:01:04 49417 0:01:47	52845 0:01:12 49415 0:01:57	52811 0:01:03 49395 0:01:45	52795 0:01:01 49395 0:01:44	52771 0:01:04 49395 0:01:50	55921 0:02:36 48749 0:03:21	time out	224196 00:00:34 85068 00:04:21
C5315.ex9	11751 18097	44267 0:00:16 32221 0:00:44	32877 0:00:36 32032 0:01:00	33783 0:01:02 33388 0:01:27	34041 00:01:03 33414 00:01:29	33782 0:01:02 33387 0:01:27	33783 0:01:02 33388 0:01:27	33783 0:01:00 33388 0:01:25	33807 0:01:01 33388 0:01:27	33807 0:01:00 33388 0:01:26	35197 0:02:19 32102 0:02:45	time out	127274 0:00:11 44089 0:01:30

	dyn3						dyn7					
	dyn3-gradual			dyn3-atonce			dyn7-gradual			dyn7-atonce		
	dyn3	gradual	ratio	dyn3	atonce	ratio	dyn7	gradual	ratio	dyn7	atonce	ratio
size	522644	1416318	0.37	373599	731376	0.51	523823	1416318	0.37	558947	1292319	0.43
run time	0:32:03	3:44:05	0.14	0:15:50	0:01:57	8.12	0:31:25	3:44:05	0.14	0:33:51	0:02:23	14.20
size (a.s.)	486689	540453	0.90	340820	400352	0.85	487149	540453	0.90	493744	580997	0.85
run time (w.s.)	0:41:56	3:59:51	0.17	0:21:01	0:11:37	1.81	0:41:13	3:59:51	0.17	0:43:11	0:20:11	2.14

Table 1: Experimental results for different solution strategies for MVO (characteristic functions of circuit clusters for functional simulation).

from [23] (columns greedy and atonce). For each example there are four lines in the table. The first line gives the size of the result as a shared BDD. The second line gives the run time for the algorithm (in format hours:minutes:seconds), the third line gives the BDD sizes after a final sifting step (if the algorithm does not fail due to "space out" or "time out") and the fourth line gives the total run time including sifting.

The "greedy at once" heuristic gives the smallest run times (if successful), but has a tendency to exceed the node limit. If it does finish, the BDD sizes are relatively large. In contrast, the "greedy gradual" heuristic is slow (there are many time outs). Also, even in the cases, when it does finish, BDD sizes are relatively large compared to our dyn$< n >$ approach even for smaller values of n. The dyn$< n >$ approach is able to provide a good trade–off between run time and quality. While for smaller values of n the run times are smaller, there are still cases, when the computation does not finish. For n equal to seven or larger all problems could be solved with a reasonable amount of runtime.

To confirm this analysis we summarize the results at the bottom of Table 1. We compare dyn3 and dyn7 to the "greedy gradual" heuristic and the "greedy at once" heuristic. In lines 1–4 we give the sums of the final BDD sizes, the run times, BDD sizes after sifting and total run times including sifting for all examples, for which both compared algorithms do not fail.

However, since the "greedy gradual" heuristic fails for 9 out of 14 examples and the "greedy at once" heuristic fails for 8 out of 14 examples, wheras dyn3 fails only for 3 examples and dyn7 does not fail for any example, we conclude that – in contrast to our dyn$< n >$ heuristic – both the "greedy gradual" heuristic and the "greedy at once" heuristic seem not to be suitable for this set of examples.

For a second experiment we have chosen pairs of benchmark circuits, for which BDDs were constructed and optimized separately. After that we transformed the BDDs into a common variable order. We used all those pairs of circuits from [23] which were at our disposal. Table 2 shows the results. As in Table 1, for each pair of circuits the first line gives the size of the result as a shared BDD, the second line gives the run time for the algorithm, the third line gives the BDD sizes after a final sifting step, and the fourth line gives the total run time including sifting.

	dyn0	dyn1	dyn2	dyn3	dyn5	dyn7	dyn10	dyn15	dyn20	dyn100	gradual	atonce
C1355/C3540	214548	174106	153335	158287	147200	138944	139500	154401	141189	146972	389563	809263
	0:01:06	0:01:36	0:02:51	0:02:40	0:04:58	0:03:36	0:03:37	0:03:30	0:03:35	0:04:30	0:20:46	0:00:46
	155391	144242	144856	145780	145260	131953	131427	150047	130955	142919	168145	159984
	0:04:14	0:04:04	0:05:08	0:05:00	0:07:16	0:05:38	0:05:34	0:05:58	0:05:38	0:06:48	0:24:26	0:07:08
C499/C1355	385108	217017	203498	171798	151366	154988	157947	165170	154553	146055	232696	483845
	0:01:59	0:01:38	0:02:20	0:03:00	0:03:17	0:05:30	0:05:04	0:04:54	0:05:35	0:03:05	0:18:13	0:00:46
	186691	174452	153742	145406	146239	139940	133187	143963	149848	133859	161276	225308
	0:06:01	0:04:59	0:05:24	0:05:27	0:05:48	0:07:47	0:07:16	0:07:14	0:08:19	0:05:31	0:21:37	0:05:54
i8/k2	3729	3781	4096	4356	3661	3646	3579	3615	4158	3667	5636	4646
	0:00:02	0:00:01	0:00:01	0:00:01	0:00:02	0:00:03	0:00:02	0:00:02	0:00:01	0:00:03	0:00:44	0:00:00
	3609	3592	3638	3616	3617	3578	3507	3556	3557	3529	3508	3350
	0:00:03	0:00:02	0:00:03	0:00:02	0:00:04	0:00:04	0:00:04	0:00:04	0:00:02	0:00:04	0:00:45	0:00:01
too_large/vda	1170	1182	1060	1060	1141	1141	1141	1141	1141	1141	7423	1509
	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:00	0:00:01	0:00:01	0:00:01	0:00:06	0:00:00
	1067	1067	1025	1025	1064	1064	1064	1064	1064	1064	1067	1097
	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:01	0:00:06	0:00:00
vda/alu4	1107	1104	1125	1115	1106	1106	1150	1104	1104	1130	1418	1394
	0:00:00	0:00:00	0:00:00	0:00:00	0:00:00	0:00:01	0:00:00	0:00:01	0:00:00	0:00:00	0:00:00	0:00:00
	1092	1092	1092	1087	1097	1097	1104	1096	1096	1097	1235	1093
	0:00:00	0:00:00	0:00:01	0:00:00	0:00:00	0:00:01	0:00:00	0:00:01	0:00:01	0:00:00	0:00:01	0:00:01

	dyn3	dyn7	gradual	atonce	dyn3-gradual ratio	dyn3-atonce ratio	dyn7-gradual ratio	dyn7-atonce ratio
size	336616	299825	636736	1300657	0.53	0.26	0.47	0.23
run time	0:05:42	0:09:11	0:39:49	0:01:32	0.14	3.72	0.23	5.99
size (a.s.)	296914	277632	335231	390832	0.89	0.76	0.83	0.71
run time (w.s.)	0:10:30	0:13:31	0:46:55	0:13:04	0.22	0.80	0.29	1.03

Table 2: Experimental results for different solution strategies for MVO (pairs of circuits).

Here all algorithms could finish all examples. Again, at the bottom of the table the results are summarized. The first line gives the sum of the BDD sizes, the second the sum of run times, the third line gives the sum of BDD sizes after sifting and, finally, the last line the sum of the total run times including sifting. In colums 2–5 these sums are given for dyn3, dyn7, "greedy gradual" and "greedy at once"[3]. In columns 6–9 we give the ratios dyn3 to "greedy gradual", dyn3 to "greedy at once", dyn7 to "greedy gradual" and dyn7 to "greedy at once". dyn3 and dyn7 provide considerable improvements both concerning size and run time compared to the "greedy gradual" heuristic. The "greedy at once" heuristic gives the best run times, but this is acheived at the cost of much larger BDDs. If we apply a final sifting step to optimize the variable orders of the results, the advantage of "greedy at once" with respect to run time is lost, because larger BDDs have to be sifted.

If we have a closer look at Table 2 we can observe again that the dyn$< n >$ is able to provide a good trade–off between run time and quality.

5 Conclusions

We presented a heuristic to solve the multiple variable order problem (MVO) for binary decision diagrams. In contrast to [23] we do not precompute a common variable order and transform the two BDDs into this variable order afterwards, rather we make use of dynamic reordering techniques. The adaption of the variable orders for the two BDDs proceeds step by step during the computation of the second BDD based on its cofactors. Experimental results prove our approach to be successful in solving the MVO problem. They also prove, that our approach defines a good trade–off between run time and quality of the result.

References

[1] S.B. Akers. Binary decision diagrams. *IEEE Trans. on Comp.*, 27:509–516, 1978.

[2] P. Ashar and S. Malik. Fast functional simulation using branching programs. In *Int'l Conf. on CAD*, pages 408–412, 1995.

[3] B. Bollig, M. Löbbing, and I. Wegener. Simulated annealing to improve variable orderings for OBDDs. In *Int'l Workshop on Logic Synth.*, pages 5b:5.1–5.10, 1995.

[4] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.

[5] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[3] The differences of sizes compared to [23] are apparently due to different initial variable orders for the circuits.

[6] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[7] R.E. Bryant. Symbolic Boolean manipulation with ordered binary decision diagrams. *ACM, Comp. Surveys*, 24:293–318, 1992.

[8] G. Cabodi, P. Camurati, and S. Quer. Improved reachability analysis of large finite state machines. In *Int'l Conf. on CAD*, pages 354–360, 1996.

[9] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. In *Int'l Workshop on Logic Synth.*, pages 5c:5.55–5.64, 1995.

[10] E. Felt, G York, R. Brayton, and A. Sangiovanni-Vincentelli. Dynamic Variable Reordering for BDD Minimization. In *European Design Automation Conf.*, pages 130–135, 1993.

[11] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 38–41, 1993.

[12] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *Int'l Conf. on CAD*, pages 2–5, 1988.

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

[14] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Int'l Conf. on CAD*, pages 472–475, 1991.

[15] C.Y. Lee. Representation of switching circuits by binary decision diagrams. *Bell System Technical Jour.*, 38:985–999, 1959.

[16] S. Malik, A.R. Wang, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[17] P.C. McGeer, K.L. McMillan, A. Saldanha, A.L. Sangiovanni-Vincentelli, and P. Scaglia. Fast discrete function evaluation using decision diagrams. In *Int'l Conf. on CAD*, pages 402–407, 1995.

[18] B.M.E. Moret. Decision trees and diagrams. In *Computing Surveys*, volume 14, pages 593–623, 1982.

[19] A. Narayan, A. Isles, J. Jain, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. Reachability analysis using partitioned-robdds. In *Int'l Conf. on CAD*, pages 388–393, 1997.

[20] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[21] C. Scholl, R. Drechsler, and B. Becker. Functional simulation using binary decision diagrams. In *Int'l Conf. on CAD*, pages 8–12, 1997.

[22] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

[23] C. Stangier, G. Cabodi, S. Quer, C. Meinel, Harald Sack, and A. Slobodová. Binary decision diagrams and the multiple variable order problem. In *Int'l Workshop on Logic Synth.*, pages 346–352, 1998.

[24] S. Tani, K. Hamaguchi, and S. Yajima. *The Complexity of the Optimal Variable Ordering Problem of Shared Binary Decision Diagrams*, volume 762 of *LNCS*. Proc. ISAAC'93, 1993.