Minimizing ROBDD Sizes of Incompletely Specified Boolean Functions by Exploiting Strong Symmetries *

Christoph Scholl† Stephan Melchior‡ Günter Hotz‡ Paul Molitor§

†Institute of Computer Science
Albert-Ludwigs-University
D 79110 Freiburg im Breisgau, FRG
(scholl@informatik.uni-freiburg.de)

‡Department of Computer Science
Universität des Saarlandes
D 66041 Saarbrücken, FRG
(<name>@cs.uni-sb.de)

§Institute of Computer Science
University of Halle
D 06099 Halle (Saale), FRG
(molitor@informatik.uni-halle.de)

Abstract

We present a method computing a minimum sized partition of the variables of an incompletely specified Boolean function into symmetric groups. The method can be used during minimization of ROBDDs *of incompletely specified Boolean functions. We apply it as a preprocessing step of symmetric sifting presented by Panda [24] and Möller [20] and of techniques for* ROBDD *minimization of incompletely specified Boolean functions presented by Chang [6] and Shiple [28]. The technique is shown to be very effective: it improves* ROBDD *sizes of symmetric sifting by a factor of 51% and by a factor of 70% in combination with a slightly modified version of the technique of Chang and Shiple.*

1 Introduction

Binary Decision Diagrams (BDDs) as a data structure for representation of Boolean functions were first introduced by Lee [17] and further popularized by Akers [1] and Moret [21]. In the restricted form of reduced ordered BDDs (ROBDDs) they gained widespread application because ROBDDs are a canonical representation and allow efficient manipulations [4]. Some fields of application are logic verification, test generation, fault simulation, and logic synthesis [18, 5]. Most of the algorithms using ROBDDs have running time polynomial in the size of the ROBDDs. The sizes depend on the variable order used.

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 [18, 11, 12] and dynamic reordering heuristics which try to improve on a given order [14, 25, 10, 2, 9]. Sifting introduced by Rudell [25] has emerged so far as the most successful algorithm for dynamic reordering of variables. This algorithm is based on finding the 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. As already observed in [23], one limitation of sifting, however, 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.

Recently, it has been shown in [20] and [24] that symmetry properties can be used to efficiently construct good variable orders for ROBDDs using modified gradual improvement heuristics. The crucial point is to locate the symmetric variables side by side and to treat them as a fixed block. This results in *symmetric sifting* which sifts symmetric groups simultaneously*. Regular sifting usually puts symmetric variables together in the order, but the symmetric groups tend to be in sub-optimal positions. The sub-optimal solutions result from the fact that regular sifting is unable to recognize that the variables of a symmetric group have a strong attraction to each other and should be sifted together. When a variable of a symmetric group is sifted by regular sifting, it is likely to return to its initial position due to the attraction of the other variables of the group [23].

The papers mentioned above only handle completely specified functions. But in many applications (e.g. checking the equivalence of two finite state machines (FSMs) [7], minimizing the transition relation of an FSM or logic synthesis for FPGA realizations [16, 31, 27]) incompletely specified Boolean functions play an important role. In applications where ROBDD sizes have a large influence on the quality of the results (such as logic synthesis for FPGA realizations) there is a strong need for ROBDD minimization techniques for incompletely specified functions.

To the best of our knowledge, no variable ordering algorithm exploiting don't cares has been presented in literature. A couple of papers, e.g., [6] and [28] investigate the ROBDD minimization problem for incompletely specified Boolean functions. They start with a fixed variable order obtained by any ordering heuristics and greedily minimize the number of nodes at every level by assigning as few don't cares as possible to either the on-set or the off-set. The variable order remains fixed during this process. However, the resulting ROBDD sizes heavily depend on the variable order. Thus, there is a need to determine good variable orders in the case of incompletely specified functions, too.

*This work was supported in part by the Graduiertenkolleg of the Universität des Saarlandes and DFG grant Mo 645/2-1

*Symmetric sifting is very efficient but does not result in optimal orders in any case as proven in [20] and [29]

As determining the symmetric groups before applying sifting has been proven to result in good variable orders for completely specified functions, it seems to be a good idea in the case of incompletely specified functions to first determine symmetric groups, then to apply symmetric sifting and techniques as those from [6, 28]. However, the symmetric groups of incompletely specified functions are not uniquely defined (see Section 3). Therefore we have the problem to compute good partitions into symmetric groups with respect to ROBDD minimization.

In [15] an algorithm is presented, which decides for an incompletely specified Boolean function (represented by a cube array), whether a given set λ of input variables forms a symmetric group or not. However, for our problem to partition the input variables into symmetric groups there remain two difficulties: first the question, how to find large candidate sets λ (of course, we cannot test for each subset of the variables whether it is a symmetric group) and secondly the question, how to combine symmetric groups to a partition of the input variables, such that the incompletely specified function is symmetric in each set of the partition *at the same time* (in Section 3 we will show that this cannot be done in a straightforward manner). To the best of our knowledge, no technique has been developed so far that targets on computing minimal partitions into symmetric groups for incompletely specified functions.

The paper is structured as follows. In Section 2 we briefly review the definitions of symmetric groups of completely and incompletely specified Boolean functions. Section 3 studies the difficulties with symmetry of incompletely specified functions. To overcome these difficulties we introduce *strong symmetry of incompletely specified functions* in Section 4. We then concentrate on computing a minimum sized partition of the variables of incompletely specified functions into symmetric groups exploiting strong symmetries in Section 5. We adjust a greedy algorithm for node coloring to heuristically solve our problem. The paper closes with experimental results proving our method to be very effective. It improves ROBDD sizes of symmetric sifting by a factor of 51% and by a factor of 70% in combination with a slightly modified version of Chang's technique [6].

2 Symmetric groups

In the following, let X be the set of variables $\{x_1, \ldots, x_n\}$ of a Boolean function f and D some subset of $\{0,1\}^n$.

2.1 Completely specified functions

In this section we will briefly review definitions and basic properties of symmetries of completely specified Boolean functions. We start with the definition of symmetry in two variables, in a set of variables, and in a partition of the set of input variables of a completely specified Boolean function.

Definition 1 *A completely specified Boolean function $f : \{0,1\}^n \to \{0,1\}$ is symmetric in a pair of input variables (x_i, x_j) if and only if $f(\epsilon_1, \ldots, \epsilon_i, \ldots, \epsilon_j, \ldots, \epsilon_n) = f(\epsilon_1, \ldots, \epsilon_j, \ldots, \epsilon_i, \ldots, \epsilon_n)$ holds $\forall \epsilon \in \{0,1\}^n$. f is symmetric in a subset λ of X iff f is symmetric in x_i and x_j $\forall x_i, x_j \in \lambda$. f is symmetric in a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ of the set of input variables iff f is symmetric in λ_i $\forall 1 \leq i \leq k$.*

If f is symmetric in a subset λ of the set of input variables, then we say 'the variables in λ form a symmetric group'.

It is well–known, that symmetry of a completely specified Boolean function f in pairs of input variables of f leads to an equivalence relation on X. Thus, there is a unique minimal partition P of X (namely the set of the equivalence classes of this relation) such that f is symmetric in P. The computation of a minimal partition P such that f is symmetric in P can be done by testing for symmetry in all pairs of input variables [19, 30].

2.2 Incompletely specified functions

The definition of symmetry of an incompletely specified Boolean function f is reduced to the definition of symmetry of completely specified extensions of f. An extension of an incompletely specified Boolean function is defined as follows:

Definition 2 *Let $f : D \to \{0,1\}$ $(D \subseteq \{0,1\}^n)$ be an incompletely specified Boolean function. $f' : D' \to \{0,1\}$ $(D' \subseteq \{0,1\}^n)$ is an extension of f iff $D \subseteq D'$ and $f'(\epsilon) = f(\epsilon)$ $\forall \epsilon \in D$.*

Definition 3 *An incompletely specified Boolean function $f : D \to \{0,1\}$ is symmetric in a pair of input variables (x_i, x_j) (in a subset λ of X / in a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ of X) iff there is a completely specified extension f' of f, which is symmetric in (x_i, x_j) (in λ / in P).*

3 Difficulties with symmetry of incompletely specified functions

In order to minimize the ROBDD size for an incompletely specified Boolean function f, we are looking for a minimal partition (or for maximal variable sets) such that f is symmetric in this partition (or these sets). Unfortunately there are some difficulties in the computation of such partitions: First of all, symmetry of f in two variables doesn't form an equivalence relation on X in the case of *incompletely* specified Boolean functions (see [8] or [15]).

Since symmetry in pairs of variables doesn't form an equivalence relation, it will be much more difficult to deduce symmetries in larger variable sets from symmetries in pairs of variables in the case of incompletely specified Boolean functions.

Even if f is symmetric in *all* pairs of variables x_i and x_j of a subset λ of the variable set of f, f is *not* necessarily symmetric in λ. This is illustrated by the following example:

Example 1 Consider $f : D \to \{0,1\}$, $D \subseteq \{0,1\}^4$.

$$f(\epsilon) = \begin{cases} 1 & \text{for } \epsilon = (0,0,1,1) \\ dc & \text{for } \epsilon = (0,1,0,1), \epsilon = (0,1,1,0), \\ & \quad \epsilon = (1,0,0,1), \epsilon = (1,0,1,0) \\ 0 & \text{for } \epsilon = (1,1,0,0) \\ 0 & \text{otherwise} \end{cases}$$

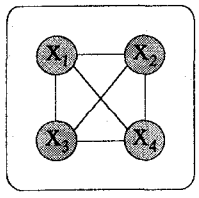

Figure 1: Symmetry graph of the function of Example 2

It is easy to see, that f is symmetric in all pairs of variables x_i and x_j, $i,j \in \{1,2,3,4\}$. The symmetry graph[†] of f is shown in Figure 1. It is a complete graph. For each completely specified extension f' of f, which is symmetric in (x_1,x_3), $f'(0,1,1,0) = 0$ holds and for each completely specified extension f'' of f, which is symmetric in (x_2,x_4), $f''(0,1,1,0) = 1$ holds. Hence there is no completely specified extension of f which is symmetric in (x_1,x_3) *and* (x_2,x_4) and therefore no extension which is symmetric in $\{x_1,\ldots,x_4\}$.

Example 1 also points out another fact: If an incompletely specified Boolean function f is symmetric in all variable sets λ_i of a partition $P = \{\lambda_1,\ldots,\lambda_k\}$, it is *not* necessarily symmetric in P (choose $P = \{\{x_1,x_3\},\{x_2,x_4\}\}$ in the example).

4 Strong symmetry

The difficulties with the detection of large symmetry groups of incompletely specified functions result from the fact that symmetry in pairs of variables doesn't form an equivalence relation on the variable set X. If we change the definition of symmetry of incompletely specified functions as given in Definition 4, symmetry in pairs of variables provides an equivalence relation as in the case of completely specified functions:

Definition 4 (Strong symmetry) *An incompletely specified Boolean function $f : D \to \{0,1\}$ is called* strongly symmetric *in a pair of input variables (x_i,x_j) iff $\forall(\epsilon_1,\ldots,\epsilon_n) \in \{0,1\}^n$ either (a) or (b) holds.*

(a) $(\epsilon_1,\ldots,\epsilon_i,\ldots,\epsilon_j,\ldots,\epsilon_n) \notin D$ and
$(\epsilon_1,\ldots,\epsilon_j,\ldots,\epsilon_i,\ldots,\epsilon_n) \notin D$

(b) $(\epsilon_1,\ldots,\epsilon_i,\ldots,\epsilon_j,\ldots,\epsilon_n) \in D$ and
$(\epsilon_1,\ldots,\epsilon_j,\ldots,\epsilon_i,\ldots,\epsilon_n) \in D$
and $f(\epsilon_1,\ldots,\epsilon_i,\ldots,\epsilon_j,\ldots,\epsilon_n)$
$= f(\epsilon_1,\ldots,\epsilon_j,\ldots,\epsilon_i,\ldots,\epsilon_n)$.

In contrast to the *strong* symmetry of incompletely specified functions the symmetry defined so far is called *weak* symmetry.

The following lemma holds for strong symmetry:

[†]The symmetry graph $G^f_{sym} = (X,E)$ of a Boolean function $f : D \to \{0,1\}$ is a undirected graph with node set X (the set of input variables of f) and edges $\{x_i,x_j\} \in E$ iff f is symmetric in (x_i,x_j).

Lemma 1 *Strong symmetry in pairs of variables of an incompletely specified Boolean function $f : D \to \{0,1\}$ forms an equivalence relation on the variable set X of f.*

Due to Lemma 1 there is a unique minimal partition P of the set X of input variables such that f is strongly symmetric in P. As in the case of completely specified Boolean functions, f is strongly symmetric in a subset λ of X iff $\forall x_i,x_j \in \lambda$ f is strongly symmetric in (x_i,x_j). f is strongly symmetric in a partition $P = \{\lambda_1,\ldots,\lambda_k\}$ of X iff $\forall 1 \le i \le k$ f is strongly symmetric in λ_i.

Of course, if a function f is weakly symmetric in a partition P, it needs not to be strongly symmetric in P, but it follows directly from Definition 3 that there is an extension of f which is strongly symmetric in P.

5 Minimum sized partition of the variables of an incompletely specified function into symmetric groups

We have to solve the following problem MSP (Minimal Symmetry Partition):

Given: Incompletely specified function $f : D \to \{0,1\}$, represented by ROBDDs for f_{on} and f_{dc}. [‡]

Find: Partition P of the set $X = \{x_1,\ldots,x_n\}$ such that

- f is symmetric in P and
- for any partition P' of X in which f is symmetric, the inequation $|P| \le |P'|$ holds.

We can prove the following theorem [26]:

Theorem 1 *MSP is NP-hard.*

To solve the problem heuristically, we use a heuristic for the problem 'Partition into Cliques (PC)' [13] for the symmetry graph G^f_{sym} of f. However, the examples in Section 3 showed that f is *not* symmetric in *all* partitions into cliques of G^f_{sym}. The heuristic has to be changed in order to guarantee that f is symmetric in the resulting partition P.

The heuristic to solve the problem PC makes use of the following well–known lemma:

Lemma 2 *A graph $G = (V,E)$ can be partitioned into k disjoint cliques iff $\bar{G} = (V,\bar{E})$ can be colored with k colors. ($\bar{G}$ is the inverse graph of G, which has the same node set V as G and an edge $\{v,w\}$ between two nodes v and w iff there is no edge $\{v,w\}$ in G, i.e., $\bar{E} = \{\{v,w\} | \{v,w\} \notin E\}$.)*

Thus, heuristics for node coloring can be directly used for the solution of partition into cliques. Nodes with the same color in $\bar{G}$ form an 'independent set' and thus a clique in G. Our implementation is based

[‡]f_{on} is the completely specified Boolean function with the same on-set as f and f_{dc} is the completely specified function with $\{0,1\}^n \setminus D$ as on-set.

Input: Incompletely specified function $f : D \to \{0,1\}$, $D \subseteq \{0,1\}^n$, represented by f_{on} and f_{dc}
Output: Partition P of $\{x_1, \ldots, x_n\}$, such that f is symmetric in P
Algorithm:

```
1  Compute symmetry graph G^f_sym = (V, E) of f (or the complement G^f_sym = (V, Ē)).
2  ∀1 ≤ k ≤ n : color(x_k) := undef.
3  P = {{x_1}, {x_2}, ..., {x_n}}
4  node_candidate_set := {x_1, ..., x_n}
5  while (node_candidate_set ≠ ∅) do
6        /* f is strongly symmetric in P */
7        Choose x_i ∈ node_candidate_set according to Brélaz/Morgenstern criterion
8        color_candidate_set := {c | 1 ≤ c ≤ n, ∄x_j with {x_i, x_j} ∈ Ē and color(x_j) = c}
9        while (color(x_i) = undef.) do
10             curr_color := min(color_candidate_set)
11             color(x_i) := curr_color
12             if (∃ colored node x_j with color(x_j) = color(x_i))
13               then
14                     if (f symmetric in (x_i, x_j))
15                       then
16                             P := P \ {[x_j], {x_i}} ⋃ {[x_j] ∪ {x_i}}
17                             /* f is symmetric in P */                    (*)
18                             Make f strongly symmetric in P.              (**)
19                       else
20                             color_candidate_set := color_candidate_set \ {curr_color}
21                             color(x_i) := undef.
22                     fi
23             fi
24       od
25       node_candidate_set := node_candidate_set \ {x_i}
26 od
```

Figure 2: Algorithm to solve MSP.

on Brélaz algorithm for node coloring [3] which has a running time of $\mathcal{O}(N)$ in an implementation of Morgenstern [22] (N is the number of nodes of the graph which has to be colored). It is a greedy algorithm, which colors node by node and doesn't change the color of a node which is already colored. In the algorithm there are certain criteria to choose the next node to color and the color to use for it in a clever way [3, 22]. Figure 2 shows our heuristic for the problem MSP, which is derived from the Brélaz/Morgenstern heuristic for node coloring.

First of all the symmetry graph G^f_{sym} of f (or the inverse graph $\overline{G^f_{sym}}$) is computed. The nodes of $\overline{G^f_{sym}}$ are the variables $x_1, \ldots, x_n$. These nodes are colored in the algorithm. Nodes with the same color form a clique in G^f_{sym}. Note that partition P (see line 3) has the property that it contains set $\{x_k\}$ for any uncolored node x_k and that nodes with the same color are in the same set of P, at any moment. The crucial point of the algorithm is that the invariant 'f is strongly symmetric in P' of line 6 is always maintained.

Now let us take a look at the algorithm in more detail. At first glance, the set of all admissible colors for the next node x_i is the set of all colors between 1 and n except the colors of nodes which are adjacent to x_i in $\overline{G^f_{sym}}$. In the original Brélaz/Morgenstern algorithm the minimal color among these colors is chosen for x_i (*curr_color* in lines 10, 11). However, since we have to guarantee that f is symmetric in the partition P which results from coloring, it is possible that we are not allowed to color x_i with *curr_color*. If there is already another node x_j which is colored by *curr_color*, then f has to be symmetric in the partition P' which results by union of $\{x_i\}$ and $[x_j]$[§]. If there is such a node x_j, we have to test whether f is symmetric in (x_i, x_j) (line 14) (this test can have a negative result, since the don't care set of f is reduced during the algorithm). If f is not symmetric in (x_i, x_j), *curr_color* is removed from the set of color candidates for x_i (line 20) and the minimal color in the remaining set is chosen as the new color candidate (line 10). If the condition of line 14 is true, the new partition P results from the old partition P by union of $\{x_i\}$ and $[x_j]$ (line 16). Now f is symmetric in the new partition P (invariant (*) from line 17, see Lemma 3), and we can assign don't cares of f such that f is strongly symmetric in P (line 18).

At the end we receive an extension of the original incompletely specified Boolean function which is strongly symmetric in the resulting partition P.

To prove invariant (*) in line 17, we need the following lemma [26]:

Lemma 3 *Let $f : D \to \{0,1\}$ be* strongly symmetric *in P, $[x_i], [x_j] \in P$ two subsets with $|[x_i]| = 1$, and let f be symmetric in (x_i, x_j), then f is symmetric in $P' = P \setminus \{[x_j], \{x_i\}\} \bigcup \{[x_j] \cup \{x_i\}\}$.*

Note that the lemma cannot be proved if we replace 'f *strongly* symmetric in P' by 'f (weakly) symmetric in P' or if we don't assume $|[x_i]| = 1$. However

[§]If $P = \{\lambda_1, \ldots, \lambda_k\}$ is a partition of $\{x_1, \ldots, x_n\}$, then $[x_j]$ denotes λ_q with $x_j \in \lambda_q$.

Procedure *make_strongly_symm*
Input: $f : D \rightarrow \{0,1\}$, represented by f_{on}, f_{off}, f_{dc}. f is (weakly) symmetric in (x_i, x_j).
Output: minimal extension f' of f (represented by f'_{on}, f'_{off}, f'_{dc}), which is strongly symmetric in (x_i, x_j).
Algorithm:

1. $f'_{on} = \overline{x_i}\,\overline{x_j} f_{on_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{on_{x_i x_j}} + (x_i\overline{x_j} + \overline{x_i}x_j)(f_{on_{x_i\overline{x_j}}} + f_{on_{\overline{x_i}x_j}})$
2. $f'_{off} = \overline{x_i}\,\overline{x_j} f_{off_{\overline{x_i}\,\overline{x_j}}} + x_i x_j f_{off_{x_i x_j}} + (x_i\overline{x_j} + \overline{x_i}x_j)(f_{off_{x_i\overline{x_j}}} + f_{off_{\overline{x_i}x_j}})$
3. $f'_{dc} = \overline{f'_{on} + f'_{off}}$

Figure 3: Procedure *make_strongly_symm*

the given conditions coincide exactly with the conditions existing in the algorithm. (Thus it is *necessary* to make f *strongly* symmetric in P in line 18 of the algorithm and to maintain the invariant 'f is strongly symmetric in P' of line 6.)

Next we have to explain how f is made strongly symmetric in the partition P in line 18 of the algorithm. From the definition of symmetry of incompletely specified functions it is clear that it is possible to extend a function f, which is (weakly) symmetric in a partition P, to a function which is strongly symmetric in P. From the set of all extensions of f which are strongly symmetric in P we choose the extension with a maximum number of don't cares. If f is (weakly) symmetric in a pair of variables (x_i, x_j), the extension f' of f, which is strongly symmetric in (x_i, x_j) and which has a maximal don't care set among all extensions of f with that property, can be easily computed from the ROBDD representations of f_{on}, f_{dc} and f_{off} by the procedure *make_strongly_symm* in Figure 3. We can prove that a sequence of at most $||[x_j]||$ calls of the procedure *make_strongly_symm* is enough to make f strongly symmetric in the partition P in line 18 of the algorithm.

6 Experimental results

We have carried out experiments to test the algorithms described above. To generate incompletely specified functions from completely specified functions, we used a method proposed in [6]: After collapsing each benchmark circuit to two level form, we randomly selected minterms in the on-set with a probability of 40% to be included into the don't care set. The last three Boolean functions in Table 1 are partial multipliers $partmult_n$[¶].

We performed three experiments: First of all, we applied symmetric sifting to the ROBDDs representing the on-set of each function. The results are shown in column 5 (*sym_sift*) of Table 1. The entries are ROBDD sizes in terms of internal nodes.

[¶]The n^2 inputs are the bits of the n partial products and the $2n$ outputs are the product bits. The don't care set contains all input vectors which cannot occur for the reason that the input bits are not independent from each other, because they are conjunctions $a_i b_j$ of bits of the operands $(a_1, \ldots, a_n)$ and $(b_1, \ldots, b_n)$ of the multiplication.

In a second experiment, we applied our algorithm to minimize the number of symmetric groups followed by symmetric sifting. Column 6 (*sym_group*) of Table 1 shows the results. *sym_group* provides a partition $P = \{\lambda_1, \ldots, \lambda_k\}$ and an extension f' of the original function f, such that f' is strongly symmetric in P. The variable order of the ROBDD representing f' is a 'symmetric order' [24, 20] with the variables in λ_i before the variables in λ_{i+1} $(1 \leq i < k)$. On the average, we can improve the ROBDD size by 51%.

In a last experiment we started with the results of *sym_group* and then went on with a slightly modified version of the technique of Chang [6] and Shiple [28]. This technique minimizes the number of nodes at every level of the ROBDD by an operation *remove_z* assigning as few don't cares as possible to either the on-set or the off-set, i.e., the number of so-called linking nodes immediately below a cut line between 2 adjacent variables is minimized. After the minimization of nodes at a certain level of the ROBDD they use the remaining don't cares to minimize the number of nodes at the next level. The cut line is moved from top to bottom in the ROBDD. Under certain conditions, this method does preserve strong symmetry: Let f be an incompletely specified Boolean function which is strongly symmetric in $P = \{\lambda_1, \ldots, \lambda_k\}$ and assume that the variable order of the ROBDD representing f is a 'symmetric order' with the variables in λ_i before the variables in λ_{i+1} $(1 \leq i < k)$. If we restrict the operation *remove_z* presented in [6] to cut lines between 2 symmetric groups λ_i and λ_{i+1}, then it preserves strong symmetry in P. Since our technique to restrict *remove_z* to cut lines between symmetric groups doesn't destroy the symmetric groups, we can perform symmetric sifting after the node minimization with the same symmetric groups as before. Column 7 (*sym_cover*) of Table 1 shows the resulting ROBDD sizes. On the average, the technique leads to an improvement of the ROBDD sizes by 70%.

A comparison to the results of the *restrict* operator [7] (applied to ROBDDs whose variable order was optimized by regular *sifting*) in column 4 of Table 1 shows that our ROBDD sizes are on the avarage 44% smaller.

7 Conclusions

In this paper we presented algorithms to minimize ROBDD sizes for incompletely specified Boolean functions based on the exploitation of strong symmetries. Incompletely specified Boolean functions which are represented by ROBDDs play an important role in many applications of CAD. Looking for extensions of such functions with small ROBDD representations can have a large effect on the quality of the results (e.g. in logic synthesis for FPGA realizations where there is a direct relationship between ROBDD sizes and the number of CLBs needed to realize the function). Experimental results prove our approach to be very effective.

References

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

[2] 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.

Circuit	i	o	*restrict*	*sym_sift*	*sym_group*	*sym_cover*
5xp1	7	10	63	67	66 (0.2 s)	53 (0.5 s)
9symml	9	1	67	108	25 (0.3 s)	25 (0.4 s)
alu2	10	6	192	201	201 (0.7 s)	152 (2.6 s)
apex6	135	99	993	1033	983 (267.6 s)	612 (459.7 s)
apex7	49	37	730	814	728 (27.7 s)	340 (52.2 s)
b9	41	21	213	256	185 (8.6 s)	122 (11.5 s)
c8	28	18	110	156	95 (1.7 s)	70 (3.2 s)
example2	85	66	497	491	484 (69.2 s)	416 (119.4 s)
mux	21	1	32	34	29 (0.6 s)	29 (0.7 s)
pcler8	27	17	111	78	73 (1.9 s)	72 (3.3 s)
rd73	7	3	75	76	34 (0.3 s)	27 (0.4 s)
rd84	8	4	135	144	42 (0.7 s)	42 (0.7 s)
sao2	10	4	89	104	104 (0.4 s)	70 (0.8 s)
x4	94	71	814	829	633 (121.9 s)	485 (203.4 s)
z4ml	7	4	47	51	32 (0.2 s)	17 (0.3 s)
partmult3	9	6	70	152	35 (1.0 s)	29 (1.2 s)
partmult4	16	8	307	971	222 (49.5 s)	114 (50.6 s)
partmult5	25	10	857	4574	998 (1540.4 s)	365 (1548.4 s)
total			5402	10139	4969	3040

Table 1: Experimental results. The table shows the number of nodes in the ROBDDs of each function. Numbers in parenthesis show the CPU times (measured on a SPARCstation 20 (96 MByte RAM)).

[3] D. Brélaz. New methods to color vertices of a graph. *Comm. of the ACM*, 22:251–256, 1979.

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

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

[6] S. Chang, D. Cheng, and M. Marek-Sadowska. Minimizing ROBDD size of incompletely specified multiple output functions. In *European Design & Test Conf.*, pages 620–624, 1994.

[7] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 365–373, 1989.

[8] D.L. Dietmeyer and P.R. Schneider. Identification of symmetry, redundancy and equivalence of boolean functions. *IEEE Trans. on Electronic Comp.*, 16:804–817, 1967.

[9] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. In *Int'l Workshop on Logic Synth.*, pages P5c: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] 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.

[12] 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.

[13] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to NP-Completeness.* Freemann, San Francisco, 1979.

[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] B.-G. Kim and D.L. Dietmeyer. Multilevel logic synthesis of symmetric switching functions. *IEEE Trans. on CAD*, 10(4), 1991.

[16] Y.-T. Lai, M. Pedram, and S.B.K. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Design Automation Conf.*, pages 642–647, 1993.

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

[18] 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.

[19] D. Möller, J. Mohnke, and M. Weber. Detection of symmetry of boolean functions represented as robdds. In *Int'l Conf. on CAD*, pages 680–684, 1993.

[20] D. Möller, P. Molitor, and R. Drechsler. Symmetry based variable ordering for ROBDDs. *IFIP Workshop on Logic and Architecture Synthesis, Grenoble*, pages 47–53, 1994.

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

[22] C. Morgenstern. A new backtracking heuristic for rapidly four-coloring large planar graphs. Technical Report CoSc-1992-2, Texas Christian University, Fort Worth, Texas, 1992.

[23] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

[24] S. Panda, F. Somenzi, and B.F. Plessier. Symmetry detection and dynamic variable ordering of decision diagrams. In *Int'l Conf. on CAD*, pages 628–631, 1994.

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

[26] C. Scholl. *Mehrstufige Logiksynthese unter Ausnutzung funktionaler Eigenschaften.* PhD thesis, Universität des Saarlandes, 1996.

[27] C. Scholl and P. Molitor. Communication Based FPGA Synthesis for Multi-Output Boolean Functions. In *ASP Design Automation Conf.*, pages 279–287, 1995.

[28] T.R. Shiple, R. Hojati, A.L. Sangiovanni-Vincentelli, and R.K. Brayton. Heuristic minimization of BDDs using don't cares. In *Design Automation Conf.*, pages 225–231, 1994.

[29] D. Sieling. Variable orderings and the size of OBDDs for partially symmetric boolean functions. In *SASIMI*, pages 189–196, 1996.

[30] C.C. Tsai and M. Marek-Sadowska. Generalized reed-muller forms as a tool to detect symmetries. *IEEE Trans. on Comp.*, 45:33–40, 1996.

[31] B. Wurth, K. Eckl, and K. Antreich. Functional multiple-output decomposition: Theory and implicit algorithm. In *Design Automation Conf.*, pages 54–59, 1995.