SBDD Variable Reordering Based on Probabilistic and Evolutionary Algorithms *

M. A. Thornton
Mississippi State University
J. P. Williams
NCR Corporation
R. Drechsler, N. Drechsler
University of Freiburg, Germany
D. M. Wessels
University of Arkansas

Abstract

Modern CAD tools must represent large Boolean functions compactly in order to obtain reasonable runtimes for synthesis and verification. The Shared Binary Decision Diagram (SBDD) with negative edge attributes can represent many functions in a compact form if a proper variable ordering is used. In this work we describe a technique for reordering the variables in an SBDD to reduce the size of the data structure. A common heuristic for the variable ordering problem is to group variables together that have similar characteristics. We use this heuristic to formulate a technique for the reordering problem using probability based metrics. Our results indicate that this technique outperforms sifting with comparable runtimes. Furthermore, the method is robust in that the final result is independent of the initial structure of the SBDD.

1 Introduction

The use of binary decision programs to represent switching circuits was proposed by Lee in the late 1950's [12] and later refined as a representation of Boolean functions by Akers [1] resulting in the Binary Decision Diagram (BDD) concept. Bryant further refined the definition of BDDs and created a new data structure called Ordered Binary Decision Diagrams (OBDD) [5]. The OBDD representation requires that any given path may contain a vertex representing a Boolean variable at most once, and the order in which the variables may be encountered is the same for all paths. For a given variable ordering, OBDDs are canonical representations of Boolean functions.

A further development was the Shared Binary Decision Diagram (SBDD) with negative edge-attributes [15] [4]. This data structure utilizes the principle of variable ordering as developed in [5] and also allows for the representation of multi-output functions through shared common subgraphs. The incorporation of negative edge-attributes allows for a reduction in the size of most SBDDs. In this work, we utilize the negative edge-attributed SBDD for function representation.

Although SBDDs can represent Boolean functions using a relatively small amount of storage space, a proper variable ordering is crucial. If a poor variable ordering is chosen, the SBDD may become quite large, with a worst case requirement of an exponential amount of memory in terms of the function variables. In fact, it has been shown that for some circuits (e.g., integer addition circuits) the size of the OBDD representing the function may vary from linear to exponential depending on the variable ordering [5] [8]. Therefore, methods to determine efficient orderings are needed to utilize the computational advantages of SBDDs. Furthermore, it is known that improving the variable ordering of SBDDs is NP-complete and finding the optimal form is NP-hard [3]. Since the problem is provably intractable, heuristic methods are applied to find a *good* but not necessarily *best* solution. Fortunately, it has been found that for many functions of interest, good variable orderings produce SBDDs of acceptable size. This reasoning provides the motivation for the investigation of new variable reordering heuristics.

In this paper we describe a static variable ordering heuristic that makes use of properties of the function only, but is not dependent on the initial representation. We utilize a method that was motivated by grouping symmetric variables close to one another in the SBDD ordering. Since we can compute output probability values efficiently on SBDDs, we use the quantities to indicate the possible existence of a symmetry relation. After analysis, we found that grouping the variables close together that were likely to be symmetric did not always result in the best sizes. Alternatively, we found that spacing these variables as far apart from one another as possible often resulted in smaller SBDDs. As an alternative to this fast heuristic we also present a new simulation based approach based on *Evolutionary Algorithms*

*This project was supported by NSF grants CCR-9633085, SBE-9815371 and DAAD grant 315/PPP/gü-ab.

(EAs) that yields very good minimized SBDDs but can require larger amounts of computation time.

The remainder of the paper is structured as follows. In section 2 the problem domain is formulated and the probabilistic based heuristic is described. Next, in Section 3, the evolutionary minimizer for optimizing BDD variable orders is presented. Experimental results are given in Section 4 and the paper is summarized in Section 5.

2 Heuristic Description and Motivation

Many researchers have utilized heuristics that measure the relationship between primary inputs. For example, techniques based on topological closeness were among the first variable ordering methods [13] [7]. Techniques that exploit relationships between dependent variables continued to be popular particularly after it was noted that the application of sifting [20] led to orderings where symmetric variables tended to cluster together [16] [18].

The determination of symmetry among variables is not a computationally efficient process to undertake by pure manipulation of SBDD structures. Therefore an alternative method based on circuit output probabilities is formulated here to utilize symmetry indicators for the purpose of reordering.

2.1 Output Probability

The output probability of a Boolean function expresses the probability that the function will evaluate to a logic "1" value given the probability distributions of the dependent variables [19]. When the function is fully specified and all variables are equally likely to be 0 or 1, the output probability is simply the percentage of minterms that cause the function to evaluate to logic "1". We denote the output probability of a function f as $\wp\{f\}$. As an example, $\wp\{x_1 + x_2 + x_3\} = \frac{7}{8}$.

The output probability of a function may be computed in time linear with respect to the size of a SBDD. Various algorithms have been developed that have complexity proportional to the number of vertices in the BDD [6] [11] and they are applicable to SBDDs with negative-edge attributes [14].

Since we are interested in using symmetry relations among the variables of a Boolean function, the relationship in Theorem 1 provides the connection between variable symmetry and output probabilities.

Theorem 1 *$\wp\{f \oplus x_i\} = \wp\{f \oplus x_j\}$ if $x_i, x_j \in S$ where S is the set of all independent variables that support f and $x_i \leftrightarrow x_j$ (that is, x_i and x_j are symmetric in f).*

Proof: From the Shannon expansion theorem we have:

$f = \overline{x}_i \cdot f_{\overline{x}_i} + x_i \cdot f_{x_i}$

Using the properties of probability theory and assuming that x_i and x_j are statistically independent, temporally and spatially uncorrelated and equally likely to be "0" or "1" (i.e. $\wp\{x_i\} = \wp\{x_j\} = \frac{1}{2}$):

$\wp\{f_{x_i}\} = \wp\{f|x_i\} = \frac{\wp\{f \cdot x_i\}}{\wp\{x_i\}} = 2\wp\{f \cdot x_i\}$

Substituting this result into the output probability relationship for the logical XOR of a function with its' i^{th} dependent variable yields:

$\wp\{f \oplus x_i\} = \wp\{f\} + \wp\{x_i\} - 2\wp\{f \cdot x_i\}$

$\wp\{f \oplus x_i\} = \wp\{f\} + \wp\{x_i\} - \wp\{f_{x_i}\}$

Clearly, $x_i \leftrightarrow x_j$ implies that $\wp\{f_{x_i}\} = \wp\{f_{x_j}\}$. Thus,

$\wp\{f\} + \wp\{x_i\} - \wp\{f_{x_i}\} = \wp\{f\} + \wp\{x_j\} - \wp\{f_{x_j}\}$

This relationship can then be rewritten as:

$\wp\{f \oplus x_i\} = \wp\{f \oplus x_j\}$ □

The output probabilities described here may be computed using algorithms that perform single traversals of SBDDs [6] [14]. Thus, it is practical to compute the n probability values of $\wp\{f \oplus x_i\}$ for all $x_i \in S$.

2.2 Relationship of Variable Order and Output Probability

During the formulation of this technique, we generated plots of $\wp\{f \oplus x_i\}$ versus the dependent variable index i for the best known orderings of several benchmark circuits in SBDD form with negative edge attributes. Due to the heuristic based on symmetric variables we expected to see probability values with similar magnitudes clustered together. The resultant plots illustrated an unexpected trend of periodicity. This periodic trend was found in many other benchmark circuits and suggests that dependent variables with the same $\wp\{f \oplus x_i\}$ values tend to position themselves as far apart from one another as possible. Figure 1 illustrates the behavior using the best known ordering (see Section 4) for a circuit from the *ISCAS85* benchmark set.

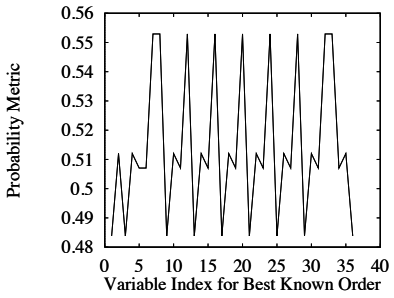

Figure 1: Behavior of $\wp\{f \oplus x_i\}$ versus Best Known Ordering for $c432$, Output $370gat$

In order to exploit the characteristic of periodicity, an algorithm was developed that orders the variables such that the corresponding $\wp\{f \oplus x_i\}$ values are as periodic as possible. The technique involves the following steps:

1. Choose an output about which to compute n values of $\wp\{f \oplus x_i\}$
2. Compute all the probability values
3. Determine histogram bin sizes and widths

4. Form a histogram of the $\wp\{f \oplus x_i\}$ values
5. Starting from the bin with the lowest (or highest) probability value, choose a value
6. Move to the next adjacent bin (in a circular fashion) and choose a value
7. When all values have been removed from the bin, build a list of dependent variables in the same order in which the probability values were chosen
8. Reorder the initial SBDD according to the list just generated
9. Perform a modified sifting routine, we refer to as "bin-sifting"

Figure 2 contains a plot of the $\wp\{f \oplus x_i\}$ values versus the index of x_i after application of the variable reordering algorithm. This data corresponds to the same output as was used in Figure 1, 370*gat*. Clearly, periodicity of the probability values has been enforced.

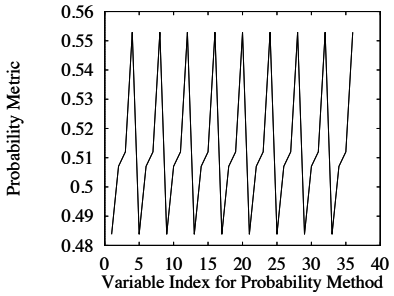

Figure 2: Behavior of $\wp\{f \oplus x_i\}$ versus Probability Based Ordering for *c*432, Output 370*gat*

The histogram bin width has proven to be a crucial factor. Varying this parameter can affect our results. The bin width is currently calculated by the following procedure:

1. Find the two closest valued $\wp\{f \oplus x_i\}$ values that are not equal and compute their difference, δ_s.
2. Find the difference, δ_b, that corresponds to subtracting the overall smallest $\wp\{f \oplus x_i\}$ value from the largest.
3. Set `BINSIZE`=$\frac{\delta_b}{\delta_s}$.
4. If `BINSIZE` exceeds a threshold value, set it equal to the threshold. In the results given here, we use a threshold of 10,000.

2.3 Bin-Sifting

When the order list is created by traversing the histogram, the particular variable chosen from those contained in the same histogram bin is arbitrary. Thus, the resulting intermediate SBDD can be biased by the choice of which variable within a particular histogram bin is chosen. To alleviate effects caused by this bias, we invoke a modified form of the sifting algorithm, referred to as "bin-sifting". In bin-sifting, we use the sifting paradigm but only among variables with common $\wp\{f \oplus x_i\}$ values. That is, all variables within a certain histogram bin are sifted only amongst themselves. This preserves the periodic nature of the histogram of $\wp\{f \oplus x_i\}$ values but often leads to significantly smaller SBDDs. Part of the reason this occurs is that our condition for detecting symmetry is only a necessary one. The condition is not sufficient to prove that two variables are in fact symmetrical.

We have evaluated the effectiveness of bin-sifting alone and some results are given in the experimental section of this paper. We note that using bin-sifting only results in SBDDs that are influenced by their initial variable order since the initial order essentially "fixes" a subset of locations that any one variable can be moved to in the order list. However, bin-sifting in general seems to outperform sifting alone and may be preferable for functions that exhibit little or no variable symmetry.

2.4 Example of the Method

To illustrate the technique described above, a small example is shown with all intermediate calculations given. We have chosen the "toy benchmark" `c17` for this purpose. Although the technique was used for SBDDs with negative-edge attributes, here we use an ROBDD formulation of the `c17` output labeled `23gat` for simplicity.

Initially, the ROBDD exists with a variable order X_2, X_3, X_6, X_7 and is shown graphically in Figure 3. The first step of the process is to compute the output probabilities, $\wp\{f \oplus x_i\} \forall i$. This step requires n traversals of the SBDD which contains N vertices. Since N may range from n to 2^n, the worst case complexity of this step is $O(n2^n)$. However, in practice it is unlikely that a reordering method would be invoked for a SBDD with an exponential number of vertices since such a representation would be impractical to formulate initially. The computed output probabilities for the functions $f \oplus x_i$ are: $\wp\{f \oplus X_2\} = 0.6875$, $\wp\{f \oplus X_3\} = 0.3125$, $\wp\{f \oplus X_6\} = 0.6875$ and $\wp\{f \oplus X_7\} = 0.3125$.

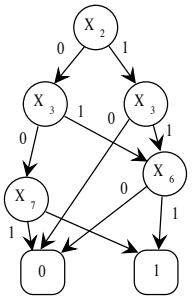

Figure 3: ROBDD of Example Function with Initial Variable Order

A histogram is formed using the probability metrics to guide which variables are grouped together. Figure 4 contains the histogram. A new variable order is obtained by visiting the histogram bins and removing a variable in a circular fashion, thus ensuring periodicity of the $\wp\{f \oplus X_i\}$

values. The new order obtained from the histogram is X_7, X_6, X_3, X_2. An ROBDD is formed by reordering the initial structure according to this new order. The intermediate ROBDD is shown graphically in Figure 5.

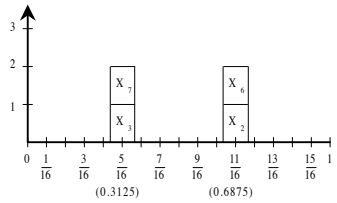

Figure 4: Histogram of Example ROBDD Variables Based on Probability Metrics

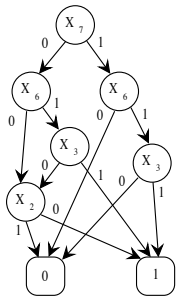

Figure 5: ROBDD of Example Function with Intermediate Variable Order

The last step of the reordering technique involves the application of the bin-sifting routine. Since the example contains two histogram bins, the bin-sifting approach allows for exchanging the locations of $\{X_3, X_7\}$ and $\{X_6, X_2\}$ only. We note that if all probability metrics had been equal, the bin-sifting approach degenerates into the standard sifting algorithm. The resulting variable order for the example function is found to be X_3, X_6, X_7, X_2. The final ROBDD is shown in Figure 6. In summary, we note that the size of the example ROBDD was initially 5 vertices and resulted in 4 vertices for the final result.

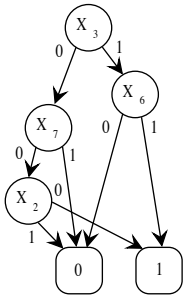

Figure 6: ROBDD of Example Function after Variable Reordering

```
evolutionary_minimizer (function) {
generate_initial_population () ;
{
  element := select_element (population) ;
  choose_operator (SIFT, INV) ;
  if (operator == INV) {
    INV_element := apply_INV (element) ;
    SIFT_element :=
        apply_SIFT (INV_element) ;
  } else {
    SIFT_element := apply_SIFT (element) ;
  }
  if (element == SIFT_element) {
    new_element :=
        apply_mutation (SIFT_element) ;
  }
  update_population () ;
} while (improvement obtained) ;
return ;
}
```

Figure 7: Sketch of basic algorithm

3 Evolutionary Algorithms for Variable Reordering

As an alternative to the reordering algorithm proposed above we now present an approach to SBDD minimization based on *Evolutionary Algorithms* (EAs). This approach is an extension of the methods described in [9], but can also deal with large functions. (Due to page limitation only the main aspects are discussed.)

The core of our simulation based algorithm is a EA. The main difference to other EA based optimization approaches is that we make use of small population sizes only. Also, we restrict the algorithm to use operators that guarantee that the SBDD size does not grow too much within one application of an operator. Thus, large SBDDs can be handled within reasonable time bounds and given memory limits can also be effectively controlled.

As is well-known for some functions the BDD can not be constructed if a "bad" variable ordering is used. Thus, methods that make use of standard EA recombination operators like PMX, OX, CX (as used in [9]) fail, since in those cases, orderings can be constructed that are too "far away" from manageable solutions.

In contrast we make use of operators that have polynomial upper bounds (as can be easily derived from the upper bounds of the *swap* operator in [2]). Additionally, we make use of sifting as an operator in our simulation algorithm.

We now briefly describe the basic algorithm.

- First an initial population of orderings of size $|\mathcal{P}|$ is determined by using the method in [10] and then applying mutations.
- $|\mathcal{P}|$ new individuals are created using *inversion* (INV) and *sifting* (SIFT) with given probabilities; the parent

elements are randomly selected. If the choice was operator INV, SIFT is applied after applying INV to obtain a (new) local minimum. If the offspring is equal to their parent element it is mutated by one of the mutation operators.

- The best $|\mathcal{P}|$ individuals are selected from the pool of "old" individuals and offspring for the new population in the next generation.

- The algorithm stops if the best element has not changed for 200 generations.

A sketch of the algorithm is given in Figure 7.

4 Experimental Results

We have computed orderings for several benchmark circuits using the techniques described above. We have compared these to sifting alone which yields reordering improvements in a small amount of CPU time, and best known orderings which were generally computed using techniques such as *Simulated Annealing* (SA) or EAs that yield very good minimized SBDDs but can require larger amounts of computation time.

The first set of experimental data compares the results of the probability based heuristic to that of sifting. Table 1 contains reordering results for some of the `pla` benchmarks that exhibited a size reduction. Many of the *pla* circuits yield non-interesting results in that the size of the resulting SBDD is the same regardless of the order, thus the small subset in Table 1. The initial ordering was arbitrarily set to the order in which the variables appear in the *pla* files. In all experiments performed using the *pla* benchmarks, the size of the resulting SBDD was always less than or equal to the original size. In some cases the reduction was up to 20%.

Table 1: Experimental Results for the Reordering Technique Using *PLA* Benchmarks

circuit		sifting		prob. method	
Name	Org. Size	Size	Time	Size	Time
5xp1	74	51	< 1	42	< 1
alu4	1197	800	< 1	639	1
bw	108	103	< 1	99	< 1
duke2	973	394	< 1	339	1
misex1	41	37	< 1	37	< 1
misex2	136	89	< 1	84	< 1
misex3	1301	704	< 1	683	5
sao2	155	94	< 1	87	< 1
misex3c	828	504	< 1	472	1
clip	226	87	< 1	87	< 1
e64	1441	231	1	231	2
apex1	28336	1356	3	1333	58
apex4	928	895	< 1	889	1
apex5	2679	1130	< 1	1130	1

Table 2 contains the data resulting from our technique, sifting alone and bin-sifting alone. Results for benchmarks `c2670` and `c7552` were not included since attempts to convert these into an SBDD with a random initial variable order resulted in exceeding the maximum SBDD vertex limit in our program. `c6288` is also not included since it represents a multiplier circuit that is known to not have any size less than exponential in the number of variables.

In all cases, the probability method performed as well or better than sifting or bin-sifting alone in terms of resultant SBDD size. The increase in CPU time is less than an order of magnitude as compared to sifting in most cases. The results using bin-sifting alone illustrate that it generally performs as well or better than sifting, but at an increase in CPU time due primarily to the large size of the initial SBDD structure.

The results in Tables 1 and 2 also contain a column for total CPU time. This is the time that was required for generating the probabilities, creating and traversing the histogram, reordering according to histogram traversal and performing bin-sifting. The time was obtained using a Sun SPARCstation 20 with 240 MB of RAM, a single 100 MHz processor and running under Solaris 2.5. It must be noted that, these times assume that it is known prior to execution which output will be used for computing the $\wp\{f \oplus x_i\}$ values.

Table 3 contains best-known size data. We compare the results obtained by our EA (last column) to the SA approach from [2] and the "best-ever" sifting results from [17]. This information is useful for comparison to the data obtained above and may be used to judge the effectiveness of our technique in terms of resulting SBDD size. As can be seen the EA obtains better results than sifting in many cases, but is still applicable where the SA fails.

Table 3: Best Known SBDD Sizes for *ISCAS*85 Benchmarks

circuit	SA	sift best-ever	EA
c432	1087	1210	1064
c499	25866	25866	25866
c880	4053	4083	4053
c1355	25866	25866	25866
c1908	5526	5708	5526
c2670	-	2008	1774
c3540	23828	23828	23828
c5315	-	2104	1719
c7552	-	3730	2212

We currently have not incorporated an automated technique for determining which SBDD output about which the output probabilities should be computed. The results given above are randomly selected outputs from the benchmark functions. We do note that there is typically not a wide variance in the resulting SBDD sizes based on the selected output. As proof of this phenomena, Table 4 contains the maximum, minimum, average and standard deviation of the sizes of SBDDs computed using several different outputs for some of the *ISCAS85* benchmark circuits. The standard de-

Table 2: Experimental Results for the Reordering Technique Using *ISCAS*85 Benchmarks

benchmark		sift		bin-sift		prob. meth.	
Circuit	Org. Size	Size	Time	Size	Time	Size	Time
c432	31172	20891	28	20891	216	1119	13
c499	53866	38196	84	36234	156	34432	245
c880	10348	5089	9	4603	20	4603	26
c1355	53866	38186	88	36234	159	34432	280
c1908	13934	8175	11	7762	23	6170	48
c3540	72858	41918	73	37644	1421	38096	257
c5315	747662	4078	225	-	-	3447	4108

viation values indicates the extent of how the output selection affects the resultant SBDD size.

Table 4: SBDD Size Statistics Versus Output About Which Probabilities are Computed

circuit name	number of outputs	max size	min size	avg size	stan. dev.
c432	6	1636	1119	1340	173
c499	32	36250	34432	35784	781
c880	12	31990	4603	7276	3197
c1355	32	36250	34432	35784	781
c1908	25	8174	6170	7607	2185

5 Conclusion

A variable reordering method for SBDDs with negative edge attributes has been presented and described. Experimental results indicate that runtimes are roughly equivalent to those of sifting, however smaller SBDDs generally result. The method differs significantly from those previously published in the way that function properties are used. This technique yields a resultant SBDD that is independent of the initial structure and is thus very robust.

The quality of our results has been shown by a comparison to an EA. The EA algorithm was able to further improve the best known SBDD sizes for many functions.

References

[1] S. B. Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, vol. C-27, no. 6:509–516, June 1978.

[2] B. Bollig, M. Löbbing, and I. Wegener. Simulated Annealing to Improve Variable Ordering for OBDDs. *Proceedings of the IEEE/ACM Workshop on Logic Synthesis*, pages 1–15, May 1995.

[3] B. Bollig and I. Wegener. Improving Variable Ordering is NP-Complete. *IEEE Transactions on Computers*, vol. 45, no. 9:993–1002, September 1996.

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

[5] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, vol. C-35, no. 8:677–691, August 1986.

[6] S. Chakravarty. On the Complexity of Using BDDs for the Synthesis and Analysis of Boolean Circuits. *Proceedings of the 27th Annual Allerton Conference on Communication, Control, and Computing*, pages 730–739, 1989.

[7] P. Y. Chung, I. N. Hajj, and J. Patel. Efficient Variable Ordering Heuristics for Shared ROBDD. *IEEE International Symposium on Circuits and Systems*, pages 1690–1693, 1993.

[8] S. Devadas. Comparing Two-Level and Ordered Binary Decision Diagram Representations of Logic Functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 5:722–723, May 1993.

[9] R. Drechsler, B. Becker, and N. Göckel. A genetic algorithm for variable ordering of OBDDs. *IEE Proceedings*, 143(6):364–368, 1996.

[10] 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.

[11] R. Krieger. PLATO: A Tool for Computation of Exact Signal Probabilities. *Proceedings of VLSI Design Conference*, 1993.

[12] C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technical Journal*, vol. 38:985–999, July 1959.

[13] 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.

[14] D. M. Miller. An Improved Method for Computing a Generalized Spectral Coefficient. *IEEE Trans. on CAD/ICAS*, vol. 17, no. 3:233–238, March 1998.

[15] S. Minato, N. Ishiura, and S. Yajima. Shared Binary Decision Diagram with Attributed Edges for Efficient Boolean Function Manipulation. *Proceedings of the ACM/IEEE Design Automation Conference*, pages 52–57, 1990.

[16] 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.

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

[18] S. Panda and F. Somenzi. Who are the Variables in Your Neighborhood. *Proceedings of the IEEE/ACM Workshop on Logic Synthesis*, pages 5–11,5–19, May 1995.

[19] K. P. Parker and E. J. McCluskey. Probabilistic Treatment of General Combinational Networks. *IEEE Transactions on Computers*, vol. c-24:668–670, June 1975.

[20] R. Rudell. Ordering for Ordered Binary Decision Diagrams. *Proceedings of the International Conference on Computer Aided Design*, pages 42–47, November 1993.