# SIMULATED ANNEALING TO IMPROVE VARIABLE ORDERINGS FOR OBDDs

**Beate Bollig***, **Martin Löbbing*** and **Ingo Wegener***

FB Informatik, LS II, Univ. Dortmund, 44221 Dortmund, Fed. Rep. of Germany

## Abstract

The choice of a good variable ordering is crucial in applications of Ordered Binary Decision Diagrams (OBDDs). A simulated annealing approach with a new type of neighborhood is presented and analyzed. Better results as by known simulated annealing algorithms and heuristics are obtained. Some theoretical results underlining the experiments are stated.

**Keywords:** formal verification, formal techniques and methods for verification, symbolic manipulation, OBDDs, variable orderings, simulated annealing.

*Supported in part by DFG grant We 1066/7–1.

## 1. INTRODUCTION

Boolean function manipulation is an important component of many logic synthesis algorithms including logic optimization and logic verification of combinational and sequential circuits. Ordered Binary Decision Diagrams (OBDDs) have proven useful in these applications as an efficient data structure for the representation and manipulation of Boolean functions (Bryant [2]), if good variable orderings are known. The best algorithm for the computation of an optimal variable ordering is due to Friedman and Supowit [4] and uses a dynamic programming approach. It needs exponential time and space. Hence, a lot of heuristic algorithms have been proposed (see e. g. [3], [5] – [13], [15]). Due to the nature of heuristics, no single heuristic always produces an appropriate ordering. It is therefore a natural idea to improve the orderings computed by a heuristic algorithm. Fujita, Matsunaga and Kakuda [6] as well as Rudell [13] suggest local search algorithms for this purpose. Simulated annealing algorithms for the computation of variable orderings are presented by Ishiura, Sawada and Yajima [7] and also by Mercer, Kapur and Ross [10]. It is known that the structure of the neighborhood of a variable ordering is essential for local search and simulated annealing algorithms. We suggest a new neighborhood which leads to better variable orderings. For all considered local changes of a variable ordering we investigate how it may increase and decrease the OBDD size. These results solve a problem posed by Rudell [13].

In general an OBDD is a directed acyclic graph where the nodes on the $i$-th layer are labeled by the $i$-th variable in the given ordering and the sinks on the $(n+1)$-th layer are labeled by Boolean constants. Each non sink node has two outgoing edges, one labeled by 0 and the other by 1. Edges starting at layer $i$ have to end at a layer $j > i$. Each OBDD node $v$ represents a Boolean function $f_v$. In order to compute $f_v(a)$ one starts at the node $v$ and follows at an $x_i$-node the edge with label $a_i$. The value of $f_v(a)$ is equal to the label of the sink which is finally reached. An OBDD is called reduced, if no node has the same 0-successor and 1-successor and if the OBDD does not contain isomorphic subgraphs. It is well-known (Bryant [2]) that reduced OBDDs are a canonical representation. Checking for the equivalence of two functions reduces to checking that the canonical forms are identical. In the following we discuss only reduced OBDDs.

In Section 2 simulated annealing algorithms for the computation of good variable orderings are presented. The local change of a variable ordering can easily be realized using the OBDD package of Brace, Rudell and Bryant [1]. In Section 3 the effectiveness of our methods is evaluated by conducting experiments on benchmark circuits. Finally, it is determined how the OBDD size may decrease and increase by a local change.

## 2. SIMULATED ANNEALING FOR THE COMPUTATION OF VARIABLE ORDERINGS

Simulated annealing algorithms compute under certain assumptions on the long run optimal solutions for optimization problems. But in practice one has not that much computation time and practical simulated annealing algorithms are heuristic algorithms which are quite successful for many problems. We present only the main features of our algorithm.

Usually the cost of a configuration can be computed very efficiently, e.g. the cost of a tour for a traveling salesman can be computed in linear time. In our situation we have to construct the OBDD according to a variable ordering $\pi$. We do not know any other general method to compute the size of the OBDD. This leads to the problem that we can in general not start from a random variable ordering. The OBDD for this ordering may be too large to be constructed. In these cases we have to start from an OBDD according to a heuristically computed ordering. Simulated annealing algorithms accept probabilistically also cost increasing steps. We perform the random decision first and decide which OBDD size we tolerate in the next step. Then we can interrupt the construction of OBDDs which certainly become too large. A crucial point seems to be the definition of the neighborhood of a variable ordering. We like to have short distances to all other orderings and nevertheless small neighborhoods. Moreover, the construction of the OBDD according to a neighbor $\pi^*$ of $\pi$ should be efficiently possible from the OBDD according to $\pi$.

Let us assume in this section that the given ordering is $x_1, x_2, \ldots, x_n$. Fujita, Matsunaga and Kakuda [6] and Rudell [13] work only with swaps. The operation swap$(i, i+1)$ exchanges the positions $i$ and $i+1$ in the ordering, the result is $x_1, \ldots, x_{i-1}, x_{i+1}, x_i, x_{i+2}, \ldots, x_n$. It is easy to use the OBDD package of Brace, Rudell and Bryant [1] to perform a swap. Only the levels $i$ and $i+1$ are changed. But for two random orderings $\pi$ and $\pi'$ we need $\Theta(n^2)$ swaps to reach $\pi'$ from $\pi$. Ishiura, Sawada and Yajima [7] and also Mercer, Kapur and Ross [10] investigate therefore more general exchanges. The operation exchange$(i, j)$ for $i < j$ exchanges the positions $i$ and $j$ in the ordering, the result is the ordering $x_1, \ldots, x_{i-1}, x_j, x_{i+1}, \ldots, x_{j-1}, x_i, x_{j+1}, \ldots, x_n$. Again the OBDD package [1] can be used for an exchange operation. Wang, Hwang and Chen [17] have shown how the new OBDD can be computed with a single call of the apply procedure. Let $x_i \equiv x_j$ test whether $x_i$ and $x_j$ have the same value and let $x_i \leftarrow \bar{x}_i$ be the operation to exchange the 0- and the 1-edges leaving $x_i$-nodes. Then $f = \text{ite}(x_i \equiv x_j, f, f_{x_i \leftarrow \bar{x}_i, x_j \leftarrow \bar{x}_j})$.

Our new operation jump$(i, j)$ causes $x_i$ to jump to the $j$-th position. The relative ordering of all other variables remains unchanged. If $j < i$, the operation is called jump-up and the result is $x_1, \ldots, x_{j-1}, x_i, x_j, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$. If $j > i$, the operation is called jump-down and the result is $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j, x_i, x_{j+1}, \ldots, x_n$. It is obvious that $f = \text{ite}(x_i, f_{|x_i=1}, f_{|x_i=0})$. The OBDDs for $f_{|x_i=1}$ and $f_{|x_1=0}$ respect the original variable ordering as well as the variable ordering after the jump operation. Hence, the new OBDD can be computed with a single call of the apply procedure. Therefore, it is not more difficult to realize a jump operation with the OBDD package [1] than to realize an exchange operation. With both operations any random ordering can be reached from $\pi$ with at most $n-1$ steps.

It is easy to see that $|i-j|$ exchanges or swap operations are necessary to simulate one jump$(i, j)$ operation but each exchange step can be simulated with two jumps. Moreover, jump operations seem to be powerful, if the considered ordering (perhaps computed by a heuristic) is quite good but some variables are placed at the wrong place.

The neighborhood of a variable ordering is now defined as the set of all variable orderings which can be produced with a single exchange or jump operation (swap is a special case of exchange and of jump). The random choice of a neighbor can be done by a random choice of a pair $(i, j)$, $i \neq j$, and a random Boolean value $c$. If $c = 0$, exchange$(i, j)$ (or exchange $(j, i)$, if $j < i$) is performed. If $c = 1$, jump$(i, j)$ is performed. Another possibility is to alternate between exchange and jump steps.

Rudell [13] uses a local search approach. He tries for some variable $x_i$ all jump$(i, \cdot)$ operations (realized by a sequence of swap operations) and chooses the best position for $x_i$. He poses the problem of determining the minimal and maximal size of the created OBDDs. We solve his problem by determining the maximal increase and maximal decrease in the OBDD size for jump, swap and exchange steps.

**Theorem 1:** Let $G$ be an OBDD and let $G^*$ be the result after one of the local changes. The following holds:

- swap: $\frac{1}{2} \cdot |G| \leq |G^*| \leq 2 \cdot |G|$
- exchange: $|G|^{1/2} \leq |G^*| \leq |G|^2$
- jump-up: $|G|^{1/2} \leq |G^*| \leq 2 \cdot |G|$
- jump-down: $\frac{1}{2} \cdot |G| \leq |G^*| \leq |G|^2$

Since the bounds of Theorem 1 are almost tight, jump-up operations are quite promising. We have allowed also jump-down operations in order to obtain symmetric neighborhoods. This means that $\pi^*$ is reachable in one step from $\pi$ if and only if $\pi$ is reachable in one step from $\pi^*$. This condition ensures the theoretical convergence to an optimal solution. Moreover, it ensures that a bad move can be corrected quickly.

We conclude that jump operations have promising properties. Simulated annealing algorithms should be improved if besides swaps and exchanges also jumps are allowed. It is easier to simulate exchanges with jumps than to simulate jumps with exchanges.

## 3. EXPERIMENTAL RESULTS

Typical heuristic algorithms for the computation of variable orderings work on circuits and determine deterministically a variable ordering reflecting the circuit structure. They do not need much storage space and the computation of the variable ordering is usually much more time efficient than the translation of the circuit into an OBDD according to this variable ordering. The computation of a variable ordering or the improvement of a variable ordering with the simulated annealing approach has different features. Many orderings are tried and, therefore, simulated annealing algorithms can never compete with circuit based heuristics with respect to computation time. Hence, simulated annealing should be applied only, if the other heuristics do not lead to good variable orderings or if one is interested in optimal orderings. The computation time needed to get an OBDD with fewer nodes is sometimes compensated by the algorithms working on these OBDDs afterwards. Because simulated annealing is a randomized search strategy, it admits the possibility of finding good variable orderings which cannot be computed directly from the circuit. For this reason we have performed experiments with an adder and eight benchmark circuits (ttt2, vda, C432, C499, C1355, C880, C1908 and C3540). For each of these circuits we constructed shared OBDDs representing all outputs at once. It would be easy to perform simulated annealing on each output separately, but we want to show that our algorithm is suitable even for large circuits. Our results can also be used to appraise other heuristics by comparing the sizes of the OBDDs.

An example where jump steps are much better than exchange steps are binary adders for the numbers $a = (a_{n-1}, \ldots, a_0)$ and $b = (b_{n-1}, \ldots, b_0)$. An optimal ordering is $a_{n-1}, b_{n-1}, a_{n-2}, b_{n-2}, \ldots, a_0, b_0$ which is also found by some of the known heuristics. With jump steps and starting from a random ordering a variable $a_i$ can find its partner $b_i$ in one step without destroying other pairs $(a_j, b_j)$. This is not possible with exchange steps. This example confirms our claim that exchange and jump steps should be used.

We have worked with the benchmark circuits ttt2 (24 inputs, 21 outputs) and vda (17 inputs, 39 outputs) as toy examples. These circuits lead to OBDDs of moderate size even for random orderings. Our aim was to investigate whether the most often used heuristics find optimal orderings in these situations or whether our simulated annealing algorithm computes better orderings than the heuristic algorithms even for very simple circuits. We have used the most well-known heuristics: depth first search, breadth first search and fault simulation [3], the level and the fanin heuristic [9], and the fanout heuristic [5].

Further we discuss the experiments for six more complex benchmark circuits, the priority decoder C432 (36 inputs, 7 outputs, 160 gates), C499 (41 inputs, 32 outputs, 202 gates, equivalent to C1355) and C1908 (33 inputs, 25 outputs, 880 gates) for fault corrections and the ALU's C880 (60 inputs, 26 outputs, 383 gates) and C3540 (50 inputs, 22 outputs, 1669 gates).

The results are shown in Table 1. The rows [6] sim. ann. and [7] describe also results for simulated annealing algorithms. The first row of the sifting algorithm by Rudell [14] states the results of an uniform choice of the controlling parameters, the second the "best ever" results using this algorithm with different parameters for each circuit. In comparison with simulation annealing it needs much less computation time, but cannot reach the good results of this approach.

Our cooling schedule [16] is defined as follows: The first transitions are always accepted resulting in a random start permutation. At the same time the size of the encountered OBDDs is used to calculate an initial temperature. In the real cooling process on each temperature level a fixed number of accepted transitions are made before the temperature is reduced using a dynamic approach. The process stops when no further improvement is found.

As an example for computation times we look at the circuit C499. With our cooling schedule the simulated annealing algorithm made about 20,000 transitions and needed

| | ttt2 | vda | C432 | C499 | C1355 | C880 | C1908 | C3540 |
|---|---|---|---|---|---|---|---|---|
| original slif | 223 | 4,348 | 1,733 | 45,922 | 45,922 | 346,686 | 36,007 | 604,558 |
| dfs | 190 | 529 | 32,278 | 39,858 | 39,858 | 6,739 | 19,498 | 205,158 |
| bfs | 171 | 3,647 | 8,785 | 122,374 | 100,181 | 39,291 | 17,944 | 257,343 |
| level | 163 | 2,733 | 29,801 | 48,431 | 46,044 | 500,731 | 36,012 | 502,184 |
| fanin | 170 | 515 | 26,680 | 39,858 | 58,513 | 346,762 | 18,435 | 611,088 |
| fanout | 186 | 533 | 26,680 | 57,434 | 54,130 | 346,762 | 9,356 | 611,088 |
| fault sim | 234 | 533 | 31,073 | 38,314 | 49,980 | 70,071 | 18,764 | 590,215 |
| [1] | – | – | 30,200 | 49,786 | 29,858 | 7,655 | 12,463 | 208,947 |
| [3] B | – | – | 42k | n.p. | – | n.p. | 143k | n.p. |
| [3] D | – | – | n.p. | 238k | 483k | 35k | 180k | – |
| [3] SIM | – | – | n.p. | n.p. | – | n.p. | 228k | – |
| [3] SC | – | – | 80k | n.p. | – | n.p. | 342k | n.p. |
| [3] cmu | – | – | n.p. | 235k | 477k | 29k | 147k | – |
| [3] ucb | – | – | n.p. | 217k | 438k | 33k | 147k | n.p. |
| [6] heu | – | – | 11,262 | – | – | 27,656 | – | – |
| [6] sim. ann. | – | – | 8,387 | – | – | 24,763 | – | – |
| [7] | – | – | 1,209 | 32,784 | – | 4,414 | 6,153 | – |
| [11] | – | – | 89,338 | 36,862 | 119,201 | 30,548 | 39,373 | – |
| [14] sifting | – | – | 1,213 | 40,061 | 40,061 | 6,535 | 6,049 | 23,850 |
| [14] sifting "best ever" | – | – | 1,213 | 29,569 | 31,447 | 5,559 | 5,709 | 23,837 |
| simann | 107 | 478 | 1,087 | 25,866 | 25,866 | 4,053 | 5,526 | 23,828 |

Table 1: Experimental results for benchmark circuits (n.p. = not possible)

60,000 seconds on a Sun SparcServer 670/140. For five different starting values for the random number generator the process ended three times with an OBDD of best found size (25,866 nodes).

To check the effectivity of our jump operation, experiments have also been made with different neighborhoods (and other different parameters). Using only the swap operation one obtains for the small examples quite often the best ordering but one needs a lot of steps and also a lot of computation time although a swap step can be realized much faster than a jump or an exchange step. It is better to use only exchange steps than to use only jump-up steps but the best ordering is not computed often with exchange steps only.

For the same example as above, the neighborhood allowing only exchange steps leads four times to OBDDs with 26,408 nodes or more, only once to one with 25,874 nodes; an OBDD with 25,866 nodes is never obtained. The greater size results in a greater need of time (about 70,000 seconds), since many OBDDs are examined at the end of the process. It is obvious that an enlarged neighborhood can only reduce the number of local minimums and so reduce the number of possible nonoptimal solutions. This may be the reason for the better results.

Another example for the better performance of the large neighborhood is C3540. An OBDD for C3540 with size 23,829 is found almost always with the large neighborhood, never with the smaller one (the same cooling schedules provided), but the needed time of more than 100,000 seconds is enormous. The computation time needed for C880 is not so large, but the best result is obtained only by one of forty processes. The successful process uses the large neighborhood and needs 73,000 seconds.

As we have conjectured it is the best approach to use exchange, jump-up, and jump-down steps. With this large neighborhood the best orderings are computed and for the small examples these best orderings are computed in almost all runs of our algorithm and much faster than with swap steps only.

We conclude that our simulated annealing algorithm is better than the previous simulated annealing algorithms and the reason is mainly the improved neighborhood of the variable orderings.

## 4. THEORETICAL BACKGROUND

We give concrete upper bounds for the OBDD size after one iteration step for jump operations. The results for swap and exchange operations are then easy to obtain, since a swap is a special jump-up and an exchange step can be performed by a jump-up followed by a jump-down.

Let us first investigate the jump-up$(i,j)$ operation. Let $s_k$ be the number of nodes labeled $x_k$ in the OBDD $G$ according to $x_1,\ldots,x_{j-1},x_j,x_{j+1},\ldots,x_{i-1},x_i,x_{i+1},\ldots,x_n$ and $s_k^*$ the number of nodes labeled $x_k$ in the OBDD $G^*$ according to $x_1,\ldots,x_{j-1},x_i,x_j,x_{j+1},\ldots,x_{i-1}$, $x_{i+1},\ldots,x_n$. If for some variable $x_k$ the same variables are tested in $G$ and $G^*$ before $x_k$, then obviously $s_k=s_k^*$.

**Theorem 2:** The size of the OBDD after the jump-up$(i,j)$ operation is bounded above by

$$2(s_1 + \ldots + s_{i-1}) + 1 + s_{i+1} + \ldots + s_n$$

and also by

$$s_1 + \ldots + s_{j-1} + 3(s_j + \ldots + s_{i-1}) + s_i + s_{i+1} + \ldots + s_n.$$

In particular, the size of an OBDD can be at most doubled by a jump-up operation.

**Sketch of proof:** The number of $x_i$-nodes in $G^*$ is bounded by $s_1 + \ldots + s_{j-1} + 1$, since this is the maximum number of edges leaving the first $j-1$ levels. The number of $x_k$-nodes, $j \leq k < i$, is at most doubled, since the value of $x_i$ has to be stored. This proves the first bound. The proof of the second bound is more involved. $\Box$

Now we consider the jump-down$(i,j)$ operation. Let $s_k^*$ be the number of nodes labeled $x_k$ in the OBDD $G^*$ according to $x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_j, x_i, x_{j+1}, \ldots, x_n$.

**Theorem 3:** The size of the OBDD after the jump-down$(i,j)$ operation is bounded above by

$$s_1 + \ldots + s_{i-1} + (s_{i+1} + \ldots + s_n + 2)^2$$

and also by

$$s_1 + \ldots + s_n + (2^{j-i+1} - 2)s_i.$$

In particular, the size of an OBDD can be at most squared by a jump-down operation (here the sinks are also counted).

**Sketch of proof:** The jump-down operation can be implemented as follows. We construct $G_1$ and $G_0$ by replacing $x_i$ by 1 resp. 0. Then $G_1$ and $G_0$ are OBDDs respecting the new ordering, since they do not contain $x_i$-nodes. $G^*$ is obtained by using the ite-operator with respect to $x_i$, $G_1$, and $G_0$. Since the upper parts in $G_1$ and $G_0$ are isomorphic, our first bound follows. For the second bound we use the fact that a jump-down operation can be simulated by swap operations. $\Box$

Theorem 1 follows from Theorem 2 and Theorem 3.

## 5. CONCLUSION

A new simulated annealing algorithm for the computation of good variable orderings for OBDDs is presented. The new neighborhood is investigated theoretically by determining the maximal increase and maximal decrease of the OBDD size by one local step. The algorithm computes for some benchmark circuits orderings which are better than the formerly best known orderings.

### 6. APPENDIX: VARIABLE ORDERINGS

The best orderings we have found are the following ones.

ttt2: 1, 2, 16, 3, 22, 21, 4, 7, 9, 5, 8, 6, 20, 24, 19, 18, 17, 15, 14, 23, 11, 13, 12, 10.

vda: 17, 13, 15, 14, 4, 7, 11, 16, 9, 10, 5, 8, 3, 2, 6, 1, 12.

C432: 2, 1, 30, 28, 3, 31, 5, 33, 34, 32, 35, 36, 26, 24, 27, 29, 22, 20, 23, 25, 18, 16, 19, 21, 6, 4, 7, 9, 10, 8, 11, 13, 17, 15, 12, 14.

C499/C1355: 3, 8, 4, 7, 24, 19, 20, 23, 27, 32, 28, 31, 12, 16, 11, 15, 26, 25, 17, 18, 35, 22, 21, 30, 29, 36, 41, 9, 13, 1, 5, 33, 2, 10, 39, 6, 14, 40, 38, 34, 37.

C880: 56, 59, 57, 55, 1, 11, 8, 13, 3, 12, 14, 9, 5, 16, 60, 6, 15, 17, 10, 40, 23, 4, 2, 50, 24, 22, 34, 42, 25, 33, 41, 49, 43, 26, 37, 38, 36, 27, 18, 44, 39, 21, 35, 28, 45, 29, 30, 31, 46, 47, 19, 48, 58, 32, 51, 53, 7, 20, 54, 52.

C1908: 33, 31, 22, 25, 13, 11, 14, 4, 16, 1, 2, 15, 10, 12, 3, 6, 7, 9, 5, 8, 21, 26, 17, 30, 29, 27, 18, 28, 19, 20, 32, 24, 23.

C3540: 1, 3, 27, 48, 50, 2, 4, 6, 5, 38, 23, 26, 25, 24, 16, 15, 17, 46, 18, 19, 20, 21, 49, 28, 29, 22, 30, 7, 8, 31, 45, 9, 32, 33, 10, 34, 11, 44, 12, 35, 13, 43, 39, 36, 14, 37, 42, 41, 40, 47.

## References


[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *Proc. of the 27th ACM/IEEE Design Automation Conference*, pages 40–45, 1990.

[2] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. Computers*, C-35(8): pages 677–691, 1986.

[3] K. M. Butler, D. Ross, R. Kapur, and M. R. Mercer. Heuristics to compute variable orderings for efficient manipulation of ordered binary decision diagrams. In *Proc. of the 28th ACM/IEEE Design Automation Conference*, pages 417–420, 1991.

[4] S. J. Friedman and K. J. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Trans. Computers*, C-39(5): pages 710–713, 1990.

[5] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of Boolean comparison method based on binary decision diagrams. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 2–5, 1988.

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

[7] N. Ishiura, H. Sawada, and S. Yajima. Minimization of binary decision diagrams based on exchange of variables. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 472–475, 1991.

[8] S.-W. Jeong, B. Plessier, G. D. Hachtel., and F. Somenzi. Variable ordering and selection for FSM traversal. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 476–479, 1991.

[9] S. Malik, A. R. Wang, R. K. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 6–9, 1988.

[10] M. R. Mercer, R. Kapur, and D. E. Ross. Functional approach to generating orderings for efficient symbolic representation. In *Proc. of the 29th ACM/IEEE Design Automation Conference*, pages 624–627, 1992.

[11] S.-I. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient Boolean function manipulation. In *Proc. of the 27th ACM/IEEE Design Automation Conference*, pages 52–57, 1990.

[12] D. E. Ross, K. M. Butler, R. Kapur, and M. R. Mercer. Fast functional evaluation of candidate OBDD variable orderings. In *Proc. of the European Design Automation Conference*, pages 4–10, 1991.

[13] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 42–47, 1993.

[14] R. Rudell. Personal communication. 1994.

[15] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDD's. In *Proc. of the ACM/IEEE Int. Conf. on Computer Aided Design*, pages 130–133, 1991.

[16] P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, 1987.

[17] K. H. Wang, T. T. Hwang, and C. Chen. Restructuring binary decision diagrams based on functional equivalence. In *Proc. of the European Design Automation Conference*, pages 261–265, 1993.