BDD Variable Ordering By Scatter Search

William N. N. Hung
Intel Architecture Group
Intel Corporation
Hillsboro, Oregon, USA

Xiaoyu Song
Department of ECE
Portland State University
Portland, Oregon, USA

Abstract

Reduced Ordered Binary Decision Diagrams (BDDs) are a data structure for representation and manipulation of Boolean functions which are frequently used in VLSI Design Automation. The variable ordering largely influences the size of the BDD, varying from linear to exponential. In this paper we study BDD minimization problem based on scatter search optimization. The results we obtained are very encouraging in comparison with other heuristics (genetic and simulated annealing). This work is the first successful experience of using scatter search approach in design automation area. The approach can be applied to many design automation applications.

1. Introduction

Reduced Ordered Binary Decision Diagrams (BDD's) [2] have found extensive use in various algorithms for analyzing Boolean functions. Some fields of application are logic design verification, test generation, fault simulation, and logic synthesis [18][19]. Most of the algorithms using BDD's have polynomial run times in the size of the BDD's. Unfortunately, in many applications, very large sized BDDs can be generated which can render a BDD based analysis scheme impractical or inefficient. In particular, BDD sizes are very sensitive to the order chosen on input variables; any "carelessness" in choosing orders can easily lead to exponentially large graphs even though the given function could easily have been represented as a very compact BDD under a good variable order [11][19]. 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 a BDD.

Although there are numerous heuristics in BDD minimization [3][15][10][5][24], the lower bound on BDD sizes can still be exponential in some cases [14]. In general, exact BDD minimization is NP-complete [27]. This paper explores a powerful class of optimization techniques based on scatter search [13][17][4]. Scatter search was introduced by Fred Glover [13][12][17] in 1977. It has been largely ignored for about twenty years until recently. Scatter search is very aggressive and attempts to find high quality solutions fast. It derives its foundations from strategies originally proposed for combining decision rules and constraints in the context of integer programming. In this paper we study BDD minimization by using scatter search optimization techniques.

The organization of this paper is as follows. In Section 2, we present the BBD definitions and ordering problem definition. In Section 3, we present the optimization procedure based on scatter search. In Section 4, we use the scatter search to solve the BDD minimization problem. In Section 5, we report some experimental results performed for the model. Section 6 concludes the paper. Our experiment demonstrates the effectiveness of our approach.

2. Preliminaries

We start with a brief review of the essential definitions.

Definition 1: A BDD is a rooted directed acyclic graph $G=(V, E)$ with vertex set V containing two types of vertices, *nonterminal* and *terminal* vertices. A nonterminal vertex v has as label a variable $index(v) \in \{x_1, x_2, ..., x_n\}$ and two children $low(v)$, $high(v) \in V$. A terminal vertex v is labeled with a value $value(v) \in \{0,1\}$ and has no outgoing edge.

A BDD can be used to compute a Boolean function $f(x_1, x_2, ..., x_n)$ in the following way. Each input $a = (x_1, x_2, ..., x_n) \in \{0,1\}^n$ defines a computation path through the BDD that starts at the root. If the path reaches a nonterminal node v that is labeled by x_i, it follows the path $low(v)$ iff $a_i=0$, and it follows the path $high(v)$ iff $a_i=1$. On all paths a terminal vertex is reached since a BDD is directed and acyclic. The label of the terminal vertex determines the return value of the BDD on input a.

More formally, we can define the Boolean function corresponding to a BDD recursively.

Definition 2: A BDD having root vertex v denotes a Boolean function f_v defined as follows.

1. If v is a terminal vertex and $value(v)=1$ ($value(v)=0$), then $f_v=1$ ($f_v=0$).
2. If v is a nonterminal vertex and $index(v)=x_i$, then f_v is the function $f_v(x_1,...,x_n) = \bar{x}_i \bullet f_{low(v)}(x_1,...,x_n)+x_i \bullet f_{high(v)}(x_1,...,x_n)$.

   The variable x_i is called the decision variable for v.

Definition 3: An ROBDD is a BDD with the following properties:

1. The BDD is *ordered*, i.e., there is a fixed order π: $\{1, ..., n\} \rightarrow \{x_1,...,x_n\}$ such that for any nonterminal vertex v, $\text{index}(\text{low}(v)) = \pi(k)$ with $k > \pi^{-1}(\text{index}(v))$ and $\text{index}(\text{high}(v)) = \pi(q)$ with $q > \pi^{-1}(\text{index}(v))$ holds if $\text{low}(v)$ $\text{high}(v)$ are also nonterminal vertices.
2. The BDD is *reduced*, i.e., there exists no $v \in V$ with $low(v) = high(v)$, and there are no two vertices v and v' such that the sub-BDD's rooted by v and v' are isomorphic.

Since we work only with ROBDD's in the following, we briefly call them BDD's. BDD is a canonical representation of boolean functions. It represents many common encountered functions in reasonable sizes. The time complexity of any single operation is bounded by the product of the related function graph sizes [2].

There are no redundant vertices and no isomorphic subtrees in a BDD and the variables appear in the same order along each path from the root to a terminal vertex, which means ordered graph. Given an ordering, the reduced graph for a function is unique. Hence the BDD is a canonical form.

The size of a BDD is largely influenced by the choice of the variable ordering. For an n-bit comparator: for order $a_1 < b_1 < ... < a_n < b_n$, the number of vertices is $3n+2$. For order $a_1 < ... < a_n < b_1 ... < b_n$, the number of vertices is $(3*2^n-1)!$. The problem of finding the optimal variable order is NP-complete [27]. Some Boolean functions have exponential size BDDs for any order (e.g., multiplier) [19].

Node ordering plays a significant role in determining the size of BDDs and affects their efficient manipulation. In this paper we study the dynamic ordering heuristics by a robust scatter search algorithm to minimize the BDD graph size.

3. Scatter Search

Scatter search operates on a set of solutions, the reference set, by combining these solutions to create new ones. The main mechanism for combining solutions is such that a new solution is created from the linear combination of two other solutions. Unlike a "population" in genetic algorithms, the reference set of solutions in scatter search tends to be small. In genetic algorithms, two solutions are randomly chosen from the population and a "crossover" or combination mechanism is applied to generate one or more offspring. A typical population size in a genetic algorithm consists of 100 elements, which are randomly sampled to create combinations. In contrast, scatter search chooses two or more elements of the reference set in a *systematic* way with the purpose of creating new solutions. Since the combination process considers at least all pairs of solutions in the reference set, there is a practical need for keeping the cardinality of the set small. Typically, the reference set in scatter search has 20 solutions or less.

Scatter search is a very aggressive search method that attempts to find high quality solutions fast. The search may be sketched as follows:

Stage 1: Generate a starting set of solution vectors to guarantee a critical level of diversity and apply heuristic processes designed for the problem. Designate a subset of the best vectors to be reference solutions.

Stage 2: Create new solutions consisting of structured combinations of subsets of the current reference solutions. The structured combinations are: (a) Chosen to produce points both inside and outside the convex regions spanned by the reference solutions. (b) Modified to yield acceptable solutions.

Stage 3: Apply the heuristic processes used in Stage 1 to improve the solutions created in Stage 2.

Stage 4: Extract a collection of the "best" improved solutions from Stage 3 and add them to the reference set. Repeat Stages 2, 3 and 4 until the reference set does not change. Diversify the reference set, by restarting from Stage 1. Stop when reaching a specified iteration limit.

The two fundamental features of the scatter search methodology are:

- Useful information about the form (or location) of optimal solutions is typically contained in a suitably *diverse collection of elite solutions*.
- Constructing combinations that extrapolate beyond the regions spanned by the solutions considered, incorporating both diversity and quality. Taking account of *multiple solutions simultaneously*, as a foundation for creating combinations, enhances the opportunity to exploit information contained in the union of elite solutions.

4. BDD Minimization by Scatter Search

Given N variables and a set of functions represented by BDDs, we want to find the optimal variable ordering (permutation of variables) that minimizes the number of BDD nodes that represent these functions. As mentioned above, scatter search process consists of diversification generation, improvement, reference set update, subset

generation and solution combination. In what follows, we discuss the implementation details of each procedure in solving the BDD minimization problem.

4.1. Reference Set Update

The idea of maintaining reference sets in Scatter Search is to maintain both quality and diversity. We partition the reference set *RefSet* into two subsets: $RefSet_1$ and $RefSet_2$. $RefSet_1$ is the high quality subset of size b_1, it contains the b_1 orders so far encountered with the smallest BDD size. $RefSet_2$ is the high diversity subset of size b_2, it contains the b_2 orders so far encountered that do not qualify for $RefSet_1$ but with the largest distance. In a reference set update, new variable orders are first considered for membership in $RefSet_1$. If not, they are considered for membership in $RefSet_2$. Quality is defined in terms of BDD size. The smaller the number of nodes in BDDs, the better the quality. Diversity is calculated in terms of distance. Let $d(x, y)$ be the distance of two variable orders. The degree of diversity is defined in terms of minimum distance $d_{min}(x) = \min\{d(x, y) | \forall y \in RefSet_1 \cup RefSet_2\}$. Thus, each order in $RefSet_2$ has the largest minimum distance from any other order in the reference set.

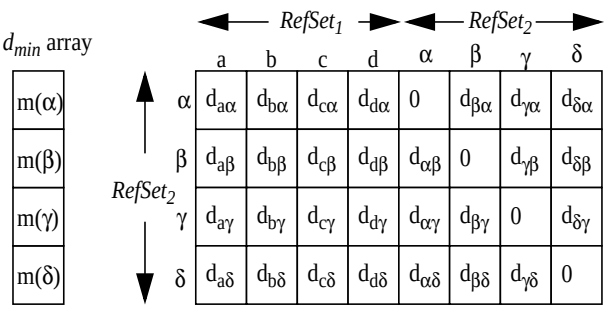

Figure 1. Dependency matrix for computing distances.

To calculate the distance between two variable orders, we consider the difference in positions between two orders with regard to the same variable. Given variable orders *x* and *y*, let the position of variable *v* in *x* and *y* be $p_x(v)$ and $p_y(v)$, respectively. Therefore, the variable position difference is $\delta(v) = |p_x(v) - p_y(v)|$. The distance between *x* and *y* is the sum of all variable position differences:

$$d(x, y) = \sum_v \delta(v) = \sum_v |p_x(v) - p_y(v)|$$

Since $RefSet_2$ is defined in terms of distances between every element in $RefSet_2$ and every other element in $RefSet_1 \cup RefSet_2$, this calculation is potentially expensive. We use a distance matrix to simplify this problem. The idea is to maintain a matrix $d(x, y)$ for all $x \in RefSet_1 \cup RefSet_2$, $y \in RefSet_2$. In addition, we maintain an array $m(x) = d_{min}(x)$ for all $x \in RefSet_2$. If order $\alpha \in RefSet_1$ is updated, we only need to update the row $\{d(x, \alpha) | \forall x \in RefSet_2\}$ and $\{m(x) | \forall x \in RefSet_2\}$. If order $\beta \in RefSet_2$ is updated, we need to update the column $\{d(\beta, y) | \forall y \in RefSet_1 \cup RefSet_2\}$ and the row $\{d(x, \beta) | \forall x \in RefSet_2\}$ and $\{m(x) | \forall x \in RefSet_2\}$.

An example is given in Figure 1, where $RefSet_1 = \{a, b, c, d\}$ and $RefSet_2 = \{\alpha, \beta, \gamma, \delta\}$. Since $d(x, y) = d(y, x)$, the right side of the matrix is symmetric along the diagonal line. Suppose $\|RefSet_1\| = \|RefSet_2\| = c$, the computational complexity for all distances without the matrix is $O(\|RefSet_1 \cup RefSet_2\| \cdot \|RefSet_2\|) = O(c^2)$, with the matrix, it is reduced to $O(\|RefSet_1 \cup RefSet_2\|) = O(c)$.

4.2. Diversification

The BDD variable reordering problem can viewed as a permutation problem. We apply the permutation diversification generator in [12] to our *DGP*. As a starting point, let the initial order of BDD variables be a sequence $P = (1,2,...,n)$. Define the subsequence $P(h:s) = (s, s+h, s+2h, ..., s+rh)$, where *h* and *s* are integers, $h \le n$, $1 \le s \le h$, and *r* is the largest non-negative integer such that $s + rh \le n$. Then we define the permuted sequence $P(h) = (P(h:h), P(h:h-1), ..., P(h:1))$. It is shown in [12] that when *h* approaches $\sqrt{n}$, the minimum relative separation of each element from each other element in $P(h)$ is maximized. The diversification generator would generate a set of $P(h)$, for $h = 1, ..., n$ (excluding $P(1) = P$).

4.3. Improvement

In Scatter Search, *Improvement* is to transform a trial solution into one or more enhanced trial solutions. If no improvement can be made, the "enhanced" solution is considered to be the same as the input solution. In our implementation we used the sifting [23] algorithm. The algorithm allows us to take advantage of existing heuristics specifically tailored to the reordering of BDD variables. It is possible for a tabu style improvement heuristic to generate better result, but the sifting heuristic has been well established and robust enough for our implementation.

4.4. Subset Generation

Subset Generation operates on the reference set to produce a subset of its solutions as a basis for creating combined solutions. Given the high quality and high diversity reference set, scatter search generates the following subsets: (i) all 2-element subsets; (ii) 3-element subsets derived from 2-element subsets by augmenting each 2-element subset to include the best order (minimum BDD size) not in this subset; (iii) 4-element subsets derived from 3-element subsets by augmenting each 3-element subset to

include the best order (minimum BDD size) not in this subset; (iv) The subsets consisting of the best *i* elements for $i = 5$ to $(b_1 + b_2)$. The variable orders in each subset are combined to generate new variable order.

4.5. Combining Variable Orders

Our strategy on combining variable orders is based on the notion of variable movements. Given variable orders *x* and *y*, let the position of variable *v* in *x* and *y* be $p_x(v)$ and $p_y(v)$, respectively. Let *x'* be an auxiliary variable order derived from *x* by moving the variable *v* up or down in the variable order towards its corresponding position in *y*. We denote the position of *v* in *x'* as $p_{x'}(v)$. Similarly, let *y'* be an auxiliary variable order derived from *y* by moving the variable *v* towards its corresponding position in *x*, with position of *v* in *y'* as $p_{y'}(v)$. There is a mid-point where the variable *v* position meets: $p_{x'}(v) = p_{y'}(v)$. To do this, we construct a weight function for the variable $w(v) = p_x(v) + p_y(v)$. It is obvious that the average weight $w(v)/2 = [p_x(v) + p_y(v)]/2$ is essentially the mid-point position between the two orders. We now generalize our solution combination method for more than 2 variable orders. Given *n* variable orders: $x_1, x_2, ..., x_n$. Construct a weight function for each variable *v*:

$$w(v) = p_{x_1}(v) + p_{x_2}(v) + \ldots + p_{x_n}(v)$$

After we have computed the weights for all variables, the combined variable order is generated by sorting the variables in ascending order of their weights. Notice that the average weight

$$\frac{w(v)}{n} = \frac{p_{x_1}(v) + p_{x_2}(v) + \ldots + p_{x_n}(v)}{n}$$

is actually the center of gravity of that variable *v* among the *n* orders.

For variables whose positions remain the same among the *n* orders, their average weights are the same as their variable positions. For variables who changed position between orders, their average weights serve as compromised balances among their various positions. It is possible for several variables to end up with the same weight. This is often the case when two variables exchanged positions between two orders. When they try to move towards each other, their centers of gravity collide together. For this, we introduce a historical weight accumulator for each variable. When two or more variables came up with the same weight, we compare their historical weights and sort them in descending order (as opposed to the general ascending order). By doing this, we try to rebalance the average positions for each variable and allow historically unfavorable variables to gain more significance.

The combination of variable orders is so far based on equal weights among all orders. Besides historical adjustments, we can also increase the weight for some particular orders. Thus making the combined result more biased towards those favorable orders. When chosen correctly, this bias would guide the search to arrive at better results faster. However, the bias may also lead to more emphasis on local minimas. One of the features of scatter search is to find a balance between diversity and quality. To avoid offsetting this balance, we implemented our average weight function without any special bias.

4.6. Improving Quality in Casual Encounters

Although our solution combination method uses average weights that exploit locality of the solutions, the diversification generator and the high-diversity reference set aimed to project solutions outside the convex region of our best solutions. Our search routine need to rearrange the BDD's according to the new variable order. This rearrangement is done by swapping adjacent variables. In order to reach those diversified solutions, our rearrangement routine has to go through many variable orders along the way. These transient orders along the way sometimes have smaller BDD sizes than we have ever encountered, they are potential candidates for the high-quality reference set. However, when inserting these transient solutions to the high-quality reference set, we need to avoid two pit-falls:

1. Similar variable orders may have similar BDD sizes. If our search traversal go through a local minima, there may be several transient orders that qualify to be inserted to our high-quality reference set.
2. Our search could have just departed from a local minima that is already included in the high-quality reference set, so we may pick up several transient orders in our departure neighborhood that still qualify to be included in the high-quality reference set.

So, for each transient variable order, we compare its BDD size with our best solution so far, if it turns out to be smaller, we record that transient order and its BDD size. If there are multiple transient orders that qualify, we only pick the best (smallest BDD size) to be inserted.

5. Experimental Results

We have implemented our algorithm on the CUBDD package [26]. We tested the performance on Linux using 850MHz Pentium®III and 1GB RDRAM.

Our first experiment compares the CPU run time (in seconds) and BDD nodes between Scatter Search and the Exact Algorithm [15][9][16]. Since the Exact Algorithm is limited to a small number of BDD variables, we limit our experiments to some circuits with 17 or less inputs from the

LGSynth93 benchmark. The results are shown in Table 1. The *orig* column shows the original BDD size before reordering. The exact time, scatter time, exact nodes, scatter nodes columns refers to run time and final BDD size for exact algorithm and scatter search respectively. For run time, the exact algorithm quickly blows up as the number of inputs (BDD variables) increases, whereas our scatter search run time is several orders of magnitude smaller than those of the exact algorithm. Regarding the reduced BDD size, our scatter search algorithm is as good as the exact algorithm for the problem size that is runnable by the exact algorithm. For larger circuits, scatter search may not perform as much reduction as the exact algorithm. That is the trade off between quality and speed. So far, the results are in favor of scatter search.

Table 1. Runtime comparison between Scatter Search and Exact Algorithm

Circuit	inputs	orig	exact time	scatter time	exact nodes	scatter nodes
cm82a	5	16	0.01	0.02	12	12
z4ml	7	50	0.02	1.41	17	17
con1	7	20	0.03	1.5	15	15
inc	7	108	0.06	2.47	72	72
misex1	8	71	0.10	2.21	35	35
clip	9	180	0.28	3.28	75	75
sao2	10	117	0.60	4.21	81	81
cm85a	11	38	0.20	3.2	28	28
cm162a	14	67	7.47	4.93	30	30
cu	14	59	16.73	4.84	32	32
alu4	14	1453	31.57	29.99	564	564
table3	14	1972	62.76	28.74	751	751
b12	15	85	52.02	9.9	55	55
cm163a	16	55	13.41	5.21	26	26
pdc	16	4087	447.68	26.26	793	793
spla	16	1204	665.56	30.94	583	583
tcon	17	34	1389.75	0.81	25	25
vda	17	1259	1841.15	30.17	478	478
Total	**217**	**10875**	**4529.40**	**190.09**	**3672**	**3672**

Our second experiment compared scatter search with other heuristics that came with the CUBDD package. Again, we run these tests on circuits from the *LGSynth93* benchmark. The results are listed in Table 2. The *orig* column contains the original BDD size before reordering. The rest of the columns contains reordered BDD sizes after applying reordering methods using: scatter search, genetic algorithm [5], simulated annealing [1], sifting [23], convergent sifting, symmetric sifting [22], group sifting [21], window permutation approach of Fujita [11] and Ishiura [15] with sizes 2, 3, and 4, respectively. In all these experiments, we obtained smallest BDD sizes after reordering with scatter search. Notice that genetic algorithms and simulated annealing also resulted in fairly small BDD sizes. They are both population based optimization that bears some similarity with scatter search. Unlike genetic algorithms, which tries to mimic a natural process of improvement, scatter search incorporates a systematic intensification and diversification that depends less on the randomness of nature. For large circuit with big BDDs like *C1908*, the advantage of scatter search over other methods becomes quite significant.

6. Conclusion

Scatter Search is a powerful optimization search technique. In this paper we have investigated BDD minimization by scatter search. The experiment demonstrates the efficiency of the approach in comparison with simulated annealing, genetic agorithm and various other methods. It is our belief that by adding other heuristics, the performance can be further improved and would be at par with any other methods. Our work is the first successful experience of using scatter search approach in design automation area.

References

[1] B. Bollig, M. Löbbing, and I. Wegener, "Simulated annealing to improve variable orderings for OBDDs," *International Workshop on Logic Synthesis*, Granlibakken, CA, May 1995.

[2] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*. 35(8), August, 1986

[3] N. Calazans, Q. Zhang, R. Jacobi, B. Yernaux, and A.M. Trullemans, "Advanced Ordering and Manipulation Techniques for Binary Decision Diagrams," *Proc. of the 29th ACM/IEEE Design Automation Conference*, June 1992.

[4] V. Campos, Manuel Laguna and R. Martí, "Scatter Search for the Linear Ordering Problem," *New Ideas in Optimization*, D. Corne, M. Dorigo and F. Glover (Eds.), McGraw-Hill, 331-339 (1999).

[5] R. Drechsler, B. Becker, and N. Göckel, "A genetic algorithm for variable ordering of OBDDs," *International Workshop on Logic Synthesis*, Granlibakken, CA, May 1995.

[6] R. Drechsler, N. Drechsler, and W. Gunther, "Fast exact minimization of BDD's," *IEEE Transactions on CAD*, 19(3), 384 -389, 2000.

[7] E. Felt, G. York, R. Brayton, and A. Sangiovanni-Vincentelli, "Dynamic variable reordering for BDD minimization," *Proc. EURO-DAC '93*, 130 -135.

[8] R. Forth and P. Molitor, "An efficient heuristic for state encoding minimizing the BDD representations of the transition relations of finite state machines," *Proc. of the ASP-DAC* 2000, 61-66.

[9] S. J. Friedman and K. J. Supowit, "Finding the optimal variable ordering for binary decision diagrams," *IEEE Transactions on Computers*, 39(5), 710-713, May 1990.

[10] M. Fujita, H. Fujisawa, and Y. Matsunaga, "Variable Ordering Algorithms for Ordered Binary Decision Diagrams and their Evaluation," *IEEE Transactions on CAD*, January 1993.

[11] M. Fujita, Y. Matsunaga, and T. Kakuda, "On Variable Ordering of Binary Decision Diagrams for the Application of Multi-level Synthesis," *Proc. of European Design Automation Conference*, March 1991

[12] Fred Glover, "A Template for Scatter Search and Path Relinking," *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.-K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.), Springer, pp. 13-54.

[13] Fred Glover, Manuel Laguna, and R. Martí, "Scatter Search," *Theory and Applications of Evolutionary Computation: Recent Trends*, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag, 2000.

[14] William Hung, Adnan Aziz, and Ken McMillan, "Heuristic Symmetry Reduction for Invariant Verification," *Proc. of International Workshop on Logic Synthesis*, Tahoe City, CA, May 1997.

[15] N. Ishiura, H. Sawada, and S. Yajima, "Minimization of Binary Decision Diagrams Based on Exchanges of Variables," *Proc. of International Conference on Computer-Aided Design*, November 1991.

[16] S. W. Jeong, T.-S. Kim, and F. Somenzi. An efficient method for optimal BDD ordering computation. *International Conference on VLSI and CAD*, Korea, 1993.

[17] Manuel Laguna, "Scatter Search," *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford Academic Press, 2000.

[18] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli, "Logic Verification using Binary Decision Diagrams in a Logic Synthesis Environment," *Proc. of International Conference on Computer-aided Design*, November 1988

[19] S. Minato. *Binary Decision Diagrams and Applications for VLSI CAD.* Kluwer Academic Publishers, 1996

[20] A. Oliveria, L. Carloni, T. Villa, and A. Sangiovanni-Vincentelli, "Exact Minimization of Binary Decision Diagrams Using Implicit Techniques," *IEEE Transactions on Computers*, 47(11), 1282-1296, November 1998.

[21] S. Panda and F. Somenzi, "Who are the variables in your neighborhood," *Proc. of the International Conference on Computer-Aided Design*, pages 74-77, San Jose, CA, November 1995.

[22] S. Panda, F. Somenzi, and B. F. Plessier, "Symmetry detection and dynamic variable ordering of decision diagrams," *Proc. of the International Conference on Computer-Aided Design*, 628-631, San Jose, CA, November 1994.

[23] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *Proc. of the International Conference on Computer-Aided Design*, 42-47, Santa Clara, CA, 1993.

[24] C. Scholl, D. Moller, P. Molitor, and R. Drechsler, "BDD minimization using symmetries," *IEEE Transactions on CAD*, 18(2), 81-100, February 1999.

[25] T. Shiple, R. Hojati, A. Sangiovanni-Vincentelli, and R. Brayton, "Heuristic Minimization of BDDs Using Don't Cares," *Proc. of Design Automation Conference*, 225-231, June 1994.

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

[27] S. Tani, K.Hamaguchi, and S.Yajima, "The Complexity of the Optimal Variable Ordering of a Shared Binary Decision Diagram," *Technical Report 93-6*, Department of Information Science, University of Tokyo, 1993.

Table 2. BDD nodes generated by Scatter Search compared with other heuristics

Circuit	inputs	orig	scatter	genetic	anneal	sifting	conv	symm	group	win2	win3	win4
alu2	10	231	154	154	154	162	158	162	163	218	211	196
apex1	45	6785	1246	1247	1248	1368	1265	1368	1425	5779	5374	4658
apex2	39	13418	302	325	303	882	475	778	848	10413	7582	5915
apex6	135	1993	517	527	551	661	636	650	678	1899	1867	1746
C1908	33	127349	5526	5708	5859	10179	6309	10179	20863	107790	95112	84724
des	256	10771	2916	2983	2986	3143	3064	3143	3424	10651	10511	10341
ex4p	128	994	439	455	473	526	474	494	507	983	970	959
frg2	143	5104	856	862	965	1706	1145	1706	1620	5047	4691	4583
i4	192	85855	213	243	216	850	253	314	479	83730	82275	78408
i8	133	10366	1276	1277	1296	1336	1278	1336	1400	8146	8089	6720
ldd	9	80	64	64	64	69	64	69	71	74	69	68
too_large	38	7083	302	308	320	726	441	530	471	5068	3213	2086
x1	51	1916	407	419	443	572	456	556	578	1663	1479	1325
x3	135	3041	506	515	562	743	604	741	630	2927	2827	2791
x4	94	1124	320	337	371	375	349	375	378	1116	881	787
Total	**1441**	**276110**	**15044**	**15424**	**15811**	**23298**	**16971**	**22401**	**33535**	**245504**	**225151**	**205307**