

Formal Presentation of Two Initial Variable Ordering Algorithms for Binary Decision Diagrams

B.I. Mills
College of Information
Technology,
United Arab Emirates
University,
UAE
brucem@uaeu.ac.ae

P.W.C. Prasad
College of Information
Technology,
United Arab Emirates
University,
UAE
prasadc@uaeu.ac.ae

Ali Assi
Department of Electrical
Engineering,
United Arab Emirates
University,
UAE
ali.assi@uaeu.ac.ae

Abstract

This paper present two variable ordering methods for minimizing Reduced Ordered Binary Decision Diagrams (ROBDDs). The broad formal techniques, based on graph topology, and the details of the algorithms are suitable for further work on other algorithms of the same family. Experiments have shown that the algorithms improve on common algorithms for the majority of benchmark circuits.

Keywords: Binary Decision Diagram, Variable Ordering, Number of Nodes, Graph Topology

1. Introduction

Many combinatorial tasks can be stated as Boolean function manipulation tasks. Efficient Boolean algorithms are of increasing importance in Very Large Scale Integration (VLSI) and Computer Aided Design (CAD). Many data types, and algorithms have been used, but most are not sufficiently efficient; there is a demand for improvement of both. In the last two decades, Binary Decision Diagrams (BDDs) have been popular in Boolean algorithms [1]. Their advantages were recognized and emphasized by Bryant [2]. Significant breakthroughs in the optimization of digital circuits have been achieved using BDDs [3].

Many BDD algorithms are graph traversals and typically take polynomial time in the size of the graph. But, BDD size can be exponential in the size of the task [3]. The choice of variable order can strongly affect the order of complexity, but determining an optimal ordering is an NP-hard problem [2], [3]. It is hard to predict the effect of variable ordering on the BDD size. It is also hard to find the best order for a given Boolean function [4], [5], [6]. Usually, the number of nodes in the BDD is directly related to the variable ordering of the BDD [5], [6], [7], [8]. There is a variety of methods to find the optimal variable ordering for BDDs but none can fulfill both the time and the space requirements.

The ROBDD is a specialization, in widespread use, of the BDD. Researchers are actively involved in finding algorithms that determine a variable order that minimizes the number of nodes in the ROBDD. All these variable ordering techniques fall into two categories: Static [9] and Dynamic [7], [10]. In a BDD, each path from the root node to the terminal node corresponds to a cube (implicant) of the function. Reduction of the number of paths means reduction of the number of cubes. This also can be done by finding an optimal order of the BDD variables [4]. For a given function, the primary requirement to minimize time and space complexity is to represent its BDD with minimum number of nodes. [5], [6]

The main objective of this paper is to provide a formal presentation to the method proposed in [11] for BDD minimization using graphs parameter permutation. The rest of this paper is organized as follows: In section two and section three, background information pertaining to the graph theory and decision diagram theory is discussed. In section four, the new variable ordering technique based on graph parameters is introduced and the experimental results are given in section five. Finally in section six we conclude our paper.

2. Graph Theory Clarification

A decision diagram is defined here as a species of directed graph. There are several distinct definitions of directed graph in the literature. To be clear we briefly summarize our choices, but do not attempt to develop an intuition (assumed in this paper) for graph theory [12].

Definition 1: A directed graph is a tuple (V, E, s, d) , where V and E are disjoint sets, and $s, d \in E \rightarrow V$.

Definition 2: A rooted directed graph is (v, V, E, s, d) where (V, E, s, d) is a directed graph, and $v \in V$.

Definition 3: A directed graph with weights in W is (V, E, s, d, w, W) where $w \in (V \cup E) \rightarrow W$.

Definition 4: A path of length n in a directed graph (V, E, s, d) is a tuple $(e_1, \dots, e_n) \in E^n$, where $d(e_i) = s(e_{i+1})$ for $i = 1..n-1$. The source node of the path is $s(e_1)$. The sink node of the path is $d(e_n)$.

The term *graph* will be used interchangeably with *directed graph*. An acyclic graph has no path with the same node for both source and destination. An exit for a node is an edge with that node as its source. A deterministic node has no two exits with the same weight. A graph is deterministic if and only if each node is deterministic. The concepts combine in the natural manner, so that paths are defined in weighted and rooted graphs, and a rooted weighted graph is well defined.

The graph edges have a string-and-beads feel. Each edge e has a two ends but there can be many edges with given ends. The root is a place to start traversals in the graph. The node weight is a value to pick up at each node as we traverse the graph; likewise for the edge weight.

3. Decision Diagram Theory

Definitions of BDDs exist in standard sources [1], [2], [8]. We state them in the context of the given graph theory. Intuitively, a decision graph determines the value of a multi-valued function f given an input tuple (a_1, \dots, a_n) . Start at the root then repeatedly find the index i of the node and leave by the exit with label a_i , until at a node with no exits we find the return value. A_i is the set of possible values of the i^{th} argument. B is the set of return values. A decision graph is deterministic or complete exactly when every node is likewise.

Definition 5: The tuple $(v, V, E, s, d, w, W, A, I, B)$, is a decision graph if (v, V, E, s, d, w, W) is a rooted weighted directed graph, B is a set, A is a family of sets with index set I , $W = I \cup B \cup \bigcup_{i \in I} A_i$, and if u is a non terminal node then $w(u) \in I$, if u is a terminal node then $w(u) \in B$, and if e is an edge, then $w(e) \in A_{w(s(e))}$.

Definition 6: A node v is complete if every element of $A_{w(v)}$ occurs on at least one exit from v .

Definition 7: A node v is deterministic if every element of $A_{w(v)}$ occurs on at most one exit from v .

Definition 8: An A -valued tuple over I is a map $t \in I \rightarrow \bigcup A_i$ such that $t(i) \in A_i$.

Definition 9: A path (e_1, \dots, e_n) in a decision graph is valid for a tuple t over I if $w(e_i) = t(w(s(e_i)))$

Definition 10: For each node u , the relation f_u between A -valued tuples over I and B is defined by $(t, b) \in f_u$ if there is a valid path for t starting at u and ending in a terminal node v such that $w(v) = b$.

Definition 11: In a reduced decision graph $f_u = f_v$ implies $u = v$.

Definition 12: A decision diagram is a deterministic complete acyclic decision graph. A binary decision diagram with n variables has $I = [1, \dots, n]$ and $B = A_i = \{0, 1\}$. In a decision diagram the node functions f_u are single valued. If u is a terminal node then $f_u(a) = w(u)$, if u is a non terminal node then $f_u(a) = f_{child(u, a_{w(u)})}(a)$.

Definition 13: In an ordered decision diagram all source to sink paths encounter the variables in the same order.

It follows that in a Reduced Ordered Binary Decision Diagram (ROBDD), there are no distinct isomorphic Sub-diagrams and $child(u, x) = child(u, y)$ implies $x = y$. Defects (deviation from ROBDD status) can be eliminated; for example, given u such that $child(u, 1) = child(u, 0) = v$, for each edge (v, x) replace it with the edge (u, x) . Other defects can be handled similarly.

3.1 Variable Ordering

The size of a BDD is largely affected by the choice of the variable ordering. This is illustrated by the following example:

Example: Let $f = x_1 \cdot x_2 + \dots + x_{2n-1} \cdot x_{2n}$. If the variable ordering is given by (x_1, x_2, \dots, x_n) , i.e. $\forall i: \pi(i) = x_i$, the size of the resulting BDD is $2n$. On the other hand, using the variable ordering $(x_1, x_3, \dots, x_{2n-1}, x_2, x_4, \dots, x_{2n})$, the size of the BDD is $\theta(2^n)$. Thus, the number of nodes in the graph varies from linear to exponential depending on the variable ordering chosen. Each node has the index of a variable, and links are directed downward. Each node has one 0-exit and one 1-exit, indicated by dashed and solid lines respectively. Fig. 1 shows the effect of the variable ordering on the size of BDDs for the following function (1):

$$f = x_1 \cdot x_2 + x_1 \cdot \overline{x_2} \cdot x_3 \cdot x_4 + x_1 \cdot \overline{x_3} \cdot x_4 \quad (1)$$

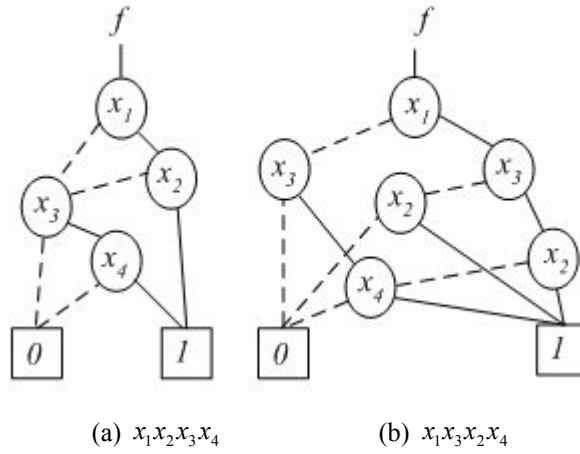


Fig. 1. Effect of the variable ordering on the size of BDDs

4. Proposed algorithms

The algorithms use topological parameters of a graph derived from the given expression [11]. Different expressions for the same function could lead to different graphs, parameters, and orders. The Boolean operators are replaced with equivalents using only AND and NOT operators [11].

4.1 Graph Parameters of (an expression for) a function

An m -output n -input function f is represented by an m -tuple (f_1, \dots, f_m) of nested terms, either primitive constants, or of the form $b(t_1, \dots, t_n)$, where b is the name of a Boolean operator. We recast this as a graph: the nodes are the terms, edges are of the form $(t_i, b(t_1, \dots, t_n))$, leading from simpler expressions to more complex. For any finite expression this graph is directed acyclic. So, the parameters are identical to those of the parse tree.

$NP(i)$ is the total number of paths from x_i to any f_j , is the number of source nodes weighted with x_i and the number of times the literal x_i appears in the expression.

$TNN(i)$ is the sum of the lengths of paths from x_i to f_j . Each source node containing x_i has weight 1, each other source node has weight 0, and each non-source node has the sum of its children's weights. In a sum-of-products expression, each x_i scores 4 and each $\overline{x_i}$ scores 5. The total is $TNN(i)$.

$MNNAP(i)$ = maximum of the lengths of paths from x_i to f_j . Each source node containing x_i has weight 1, each other source node has weight 0, and each non-source node has 1 plus the maximum of its children's weights. In a sum-of products expression, each x_i scores 4, each \bar{x}_i scores 5. The maximum is $MNNAP(i)$.

$SP(i, j) = SP(j, i)$ = length of shortest path from x_i to x_j in the undirected graph. For sum of products expressions: eliminate all other literals. The remaining terms score $x \cdot y = 2, x \cdot \bar{y} = \bar{x} \cdot y = 3, \bar{x} \cdot \bar{y} = 4, x + y = 6, x + \bar{y} = 7, \bar{x} + y = 7, \text{ and } \bar{x} + \bar{y} = 8$. Score 2 plus the number of bars plus 4 for distinct terms. Take the minimum over the available sub expressions to obtain $SP(i, j)$.

$SSP(i) = \text{Sum}(j) SP(i, j)$ is the sum of all the shortest paths from x_i to the other variables.

4.2 Level-I Algorithm

In this Level, the variable order is found based on the Number of Paths (NP), the Number of Nodes (NN) and the Maximum Number of Nodes among All Paths ($MNNAP$) per variable in the graph. Since the variable with the highest number of nodes has the greatest effect on the circuit, it is placed first in the variable ordering. The proposed algorithm is explained in the following:

Example:

Consider the following Boolean function (2):

$$f = \bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_4 + \bar{x}_1 \cdot x_4 + x_2 \cdot \bar{x}_4 + x_4 \cdot x_2 \cdot x_3 \quad (2)$$

This Boolean function that contains AND, OR and NOT operations, is first converted into an equivalent Boolean function (3) with only AND and NOT operations:

$$f = (\bar{x}_1 \cdot \bar{x}_2 \cdot \bar{x}_4) \cdot (\bar{x}_1 \cdot x_4) \cdot (x_2 \cdot \bar{x}_4) \cdot (x_4 \cdot x_2 \cdot x_3) \quad (3)$$

Fig. 2 shows the graph of the function f where each operation is represented as a node. Nodes x_1, x_2, x_3 and x_4 are the input nodes and node 13 is the output node. Nodes 1, 3, 4, 6, 7, 9, 11 and 13 represent NOT operations and nodes 2, 5, 8, 10, and 12 represent AND operations. In this graph, the NP , TNN , and $MNNAP$ are given in Table 1.

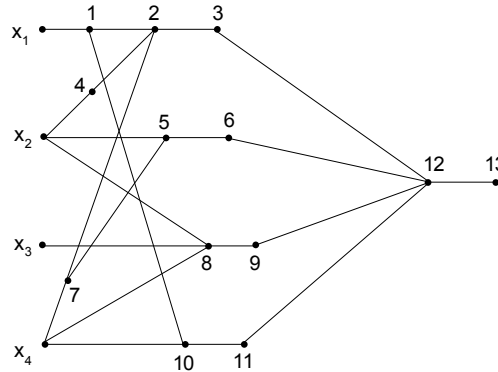


Fig. 2. Graph representation for function (3)

According to this table, the algorithm selects (x_4, x_2, x_1, x_3) as BDD variable ordering, which is the descending order of TNN . The algorithm will not compare NN or $MNNAP$ since there is no equal TNN .

Table 1. Graphical Parameters for equation (3)

Graphical Parameter	X_1	X_2	X_3	X_4
Number of different paths (NP)	2	3	1	4
Total number of nodes among all paths (TNN)	10	13	4	18
Max number of nodes among all paths (MNNAP)	5	5	4	5

4.3 Level II Algorithm

This variable ordering algorithm is an improved version of the Level I algorithm explained in Section 4.2. In addition to the three parameters *TNN*, *NP* and *MNNAP*, the algorithm uses two more parameters, which are the shortest path (*SP*) between every pair of variables and sum of the shortest paths (*SSP*). *SP* is considered as primary parameter and *NP*, *TNN*, *SSP* and *MNNAP* are considered as secondary parameters. The variables are then sorted in descending order according to secondary parameters. An arbitrary selection is made if two or more variables have equal parameters for *NP*, *TNN*, *SSP* and *MNNAP*. The first variable in the sorted list is considered the first variable in the order. The rest of the variables are selected according to the minimum value of *SP*. If two or more variables have an equal minimum value of *SP*, then preference is given to the variable with the higher value of secondary parameters: *NP*, *TNN*, *SSP* and *MNNAP*. For the obtained variable ordering we note the number of nodes of the BDD. For each permutation p of (*NP*, *TNN*, *SSP*, *MNNAP*), sort according to (p , *SP*), and construct the ROBDD. Select an ordering that generates the minimum number of nodes.

Example: Consider the following Boolean function (4):

$$f = x_1 \cdot x_2 \cdot \overline{x_4} + \overline{x_1} \cdot x_3 \cdot \overline{x_4} + \overline{x_1} \cdot x_4 + x_2 \cdot \overline{x_4} \quad (4)$$

This Boolean function (4) is converted into an equivalent Boolean function with only AND and NOT operations. Fig. 3 shows the graph of the converted function.

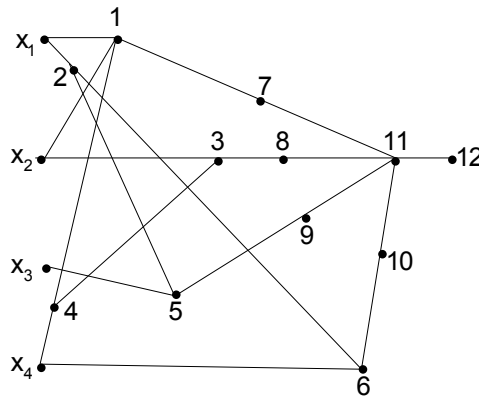


Fig. 3. Graph representation for function (6)

Table 2 summarizes the *SP* of this graph. *NP*, *NN*, *SSP*, and *MNNAP* are summarized in Table 3. The number of nodes for all possible parameter permutations p is recorded and x_4, x_2, x_3, x_1 is considered as the final variable ordering for this method.

Table 2. Shortest Path Table

	x_1	x_2	x_3	x_4
$x_1 \cdot x_1$	0	2	3	3
$x_2 \cdot x_2$	2	0	6	3
$x_3 \cdot x_3$	3	6	0	3
$x_4 \cdot x_4$	3	3	3	0

Table 3. Graphical Parameters for equation (6)

Parameters	x_1	x_2	x_3	x_4
<i>TNN</i>	14	8	4	19
<i>NP</i>	3	2	1	4
<i>MNNAP</i>	5	4	4	5
<i>SSP</i>	8	11	12	9

Both levels are use to produce the best variable ordering sequence found for the given Boolean function.

5. Experimental Results

In this section we present the experimental results obtained by applying the proposed two-level permutation method to selected ISCAS benchmark circuits using the Colorado University Decision Diagram (CUDD) Package.[13] A Pentium IV machine with 512 MB RAM was used. We have selected a larger collection of ISCAS benchmark circuits [14] to demonstrate the performance of the proposed method. Tables 4 summarize a comparison of our results to the best results obtained by three different CUDD variable reordering methods.

Table 4. Comparison of CUDD and Proposed method results for selected benchmark circuits

Benchmark Circuits			Number of Nodes								
			CUDD Methods			Proposed Algorithms	Minimum number of Nodes	Gain Factor against Miminun			
Name	Input	Output	Random swapping	Symmetric Sift	Window Permutatio			Random swapping	Symmetric Sift	Window Permuta	Proposed Algorithm
Alu2	11	6	247	199	251	198	198	0.80	0.99	0.79	1.00
Apex4	9	19	1410	1452	1583	1312	1312	0.93	0.90	0.83	1.00
Con1	8	2	17	18	19	17	17	1.00	0.94	0.89	1.00
Cu	14	22	69	75	64	66	64	0.93	0.85	1.00	0.97
Misex1	8	7	81	87	78	73	73	0.90	0.84	0.94	1.00
Sao2	10	4	117	113	171	104	104	0.89	0.92	0.61	1.00
Sqrt8	8	4	33	30	34	33	30	0.91	1.00	0.88	0.91
F51m	14	8	61	66	68	57	57	0.93	0.86	0.84	1.00
X2	10	7	53	60	68	53	53	1.00	0.88	0.78	1.00
Sct	19	15	110	106	109	109	106	0.96	1.00	0.97	0.97
Pm1	16	13	85	85	88	85	85	1.00	1.00	0.97	1.00
Tcon	17	16	56	55	56	48	48	0.86	0.87	0.86	1.00
B12	16	9	105	94	111	103	94	0.90	1.00	0.85	0.91
Apex6	143	139	1413	1056	2252	1028	1028	0.73	0.97	0.46	1.00
Apex7	48	37	910	596	1672	678	596	0.65	1.00	0.36	0.88
Apex1	45	45	2850	2502	7232	2788	2502	0.88	1.00	0.35	0.90
Cc	21	20	126	124	123	100	100	0.79	0.81	0.81	1.00
Cht	47	36	255	217	271	190	190	0.75	0.88	0.70	1.00
Comp	32	3	393	294	8981	144	144	0.37	0.49	0.02	1.00
C8	28	18	136	139	136	136	136	1.00	0.98	1.00	1.00
C880	60	26	33817	15857	464256	9333	9333	0.28	0.59	0.02	1.00
C432	36	7	3705	2470	13360	1933	1933	0.52	0.78	0.14	1.00
C499	41	32	125352	60434	144556	63823	60434	0.48	1.00	0.42	0.95
C1355	41	32	154789	64075	169382	101455	64075	0.41	1.00	0.38	0.63
C1908	33	25	27560	18062	84541	23762	18062	0.66	1.00	0.21	0.76
C3540	50	22	62500	44402	3129412	45903	44402	0.71	1.00	0.01	0.97
Misex2	25	18	184	177	178	119	119	0.65	0.67	0.67	1.00
My_adder	33	17	856	681	66155	474	474	0.55	0.70	0.01	1.00
I1	26	13	82	74	83	71	71	0.87	0.96	0.86	1.00
I6	138	67	408	433	440	410	408	1.00	0.94	0.93	1.00
I7	199	67	510	535	666	512	510	1.00	0.95	0.77	1.00
Gain factor in %								19.35	32.26	6.45	67.74

In Table 4 column 1 shows the ISCAS benchmark names and the input and output counts are given in column 2 and 3 respectively. Columns 4 to 6 show the number of nodes required for the construction of the ROBDD using three of the best CUDD methods (Random Swapping, Symmetric Sifting and Window Permutation). The results of the proposed method and the minimum number of nodes among all four methods (i.e. the minimum number of nodes shown in columns 4, 5, 6, and 7 for a given benchmark circuit) are given in column 7 and 8 respectively. The gain factors of each method against the method that gives the minimum numbers of nodes are given in columns 9, 10, and 11.

The results shown in Table 4 indicate that the proposed algorithm gain decreases the number of nodes in more than 68% of the benchmark circuit's comparison with 19%, 32% and 6% for Random Swapping, Symmetric Sift and Window Permutation respectively. It is clear that circuits Apex4, Sao2, Cc, Cht, comp, C880, C432, Misex1, Misex2, My_adder benefit the most using the proposed algorithms compared to the other three CUDD methods.

Using the proposed method gives a higher probability of achieving the minimum number of nodes for most of the benchmark circuits. The number of nodes in BDDs is directly related to the space complexity of the circuit design. So the above results prove that the proposed method minimizes the space complexity of the circuit, which will eventually minimize the cost of the design.

6. Conclusion

Sorting according to a variety of graph parameters has been illustrated as the foundation of two variable ordering algorithms for minimizing ROBDDs. The principle can be used for several types of minimization, including node count or minimum path length. The algorithms can handle multiple output minimizations. The algorithms have been implemented and tested using ISCAS benchmark circuits and compared with selected CUDD reordering methods. The algorithms were shown to perform overall better than the other selected methods. The discussion highlights a family of graph-based algorithms that are worth exploring further for other useful instances. This family of algorithm is a promising alternative to existing reordering methods to reduce the number of nodes in BDD. Our on-going work will address additional types of larger scale benchmarks, justify the algorithm family, and develop theoretical performance indicators.

References

- [1] Akers, S. B., "Binary Decision Diagram," IEEE Trans. Computers. Vol. 27, 1978, pp. 509-516.
- [2] Bryant, R. E., "Graph-Based Algorithm for Boolean Function Manipulation," IEEE Trans. Computers, vol. 35, 1986, pp. 677-691.
- [3] Harlow, J. E., and Brglez, F., "Design of Experiments and evaluation of BDD ordering Heuristics," Inter. Journal on Software tools for Technology Transfer, vol. 3, 2001, pp. 193-206.
- [4] Gorschwin, F., and Drechsler, R., "Minimizing the Number of paths in BDDs," in Proceedings of 15th symposium on Integrated Circuits and Systems Design, pp. 359-364. 2002.
- [5] Ebendt, R., "Reducing the number of variable movements in exact BDD minimization," in Proceedings of 2003 Int. Symp. on Circuits and Systems, pp. 605-608, 2003.
- [6] Ebendt, R., et. al., "Combination of lower bounds in exact BDD minimization," in Proceedings of Design Automation and Test in Europe Conf. and Exhibition, pp. 758-763, 2003.
- [7] Rudell, R., "Dynamic Variable ordering for ordered binary decision diagrams," Proceedings of IEEE Inter. Conf. on CAD, pp. 42-47, 1993.
- [8] Drechsler, R., and Sieling, D., "Binary Decision Diagrams in Theory and Practice," Springer-Verlag Trans., 2001, pp. 112-136.
- [9] Malik, A., et. al., "Logic Verification Using Binary Decision Diagrams in a Logic Synthesis Environment," in Proceedings of the International Conference on Computer Aided Design (ICCAD), pp. 6-9, 1988.
- [10] Somenzi, F., "Efficient Manipulation of Decision Diagrams," in International Journal on Software Tools for Technology Transfer (STTT), Vol. 3, 2001, pp. 171-181.
- [11] Prasad, P.W.C., et. al., "BDD Minimization Using Graph Parameter Permutation", Proceedings of International Conference on VLSI (VLSI 04), pp. 491-494, 2004.
- [12] Diestel, R., "Graph Theory", springer Verlag, Vol. 173, 2005.
- [13] Somenzi, F., "CUDD: CU Decision Diagram Package," <ftp://vlsi.colorado.edu/pub/>, 2003.
- [14] Hansen, M., et. al., "Unveiling the ISCAS-85 Benchmarks: A Case Study in Reverse Engineering," IEEE Trans. On Design and Test, Vol., 16, 1999, pp. 72-80.