Binary Decision Diagrams: An Improved Variable Ordering using Graph Representation of Boolean Functions

P.W. C. Prasad, A. Assi, A. Harb and V.C. Prasad

Abstract—This paper presents an improved variable ordering method to obtain the minimum number of nodes in Reduced Ordered Binary Decision Diagrams (ROBDD). The proposed method uses the graph topology to find the best variable ordering. Therefore, the input Boolean function is converted to a unidirectional graph. Three levels of graph parameters are used to increase the probability of having a good variable ordering. The initial level uses the total number of nodes ($NN$) in all the paths, the total number of paths ($NP$) and the maximum number of nodes among all paths ($MNNAP$). The second and third levels use two extra parameters: The shortest path among two variables ($SP$) and the sum of shortest path from one variable to all the other variables ($SSP$). A permutation of the graph parameters is performed at each level for each variable order and the number of nodes is recorded. Experimental results are promising; the proposed method is found to be more effective in finding the variable ordering for the majority of benchmark circuits.

Keywords—Binary Decision Diagrams, Graph Representation, Boolean Functions Representation, Variable Ordering

I. INTRODUCTION

A large variety of problems in digital system design, combinational optimization and verification can be formulated in terms of operations on Boolean functions [1]. Algorithms that efficiently manipulate Boolean functions have become increasingly popular and important in Very Large Scale Integration (VLSI) and Computer Aided Design (CAD) applications. Many methods have been developed, such as truth tables, disjunctive normal form and Boolean formulas. Due to the ever increasing demand for better performance, most of these methods have been unable to produce efficient solutions for combinational problems in digital systems. This demand has encouraged the researchers to look for a better method to manipulate Boolean functions.

During the last two decades, Binary Decision Diagrams (BDDs) have achieved great popularity as data structures for representing Boolean functions in solving most of the combinational problems which arise in synthesis and verification of digital systems [1]-[3]. Despite the fact that the BDD is a relatively old technique, its advantages for canonical representation were recognized and emphasized by Bryant [2]. Significant breakthroughs in the optimization of digital circuits have been achieved using the two remarkable and important properties of BDDs. First, BDDs are canonical; this property is useful when verifying the equivalence between two circuits [3]. In fact, two circuits are equivalent if and only if their BDDs are identical for a specific variable ordering. Second, BDDs are effective structures for the representation of large combinatorial sets.

BDDs can be represented by algorithms that travel through all the nodes and edges of the directed graph in some order and therefore take polynomial time in the current size of the graph. However, when new BDDs are created, it might significantly increase the number of nodes in the BDD, depending on the node placement in the graph, which can lead to exponential memory and run time requirements [4]. The choice of BDD variable order has a direct impact on the size of the BDD, and determining an optimal variable ordering is an NP-hard problem [5].

The ROBDD representation is defined by imposing restrictions on the BDD specification of Akers [6] such that the resulting form is canonical. ROBDD gained widespread use in logic design verification, test generation, fault simulation and logic synthesis. Since the size of an ROBDD heavily depends on the variable order used, the researchers in this area are actively involved in finding a variable order that minimizes the number of nodes in the ROBDD. While the ROBDD is canonical for a given variable ordering, the number of nodes that form the ROBDD may vary dramatically from one order to the other. Accordingly, much attention has been devoted to techniques used to find optimal variable orderings. All these variable ordering techniques fall into two categories: mainly Static [7], [8] or Dynamic [9]-[11] variable ordering techniques.
Some applications have ROBDDs with different variable ordering, whereas further manipulations of these ROBDDs require identical variable ordering. For truth tables, this is not the case, since truth tables are explicit representations of each and every point of the function [12]. In a BDD, each path (from the root node to the terminal node) corresponds to a cube of the Boolean function and the order of variables defines the min-terms which will be simplified as cubes. Reduction of the number of paths would imply reduction in the number of cubes. This also can be done by finding an optimal order of the BDD variables [13].

In general, it is hard to predict the effect of variable ordering on the BDD size, this requiring the trial of all possible ordering schemes. It is also hard to find the best order for a given Boolean function [12], [14], [15]. However, there are some observations that help in finding a good variable ordering.

There are two main concerns in building the BDDs of Boolean functions: time and space requirements. For a given Boolean function, the primary requirement to minimize time and space complexity is to represent its BDD with minimum a number of nodes [14]-[16]. Usually, the number of nodes in a BDD is directly related to the variable ordering of the BDD [17]-[23]. There are a variety of methods to find the optimal variable ordering for BDDs but none can fulfill both the time and the space requirements.

The rest of this paper is organized as follows: In the second section, background information pertaining to the construction and implementation of BDDs is discussed. In section three, the new variable ordering technique based on three level permutations of graph parameters (3-LPGP) is introduced and the experimental results are given in section four. Finally in section five we conclude our paper with an outline of our future work.

II. PRELIMINARIES

Basic definitions for binary decision diagrams are detailed in standard sources [2], [3], [6], [24]. However, we summarize the following definitions.

Definition 2.1: A BDD is a rooted directed acyclic graph \( G = (V, E) \) with vertex set \( V \) containing two types of vertices, non-terminal and terminal vertices. A non-terminal 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 and has no outgoing edge: \( value(v) \in \{0, 1\} \)

A BDD can be used to compute a Boolean function \( f(x_1, x_2, ..., x_n) \) in the following way. Each input \( a = (a_1, a_2, ..., a_n) \in \{0, 1\}^n \) defines a computation path through the BDD that starts at the root. If the path reaches a non-terminal node \( v \) that is labeled by \( x_i \), it follows the path low\((v) \) if \( a_i = 0 \), and it follows the path high\((v) \) if \( a_i = 1 \). The label of the terminal vertex determines the return value of the BDD on input \( a \). In a more formal way, we can define recursively a Boolean function that corresponds to a BDD.

Definition 2.2: A BDD having root vertex \( v \) denotes a Boolean function \( f_v \) defined as follows:

1) If \( v \) is a terminal vertex and \( value(v) = 1 \), then \( f_v = 1 \).

2) If \( v \) is a non-terminal vertex and index\((v) = x_i \), then \( f_v \) is given by

\[
\begin{align*}
&x_i \cdot f_{\text{low}(v)}(x_1, x_2, ..., x_n) + x_i \cdot f_{\text{high}(v)}(x_1, x_2, ..., x_n) \\
&= f_{\text{low}(v)}(x_1, x_2, ..., x_n) + f_{\text{high}(v)}(x_1, x_2, ..., x_n)
\end{align*}
\]

The variable \( x_i \) is called the decision variable for \( v \).

Definition 2.3: An ordered binary decision diagram (OBDD) is a BDD with the constraint that the input variables are ordered in every source to sink path in the OBDD visits the input variables in ascending order.

Definition 2.4: A reduced ordered binary decision diagram (ROBDD) is an OBDD where each node represents a distinct logic function. It has the following two properties:

(i) There are no redundant nodes in which both of the two edges leaving the node point to the same next node are present within the graph. If such a node exists, it is removed and the incoming edges redirected to the following node.

(ii) If two nodes point to two identical sub-graphs (i.e. isomorphic sub-graphs), then one sub-graph will be removed and the remaining one will be shared by the two nodes.

2.1 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 \( f = x_1 \cdot x_2 + ... + x_{2n-1} \cdot x_{2n} \). If the variable ordering is given by \((x_1, x_2, ..., x_n)\), i.e. \( \pi(i) = x_i \forall i \), the size of the resulting BDD is \( 2n \). On the other hand, if the variable ordering is chosen as \((x_1, x_3, ..., x_{2n-1}, x_2, x_4, ..., x_{2n})\), 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. Fig. 1 shows the effect of the variable ordering on the size of BDDs for the following function (1):

\[
f = x_1 \cdot x_2 + x_1 \cdot x_3 \cdot x_4 + x_1 \cdot x_3 \cdot x_4
\]
The proposed variable ordering algorithms use the graph topology to find a good variable order. In these algorithms the input Boolean function is converted into a unidirectional graph [25]. All Boolean operations in the Boolean function are converted into combinations of AND and NOT operations [26], which are represented as nodes in the graph. Each input and output of the Boolean function is also represented as a node in the graph.

3.1 Level-I Algorithm (Node-Path Level)

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:

**Step 1:** For every variable, we compute the NP from the corresponding input node to all output nodes.

**Step 2:** For every variable, we compute the NN in every path and we note the MNNAP.

**Step 3:** For every variable, we add the NN in all paths to get the total number of nodes (TNN).

**Step 4:** The variables are then sorted in descending order according to their TNN, then according to their NP, and MNNAP respectively. A heuristic selection is made if two or more variables have equal TNN, NP and MNNAP.

**Example:**

Consider the following Boolean function (2):

\[ f = \overline{x_1} \cdot \overline{x_2} \cdot \overline{x_4} + \overline{x_1} \cdot x_4 + x_2 \cdot \overline{x_4} + \overline{x_2} \cdot x_3 \]  

(2)

For every variable, we compute the NP from the corresponding input node to all output nodes. For every variable, we add the NN in all paths to get the total number of nodes (TNN). 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>
<thead>
<tr>
<th>Graphical Parameter</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(x_3)</th>
<th>(x_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of different paths (NP)</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Total number of nodes among all paths (TNN)</td>
<td>10</td>
<td>13</td>
<td>4</td>
<td>18</td>
</tr>
<tr>
<td>Max number of nodes among all paths (MNNAP)</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

3.2 Level II Algorithm

This level is an improved version of the Node-Path level algorithm explained in Section 3.1. It uses an additional parameter per variable that is the Sum of the Shortest Paths (SSP). The SSP of a variable is the sum of the shortest paths from this variable to all other variables. In this level, the variable order is found based on Primary (NP, TNN and SSP) and Secondary (MNNAP) parameters. In the following, we explain the proposed algorithm. Steps 1 to 3 are the same as algorithm-I and they are repeated here for convenience.

**Step 1:** For every variable, we compute the NP from the corresponding input node to all output nodes.

**Step 2:** For every variable, we compute the NN in every path and we note the MNNAP.
**Step 3:** For every variable, we add the $NN$ in all paths to get the $TNN$.

**Step 4:** For every variable, we compute the $SSP$.

**Step 5:** $NP$, $TNN$, and $SSP$ are considered as primary parameters 1, 2, and 3. $MNNAP$ is considered as secondary parameter 4.

**Step 6:** The variables are then sorted in descending order according to parameter 1, then according to parameters 2, 3 and 4. A heuristic selection is made if two or more variables have equal parameters 1, 2, 3, and 4.

**Step 7:** For the obtained variable ordering we note the number of nodes of the BDD.

**Step 8:** We then repeat step 6 for the remaining permutations of parameters 1, 2 and 3. The variable ordering and the number of nodes are recorded for every permutation.

**Step 9:** Among all permutations, the variable ordering with the minimum number of nodes is considered as the best variable ordering.

**Example:**
Consider the following Boolean function (4):

$$f = x_1 \cdot x_2 + x_1 \cdot x_2 \cdot x_3 + x_1 \cdot x_3$$

(4)

This Boolean function (4) is converted into an equivalent Boolean function, with only AND and NOT operations (5) as follows:

$$f = (x_1 \cdot x_2) \cdot (x_1 \cdot x_2 \cdot x_3) \cdot (x_1 \cdot x_3)$$

(5)

Fig. 3 shows the graph of the new function $f$. Nodes $x_1, x_2$ and $x_3$ are the input nodes and 10 is the output node.

**Table II**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP$</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$TNN$</td>
<td>14</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>$SSP$</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

**Table III**

<table>
<thead>
<tr>
<th>Parameter Sequence</th>
<th>Variables Ordering</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$NP, TNN, SSP, MNNAP$</td>
<td>$x_1, x_2, x_3$</td>
<td>3</td>
</tr>
<tr>
<td>$TNN, SSP, NP, MNNAP$</td>
<td>$x_1, x_2, x_3$</td>
<td>3</td>
</tr>
<tr>
<td>$SSP, NP, TNN, MNNAP$</td>
<td>$x_1, x_1, x_2$</td>
<td>4</td>
</tr>
<tr>
<td>$NP, SSP, TNN, MNNAP$</td>
<td>$x_1, x_1, x_2$</td>
<td>3</td>
</tr>
<tr>
<td>$SSP, NN, NP, MNNAP$</td>
<td>$x_1, x_1, x_2$</td>
<td>4</td>
</tr>
<tr>
<td>$NN, NP, SSP, MNNAP$</td>
<td>$x_1, x_2, x_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

The algorithm then selects the best variable ordering, i.e. $x_1, x_2, x_3$ or $x_1, x_3, x_2$ as the final variable order.

### 3.3 Level III Algorithm

This variable ordering algorithm is an improved version of the Parameter Permutation algorithm explained in Section 3.2. In addition to the four parameters $TNN$, $NP$, $SSP$ and $MNNAP$, the Matrix algorithm uses one more parameter, which is the shortest path (SP) between every pair of variables. In the following we explain the proposed algorithm:

**Step 1:** For every variable, we compute the $NP$ from the corresponding input node to all output nodes.

**Step 2:** For every variable, we compute the $NN$ in every path and we note the $MNNAP$.

**Step 3:** For every variable, we add the $NN$ in all paths to get the $TNN$.

**Step 4:** For every variable, we find the $SP$ to all other input variables and the $SSP$.

**Step 5:** $SP$ is considered as primary parameter 1, and $NP$, $TNN$, $SSP$ and $MNNAP$ are considered as secondary parameters 2, 3, 4 and 5 respectively.

**Step 6:** The variables are then sorted in descending order according to parameter 2, then according parameter 3, 4 and 5. A heuristic selection is made if two or more variables have equal parameters 2, 3, 4, and 5.

**Step 7:** The first variable in the sorted list is considered the first variable in the order.

**Step 8:** We repeat steps 9 to 13 for $i = 2$ to $n$ (where $i$ is the variable sequence in the variables ordering and $n$ is number of variables)

**Step 9:** The variable with which the $(i-1)^{\text{th}}$ variable has the minimum $SP$ value is ranked next in the variable ordering.
Step 10: If, in step 9, two or more variables have an equal minimum value of parameter 1 (SP), then preference is given to the variable with the higher value of parameter 2, 3, 4, and 5 in order.

Step 11: For the obtained variable ordering we note the number of nodes of the BDD.

Step 12: We then repeat steps 6 to 11 for the remaining permutations of parameters 2, 3, 4 and 5 respectively. The variables ordering and the number of nodes are recorded for all permutations.

Step 13: From all permutations, the variables ordering with the minimum number of nodes is considered as the best variable ordering.

Example:
Consider the following Boolean function (6):
\[ f = x_1 \cdot x_2 \cdot x_4 + x_1 \cdot x_3 \cdot x_4 + x_1 \cdot x_2 \cdot x_4 \]  
(6)

This Boolean function (6) is converted into an equivalent Boolean function (7) with only AND and NOT operations. Fig. 4 shows the graph of the converted function.

\[ f = (x_1 \cdot x_2 \cdot x_4). (x_1 \cdot x_3 \cdot x_4). (x_1 \cdot x_4). (x_2 \cdot x_4) \]  
(7)

Fig. 4 Graph representation for function (6)

Table 4 summarizes the SP of this graph. NP, NN, SSP, and MNNAP are summarized in Table 5.

The number of nodes for all possible parameter permutations is recorded and \( x_1, x_2, x_3, x_4 \) is considered as the final variable ordering for level III method.

IV. Experimental Results

In this section we present the experimental results obtained by applying the proposed three-level permutation method to selected ISCAS benchmark circuits using the Colorado University Decision Diagram (CUDD) Package [27]. A large collection of ISCAS benchmark circuits [28]-[30] has been selected to demonstrate the performance of the proposed method. Tables 6 and 7 summarize a comparison of our results to the best results obtained by three different CUDD variable reordering methods (Random Swapping, Symmetric Sifting and Window Permutation) for benchmark circuits with 20-139 inputs and 22-137 outputs.

In Table 6 column 1 shows the ISCAS benchmark name, and columns 2 and 3 show the size of the benchmark in term of inputs and outputs number. Columns 4 to 6 show the number of nodes required for the construction of the ROBDD using those three CUDD variable ordering methods. The results of the 3-LPGP method are shown in column 7, and the gain factors of the proposed method are given in columns 8, 9, and 10.

Table 7 shows a comparison with the result of the method that leads to the minimum number of nodes. Therefore it takes first seven columns of Table 6, along with four additional columns. Column 8 shows the minimum number of nodes required for the construction of the ROBDD using those three CUDD variable ordering methods. The results of the 3-LPGP method are shown in column 7, and the gain factors of the proposed method are given in columns 8, 9, and 10.

Table 7 shows a comparison with the result of the method that leads to the minimum number of nodes. Therefore it takes first seven columns of Table 6, along with four additional columns. Column 8 shows 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). The gain factors of each method against the method that gives the minimum numbers of nodes are given in columns 9, 10, 11 and 12.
TABLE VI
COMPARISON OF THE THREE-LEVEL PERMUTATION METHOD AGAINST THREE CUDD METHODS FOR ISCAS BENCHMARK CIRCUITS

<table>
<thead>
<tr>
<th>Benchmark Circuit</th>
<th>Number of Nodes</th>
<th>3-LPGP Method</th>
<th>Random Permutation</th>
<th>Symmetric Sift</th>
<th>Window Permutation</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>56</td>
<td>28</td>
<td>50</td>
<td>100</td>
<td>51</td>
</tr>
<tr>
<td>X2</td>
<td>89</td>
<td>44</td>
<td>77</td>
<td>100</td>
<td>89</td>
</tr>
<tr>
<td>X3</td>
<td>120</td>
<td>60</td>
<td>95</td>
<td>100</td>
<td>120</td>
</tr>
<tr>
<td>X4</td>
<td>177</td>
<td>89</td>
<td>120</td>
<td>100</td>
<td>177</td>
</tr>
<tr>
<td>X5</td>
<td>255</td>
<td>128</td>
<td>192</td>
<td>100</td>
<td>255</td>
</tr>
<tr>
<td>X6</td>
<td>361</td>
<td>181</td>
<td>244</td>
<td>100</td>
<td>361</td>
</tr>
<tr>
<td>X7</td>
<td>499</td>
<td>249</td>
<td>359</td>
<td>100</td>
<td>499</td>
</tr>
<tr>
<td>X8</td>
<td>663</td>
<td>331</td>
<td>431</td>
<td>100</td>
<td>663</td>
</tr>
<tr>
<td>X9</td>
<td>861</td>
<td>431</td>
<td>571</td>
<td>100</td>
<td>861</td>
</tr>
<tr>
<td>X10</td>
<td>1081</td>
<td>541</td>
<td>721</td>
<td>100</td>
<td>1081</td>
</tr>
<tr>
<td>X11</td>
<td>1369</td>
<td>689</td>
<td>929</td>
<td>100</td>
<td>1369</td>
</tr>
<tr>
<td>X12</td>
<td>1707</td>
<td>853</td>
<td>1157</td>
<td>100</td>
<td>1707</td>
</tr>
<tr>
<td>X13</td>
<td>2117</td>
<td>1058</td>
<td>1467</td>
<td>100</td>
<td>2117</td>
</tr>
<tr>
<td>X14</td>
<td>2601</td>
<td>1301</td>
<td>1801</td>
<td>100</td>
<td>2601</td>
</tr>
<tr>
<td>X15</td>
<td>3169</td>
<td>1583</td>
<td>2223</td>
<td>100</td>
<td>3169</td>
</tr>
<tr>
<td>X16</td>
<td>3817</td>
<td>1909</td>
<td>2519</td>
<td>100</td>
<td>3817</td>
</tr>
<tr>
<td>X17</td>
<td>4565</td>
<td>2283</td>
<td>3103</td>
<td>100</td>
<td>4565</td>
</tr>
<tr>
<td>X18</td>
<td>5413</td>
<td>2707</td>
<td>3507</td>
<td>100</td>
<td>5413</td>
</tr>
<tr>
<td>X19</td>
<td>6469</td>
<td>3234</td>
<td>4269</td>
<td>100</td>
<td>6469</td>
</tr>
<tr>
<td>X20</td>
<td>7633</td>
<td>3817</td>
<td>4857</td>
<td>100</td>
<td>7633</td>
</tr>
<tr>
<td>X21</td>
<td>9001</td>
<td>4501</td>
<td>5701</td>
<td>100</td>
<td>9001</td>
</tr>
<tr>
<td>X22</td>
<td>10569</td>
<td>5284</td>
<td>6789</td>
<td>100</td>
<td>10569</td>
</tr>
<tr>
<td>X23</td>
<td>12401</td>
<td>6201</td>
<td>8001</td>
<td>100</td>
<td>12401</td>
</tr>
<tr>
<td>X24</td>
<td>14565</td>
<td>7283</td>
<td>9565</td>
<td>100</td>
<td>14565</td>
</tr>
<tr>
<td>X25</td>
<td>17069</td>
<td>8534</td>
<td>11534</td>
<td>100</td>
<td>17069</td>
</tr>
<tr>
<td>X26</td>
<td>20001</td>
<td>10001</td>
<td>13991</td>
<td>100</td>
<td>20001</td>
</tr>
</tbody>
</table>

The results shown in Table 6 indicate that the proposed 3-LPGP algorithm decreases the number of nodes in 94% of the circuits compared to Window Permutation, 93% compared to Random Swapping and almost 69% compared to Symmetric Sift.

V. CONCLUSION
A three-level permutation of a graph parameter algorithm for minimizing the number of nodes in BDDs has been presented. The proposed 3-LPGP algorithm is capable of handling multiple output benchmark circuits. This algorithm has been implemented and tested using ISCAS benchmark circuits and the results have been compared with three selected CUDD reordering methods. The algorithm is deterministic in the sense that there is no heuristic involved in any of the primary parameters of the algorithm. Experimental results indicate that this algorithm is a promising alternative to existing reordering methods to reduce the number of nodes in BDD. Our on-going work will address. We will also analyze the cases when this method fails to produce near-best variable ordering. Our next step in this research work will be the use of SYNOPSYS tool to validate our results; especially the silicon area requirements of the benchmarks used to support and justify the proposed 3-LPGP algorithm.

REFERENCES

World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:2, 2008


