Comparison of Decision Diagrams for Multiple-Output Logic Functions

Tsutomu SASAO[1,2], Yukihiro IGUCHI[3], and Munehiro MATSUURA[1]
[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Center for Microelectronic Systems, Kyushu Institute of Technology
[3]Department of Computer Science, Meiji University

Abstract

This paper shows four different methods to evaluate multiple-output logic functions using decision diagrams: SBDD, MTBDD, BDD for characteristic functions (CF), and BDDs for ECFNs (Encoded Characteristic Function for Non-zero outputs). Methods to compute average evaluation time for each type of decision diagrams are presented. By experimental analysis using benchmark functions, the number of nodes and average evaluation time are compared. Evaluation using BDDs for ECFN outperforms those using MTBDDs, BDDs for CF, and SBDDs with respect to the size of BDDs and computation time. The sizes of BDDs for ECFNs are smaller than ones of corresponding MTBDDs (Multi-Terminal BDDs), BDDs for CFs (Characteristic Functions), and SBDDs (Shared BDDs).

1 Introduction

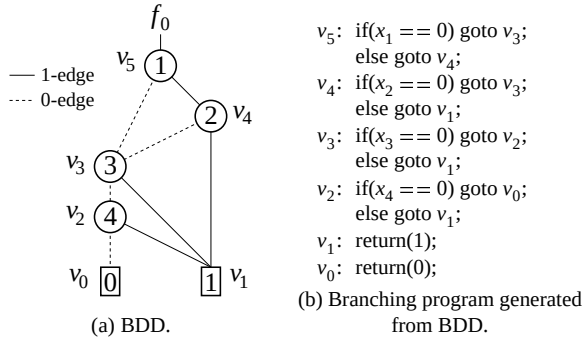
Various kinds of BDDs exist to represent multiple-output logic functions. Among them, an MTBDD (multi-terminal BDD), an SBDD (shared BDD), and a BDD representing the characteristic function (BDD for CF) are popular. For a BDD for CF or an MTBDD, the evaluation time is $O(n+m)$, where n denotes the number of input variables, and m denotes the number of output variables. For an SBDD, the evaluation time is $O(n \cdot m)$. BDDs for CFs and MTBDDs are suitable for high speed evaluation. Unfortunately, the sizes of these BDDs tend to be too large. Thus, we have to resort to SBDDs, which require longer evaluation time.

In [6], a new data structure, a BDD for ECFN (encoded characteristic function for non-zero outputs) is introduced. In this paper, we show that by using BDDs for ECFNs, logic evaluation can be more than two times faster than by using SBDDs. Also, the size of the memory is smaller than by using SBDDs. This method can be useful for logic simulation [1, 3] and embedded system [2].

2 Function Evaluation using BDDs

Another method to represent a logic function is the branching program. Fig. 2.1 shows a method to convert a BDD into a branching program. For a given logic function, construct a binary decision diagram (BDD), as shown in Fig. 2.1(a). Then, replace each non-terminal node by an *if then else* statement. The result is a branching program, as shown in Fig. 2.1(b). Then, by implementing this program by a computer, we can evaluate the logic function.

The time to evaluate a logic function for a given input is proportional to the number of non-terminal nodes that appear in the path from the root node to the terminal nodes.



(a) BDD. (b) Branching program generated from BDD.

Figure 2.1: BDD and branching program.

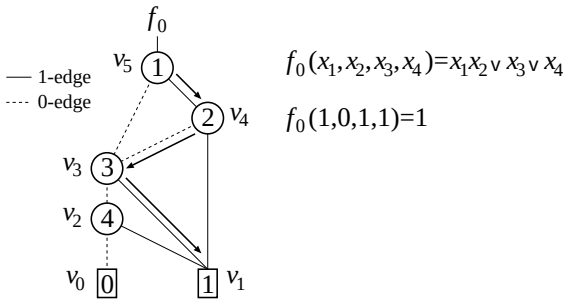


Figure 2.2: Function evaluation using BDD.

Example 2.1 *Consider the BDD in Fig. 2.1. When* $(x_1, x_2, x_3, x_4) = (1, 0, 1, 1)$ *is applied,* $f_0(1, 0, 1, 1) = 1$ *is evaluated, as shown in Fig. 2.2. Note that this involves a traverse across three edges. This figure also shows that the minimum path length is two, while the maximum path length is four.* *(End of Example)*

As shown in the above example, the evaluation time depends on the input values. To estimate the evaluation time of different DDs, we introduce a metric **average path length**, which measures the average evaluation time over all possible combinations. We measure the average evaluation time by the average path length in the BDD. We assume that each variable occurs as a 0 with the same probability as a 1. That is, at any node, a 0-edge is as likely to be traversed as a 1-edge.

Definition 2.1 The node traversing probability, *denoted by* $P(v_i)$*, is the probability of traversing the node* v_i *when a BDD is traversed from the root node to a terminal node.* **The edge traversing probability**, *denoted by* $P(e_{i_0})$

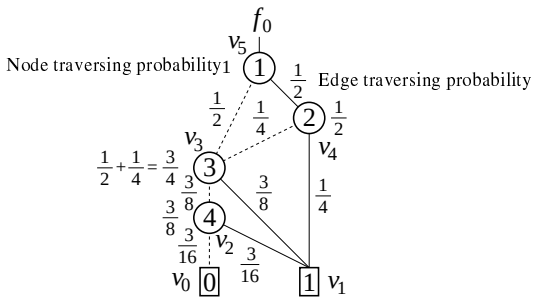


Figure 2.3: Average path length of BDD.

($P(e_{i_1})$) is the probability of traversing the 0 edge (the 1 edge) from the node v_i.

Since, the probabilities that 0 and 1 occur are assumed to be equal and 1/2, $P(e_{i_0}) = P(e_{i_1}) = P(v_i)/2$.

Lemma 2.1 *The node traversing probability is equal to the sum of the edge traversing probabilities of the incoming edges.*

Theorem 2.1 *The average path length is equal to the sum of the node traversing probabilities of the non-terminal nodes.*

Example 2.2 *Let us calculate the average path length of the BDD in Fig. 2.3.* $P(v_5) = 1$, *and we have* $P(e_{5_0}) = P(e_{5_1}) = 1/2$. $P(v_4) = 1/2$. $P(e_{4_0}) = P(e_{4_1}) = 1/4$. $P(v_3) = P(e_{5_0}) + P(e_{4_0}) = 1/2 + 1/4 = 3/4$. $P(e_{3_0}) = P(e_{3_1}) = 3/8$. $P(v_2) = 3/8$. $P(e_{2_0}) = P(e_{2_1}) = 3/16$. *Thus, the average path length of the BDD is given by* $P(v_5) + P(v_4) + P(v_3) + P(v_2) = 1 + 1/2 + 3/4 + 3/8 = 42/16 = 2.625$. *(End of Example)*

3 BDDs for Multiple-Output Functions

In this part, we introduce MTBDDs, SBDDs, and BDDs for CFs to represent multiple-output logic functions.

3.1 Evaluation using MTBDDs

Fig. 3.1 shows an example of an MTBDD representing a 4-variable 4-output logic function. Since each terminal node stores all output values, the output values are obtained by just traversing the MTBDD from the root node to a terminal node.

When the outputs are stored in 32-bit words, up to 32 output values of a terminal node can be evaluated by a single memory access, and up to 64 output values can be evaluated within two memory accesses. In general, the evaluation time is $O(n + m/32) = O(n + m)$.

Similar to the case of BDDs, the average evaluation time is equal to the average path length. Since many outputs are evaluated by one traversal of the MTBDD, the evaluation is fast. Unfortunately, the number of terminal nodes can be up to 2^m, and the number of nodes will be too large to be stored in a memory for many practical applications.

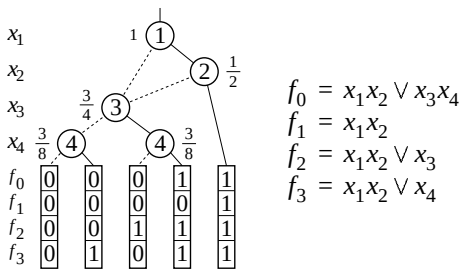


Figure 3.1: Average path length of the MTBDD.

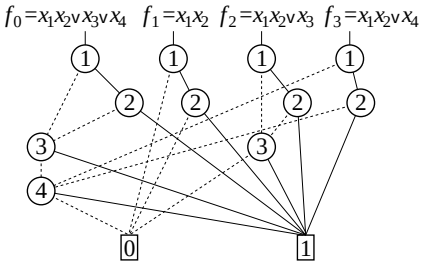


Figure 3.2: Example of SBDD.

3.2 Evaluation using SBDDs

An SBDD is obtained from the set of BDDs representing multiple-output functions, by sharing nodes among BDDs to reduce the number of nodes. For an *n*-variable *m*-output function, *m* BDDs are shared. To evaluate a multiple-output function for each output, traverse the edges from the root node to a constant node according to the values of the input variables. Thus, the evaluation time is $O(n \cdot m)$.

Fig. 3.2 is the SBDD corresponding to the MTBDD in Fig. 3.1. To evaluate all the output values, we have to traverse all the BDDs for f_0, f_1, f_2, and f_3 from the root node to the constant nodes, sequentially. The average path length of an SBDD is the sum of the average path lengths of the individual BDDs in Fig. 3.3, instead of the SBDD in Fig. 3.2.

Example 3.1 *The average path length of the SBDD shown in Fig. 3.2 is* $\frac{21}{8} + \frac{3}{2} + \frac{9}{4} + \frac{9}{4} = \frac{69}{8} = 8.625$. *The BDD in Fig. 3.4 represents the multiple-output function by using auxiliary variables* z_1 *and* z_0. *In this case, the average eval-*

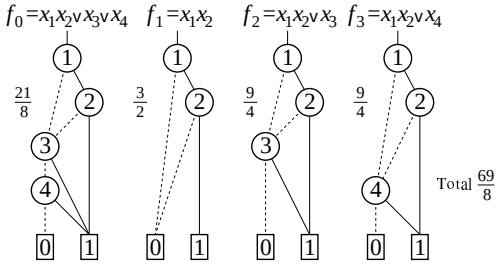


Figure 3.3: Average evaluation time of SBDD.

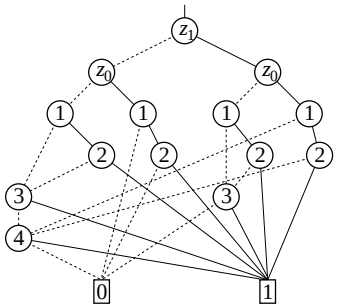


Figure 3.4: BDD for multiple-output function using auxiliary variables.

Table 3.1: Derivation of CF from the truth table of multiple-output function.

(a)

x_1	x_2	f_0	f_1
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

$f_0 = x_1 x_2$
$f_1 = x_1 \vee x_2$

(b)

x_1	x_2	f_0	f_1	CF	x_1	x_2	f_0	f_1	CF
0	0	0	0	1	1	0	0	0	0
0	0	0	1	0	1	0	0	1	1
0	0	1	0	0	1	0	1	0	0
0	0	1	1	0	1	0	1	1	0
0	1	0	0	0	1	1	0	0	0
0	1	0	1	1	1	1	0	1	0
0	1	1	0	0	1	1	1	0	0
0	1	1	1	0	1	1	1	1	1

uation time is 8.625+6=14.625. *(End of Example)*

3.3 Evaluation using BDDs for CFs

For fast evaluation of multiple-output functions, characteristic functions (CFs) are useful [1, 3, 7]. The CF for an n-variable m-output logic function is an $(n+m)$-variable 1-output logic function, where CF =1 if and only if the combinations of the inputs and the outputs are valid.

Example 3.2 *Table 3.1(a) shows the truth table of the two-output function $f_0 = x_1 x_2$, $f_1 = x_1 \vee x_2$. Table 3.1(b) is the truth table for the corresponding CF. For example, since the combination $(x_1, x_2, f_0, f_1) = (0,0,0,0)$ exists in Table 3.1(a), the value of CF in (b) the corresponding row is 1. Since the combination $(x_1, x_2, f_0, f_1) = (0,0,0,1)$ does not exist in Table 3.1(a), the value of CF in (b) the corresponding row is 0.* *(End of Example)*

For an n-variable m-output function, the number of rows in the truth table for CF is 2^{n+m}. Among them, 2^n rows have 1's and the remaining rows have 0's.

By using the BDD for CF, we can evaluate a multiple-output logic function quickly. Consider the multiple-output function from Figs. 3.1 and 3.2. The BDD for CF for the multiple-output function is shown in Fig. 3.5. Note that the non-terminal nodes having indices f_0, f_1, f_2, and f_3 correspond to auxiliary variables. In a BDD for CF, the auxiliary variable f_i can appear only after all the input variables that depend on f_i appear. Also, either the 0-edge or the 1-edge

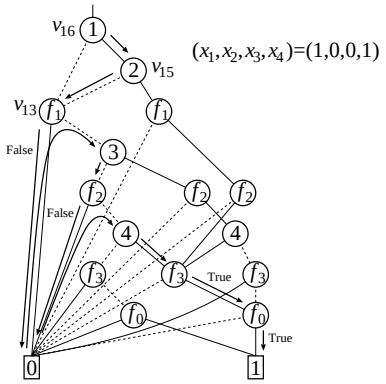


Figure 3.5: Evaluation of function using BDD for CF.

of each auxiliary variables is connected to the constant 0 node. The next example shows a method to evaluate a logic function by using a BDD for CF.

Example 3.3 *(Function evaluation using BDD for CF) Using the BDD for CF in Fig. 3.5, obtain the output values for the input $(x_1, x_2, x_3, x_4) = (1,0,0,1)$.*

The index of the root node v_{16} is x_1. Since the value of x_1 is 1, proceed to the 1-edge to node v_{15}, whose index is x_2. Since the value of x_2 is 0, proceed to the 0-edge to reach the node v_{13}, which corresponds to auxiliary variable f_1. When we arrive at an auxiliary variable, we have to proceed along either the 0-edge or the 1-edge. If we proceed along the 0-edge and arrive at the terminal node 0, then the output value is non-zero, that is, the output is 1. In this case, backtrack and proceed along the 1-edge. On the other hand, if we proceed along the 1-edge and arrive at the terminal node 0, then the output value is non-1, that is, the output is 0. In this case, we have to backtrack and proceed along the 0-edge.

In this example, we proceed along the 1-edge first and arrived at the terminal node 0. So we have to backtrack to the 0-edge, because we obtained the value $f_1 = 0$. If we proceed along the 0-edge first, we will not arrive at the terminal node 0, and we obtain the value $f_1 = 0$. In this case, we can reduce the search time for the backtrack. In this example, when we arrive at an auxiliary variable, we first proceed along the 1-edge. Similarly, we search the remaining parts, and by traversing 10 edges, we can determine that the output values are $(f_0, f_1, f_2, f_3) = (1,0,0,1)$. *(End of Example)*

If the probabilities of taking values 0 and 1 are the same for all f_i, then the evaluation time does not depend on the order of traversal. As shown in Example 3.3, the average evaluation time using a BDD for CF is obtained from the average path length of the BDD. However, when we arrive at a node v_i that corresponds to an auxiliary variable, we have to modify the algorithm as follows: At the node v_i that corresponds to an auxiliary variable, either the 0-edge or the 1-edge is connected to the terminal node 0. Let the

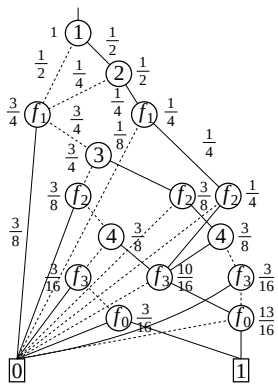


Figure 3.6: Average path length in BDD for CF.

probability taking values 0 and 1 be the same for f_i. Then, the probability of traversing the edge that is connected to the constant 0 is one half of the probability $P(v_i)$. Also, the other edge is always traversed. Thus, the sum of the passing probability for both edges is $1.5 \cdot P(v_i)$.

Example 3.4 *Fig. 3.6 shows the method to obtain the average evaluation time of the BDD for CF shown in Fig. 3.5. It is calculated as* $1+\frac{1}{2}+(\frac{3}{4}+\frac{3}{8})+(\frac{1}{4}+\frac{1}{8})+\frac{3}{4}+(\frac{3}{8}+\frac{3}{16})+(\frac{3}{8}+\frac{3}{16})+(\frac{1}{4}+\frac{1}{8})+\frac{3}{8}+\frac{3}{8}+(\frac{3}{16}+\frac{3}{32})+(\frac{10}{16}+\frac{10}{32})+(\frac{3}{16}+\frac{3}{32})+(\frac{3}{16}+\frac{3}{32})+(\frac{13}{16}+\frac{13}{32})=\frac{288}{32}=9.0$. *(End of Example)*

In this way, BDDs for CFs evaluate a logic function in $O(n+m)$ time, which is faster than the corresponding SBDDs. However, when the value of $n+m$ is large, the number of nodes is excessive, and we cannot construct the BDD for CF.

4 Function Evaluation using BDDs for ECFNs

A new method to represent multiple-output functions, using a BDD for ECFN (encoded characteristic function for non-zero outputs) [6], is faster and requires smaller amount of memory than SBDDs. This section shows the properties of the BDD for ECFN.

4.1 ECFN and Output Encoding Problem

An ECFN represents the mapping: $F : B^n \times B^u \to B$, where $u = \lceil \log_2 m \rceil$. $F(\boldsymbol{a}, \boldsymbol{b}) = 1$ iff $f_{v(\boldsymbol{b})}(\boldsymbol{a}) = 1$, where $v(\boldsymbol{b})$ is an integer represented by the binary vector $\boldsymbol{b}$.

Example 4.1 *The ECFN for the four-output function shown in Fig. 3.1 is* $F = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$. *(End of Example)*

ECFNs can be used in FPGA design, logic emulation, embedded system, etc [5]. A BDD for ECFN is considered as a generalization of an SBDD.

Definition 4.1 $z^0 = \bar{z}$ *and* $z^1 = z$.

Definition 4.2 *For an m-output function* $f_i (i = 0, 1, \ldots, m-1)$, *the ECFN is* $F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i$, *where* $\boldsymbol{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ *is a binary representation of the integer i, and* $u = \lceil \log_2 m \rceil$.

Table 4.1: Encoding Methods for Four-output Function.

z_1	z_0	*Encoding 1*	*Encoding 2*	*Encoding 3*
0	0	f_0	f_0	f_0
0	1	f_1	f_1	f_3
1	0	f_2	f_3	f_2
1	1	f_3	f_2	f_1

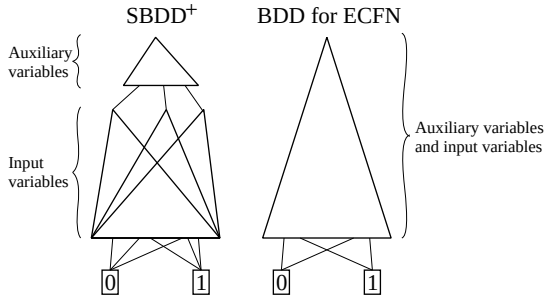


Figure 4.1: SBDD$^+$ and BDD for ECFN.

$z_0, z_1, \ldots, z_{u-1}$ are auxiliary variables. In the above definition, the integer i is represented by a binary vector $\boldsymbol{b}$ using the natural encoding. However, different encodings can simplify the representation.

Example 4.2 *Consider the four-output function* $F = (f_0, f_1, f_2, f_3)$, *where* $f_0 = 0$, $f_1 = x_1$, $f_2 = x_2$, *and* $f_3 = x_1 \vee x_2$. *The ECFN* F_1 *generated by Encoding 1 in Table 4.1 is* $F_1 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$. *In this case, we have* $F_1 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_2 \vee z_1 z_0 (x_1 \vee x_2) = z_0 x_1 \vee z_1 x_2$.

The ECFN F_2 *generated by Encoding2 in Table 4.1 is* $F_2 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_3 \vee z_1 z_0 f_2$. *In this case, we have* $F_2 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 (x_1 \vee x_2) \vee z_1 z_0 x_2 = \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_1 \vee z_1 x_2$.

This example shows that encodings influence the size of the representation. *(End of Example)*

4.2 Optimization of BDDs for ECFNs

The BDD shown in Fig. 3.4 can be considered as a special case of a BDD for ECFN. From now on, such a BDD will be simply denoted by an SBDD$^+$. As shown in Fig. 4.1, in an SBDD$^+$, the auxiliary variables appear above the input variables. However, in a general BDD for ECFN, the auxiliary variables and the input variables can be mixed together. By optimizing the ordering of the input and the auxiliary variables, the BDD can be minimized.

Definition 4.3 *The number of nodes in the BDD (including both non-terminal and terminal nodes) is denoted by nodes(BDD).*

Theorem 4.1 $nodes(\text{BDD for ECFN}) \leq nodes(\text{SBDD}^+)$

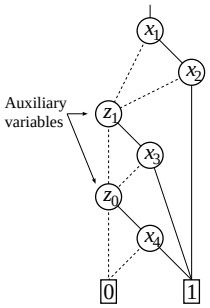


Figure 4.2: BDD for ECFN considering variable ordering.

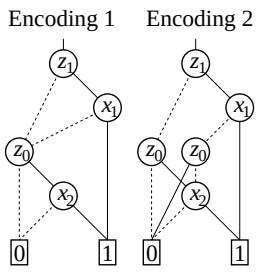


Figure 4.3: Optimization of the output encoding.

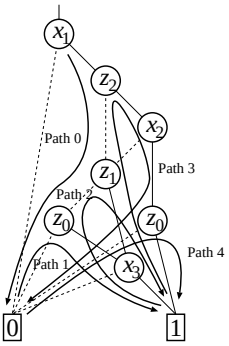


Figure 4.4: Method to traverse BDD.

Example 4.3 *Let the BDD in Fig. 3.4 be a BDD for ECFN. When we optimize the ordering of the variables, the BDD for ECFN shown in Fig. 4.2 is obtained. (End of Example)*

In this way, BDDs for ECFNs can be made smaller than corresponding SBDD$^+$s. By optimizing both the output encoding and the variable ordering, we can obtain the BDD for ECFN having fewer nodes than the corresponding SBDD$^+$.

Example 4.4 *Fig. 4.3 shows the BDDs for ECFNs that represent F_1 and F_2 in Example 4.2. Note that the numbers of nodes are 6 and 7, respectively. They use Encodings 1 and 2 in Table 4.1. This example shows that output encodings influence the size of the BDDs. (End of Example)*

4.3 Fast Evaluation using BDDs for ECFNs

By using the BDD for ECFN in Fig. 4.2, we can save memory and evaluate functions faster than the corresponding SBDD$^+$. For example, when we apply the input $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ to the BDD in Fig. 4.2, we can see that all the outputs f_0, f_1, f_2, and f_3 are 1. On the other hand, if we use the SBDD$^+$ in Fig. 3.4, we need to traverse the BDD four times. This shows that a BDD for ECFN often evaluates several outputs at one traversal.

Example 4.5 *Consider the BDD for ECFN shown in Fig. 4.4. Note that not all the auxiliary variables appear in a path from the root to a terminal node.*

Consider applying the input $(x_1, x_2, x_3) = (1, 1, 1)$ to the BDD for ECFN shown in Fig. 4.4. By traversing five paths $0 \sim 4$, we can evaluate eight outputs $(0, 1, 1, 1, 0, 1, 0, 1)$. In fact, in the path 0, f_0 is evaluated; in the path 1, f_1 is evaluated; in the path 2, f_2 and f_3 are evaluated; in the path 3, f_4 and f_6 are evaluated; in the path 4, f_5 and f_7 are evaluated. (End of Example)

From here, we will show a fast method to evaluate a BDD for ECFN. During BDD traversal, when we encounter an auxiliary variable, by searching both subtrees for 0-edge and 1-edge, we can evaluate all the outputs efficiently.

In order to evaluate all the outputs for an input, we need an efficient algorithm to traverse the BDD. The following algorithm produces the values of multiple-output logic functions in a compact form. Note that the algorithm is more complicated than that of BDD for CF.

Algorithm 4.1 *(Evaluation of Multiple-Output Function using a BDD for ECFN)*

```
eval(){
    u = ⌈log2 m⌉;
    eval(root, u);
    expand the parenthesis of the output list;
}
evalp(p, lastIndex){
    if (p == non-terminal node){
        if (p == node for an input variable){
            if (x[p → index] == 0)
                evalp(p → 0edge, lastIndex);
            else
                evalp(p → 1edge, lastIndex);
        } else if (p == node for an auxiliary variable) {
            currentIndex ← index of the current node;
            printf (2^(lastIndex−currentIndex−1));
            printf ("(");
            evalp(p → 0edge, currentIndex);
            evalp(p → 1edge, currentIndex);
            printf (")");
        } else {/* terminal node */
            printf (2^currentIndex);
            printf ("(");
            printf (the value of current terminal node);
            printf (")");
            return ():
        }
    }
}
```

The average evaluation time is equal to the average path length, which can be obtained similarly to the case of BDDs. Change the values of (z_2, z_1, z_0) to $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$,. . ., and $(1, 1, 1)$, and obtain the sum of the lengths

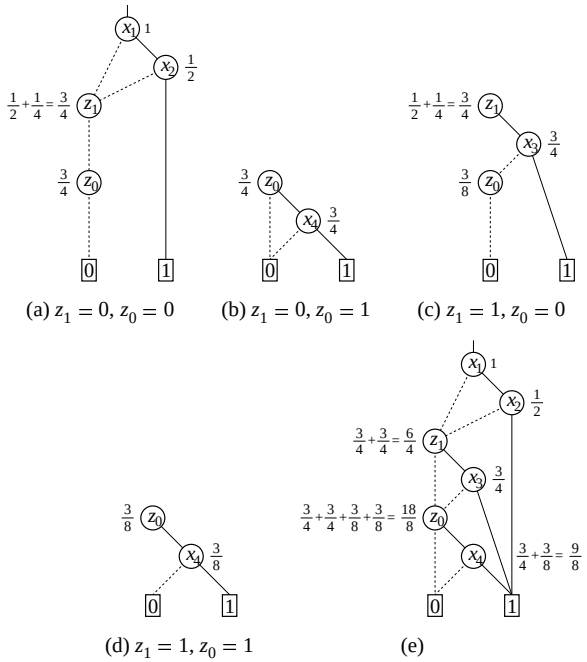


Figure 4.5: Average path length of BDD for ECFN.

of paths from the auxiliary variables to the terminal nodes.

Example 4.6 *Figs. 4.5(a)–(e) illustrate a method to obtain the average evaluation time of the BDD for ECFN in Fig. 4.2. The average evaluation time is calculated as* $1 + \frac{1}{2} + \frac{6}{4} + \frac{3}{4} + \frac{18}{8} + \frac{9}{8} = \frac{57}{8} = 7.12$. *(End of Example)*

5 Experimental Results

For selected MCNC benchmark functions, we constructed MTBDDs, BDDs for CFs, and SBDD$^+$s. Table 5.1 compares the average number of nodes and evaluation time of decision diagrams. To construct BDDs for ECFNs, we used the encoding method in [5]. In the table, *Name* denotes the function name, *In* denotes the number of inputs, *Out* denotes the number of outputs, *Node* denotes the number of nodes, and *Eva* denotes the average evaluation time. In the columns *BDD for ECFN*, *Min* denotes the case where the auxiliary variables and the input variable are mixed to reduce the BDDs.

6 Summary

In this paper, we presented four different methods to represent multiple-output functions by using BDDs. We compared the sizes and average evaluation time of the BDDs. We experimentally showed that BDDs for ECFNs require fewer nodes and require relatively short time to evaluate logic functions. A future research area includes efficient method to find good encodings for ECFNs.

Acknowledgments

This research is partly supported by the Grant in Aid for Scientific Research of The Japan Society for the Promotion of Science (JSPS), and the Takeda Foundation. Prof. Jon T. Butler improved English presentation.

Table 5.1: Average number of nodes and evaluation time of decision diagrams.

Name	In	Out	MTBDD		BDD for CF		SBDD$^+$		BDD for ECFN	
			Nod	Eva	Nod	Eva	Nod	Eva	Nod	Eva
apex1	45	45	942	8.8	3594	76.3	1324	269.8	1087	31.8
apex3	54	50	537	6.6	2449	81.6	986	286.6	708	28.2
duke2	22	29	662	6.4	997	49.9	366	150.3	346	22.5
e64	65	65	131	2.0	260	99.5	194	256.0	194	256.0
exep	30	63	1170	7.8	3030	102.3	675	255.7	660	38.0
k2	45	45	929	8.8	3594	76.3	1321	269.8	1167	29.1
mainpla	27	54	632	4.2	3114	85.2	1857	277.5	1018	49.2
mark1	20	31	4138	6.4	745	52.9	119	115.7	117	109.2
misex2	25	18	118	4.9	184	31.9	100	75.6	98	18.9
opa	17	69	241	4.4	1778	107.9	428	322.2	364	37.8
pdc	16	40	19178	10.2	5852	70.2	596	215.4	590	181.1
rckl	32	7	65	2.0	135	12.5	198	100.6	67	4.8
seq	41	35	378	3.3	1197	55.8	1284	168.0	493	28.2
shift	19	16	196095	15.5	3746	39.5	78	86.0	62	55.0
spla	16	46	11100	9.7	7522	78.1	628	226.6	604	99.9
t2	17	16	304	7.5	484	31.7	145	72.9	140	44.9
table5	17	15	436	8.0	677	30.5	685	114.1	476	10.6
ts10	22	16	589837	5.5	589826	31.5	163	88.0	83	14.5
vg2	25	8	420	11.8	471	23.8	90	48.9	82	45.7
x1dn	27	6	214	9.8	245	21.2	139	41.0	139	41.0
x6dn	39	5	195	4.1	225	11.6	235	41.2	193	13.4
x9dn	27	7	292	9.8	237	20.3	139	50.4	139	50.4
xparc	41	73	3844	3.6	4773	113.1	1947	304.8	1232	21.8
Ratio			271.3	0.072	162.5	0.36	1.0	1.0	0.81	0.41

Nod: Number of terminal and non-terminal nodes
Min: When the auxiliary variables can be any place
Eva: Average evaluation time
Ratio: Average of ratios when the value for SBDD is 1.0

References

[1] P. Ashar and S. Malik, "Fast functional simulation using branching programs," *ICCAD*, pp. 408-412, Oct. 1995.

[2] F. Balarin, M. Chiodo, P. Giusto, H. Hsieh, A. Jurecska, L. Lavagno, A. Sangiovanni-Vincentelli, E. M. Sentovich, and K. Suzuki, "Synthesis of software programs for embed-ded control applications," *IEEE Trans. CAD*, Vol. 18, No. 6, pp. 834-849, June 1999.

[3] P. C. McGeer, K. L. McMillan, A. Saldanha, A. L. Sangiovanni-Vincentelli, and P. Scaglia, "Fast discrete function evaluation using decision diagrams," *ICCAD'95*, pp. 402-407, Nov. 1995.

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

[5] T. Sasao, "Compact SOP representations for multiple-output functions: An encoding method using multiple-valued logic," *International Symposium on Multiple-Valued Logic*, Warsaw, Poland, 2001, pp. 207-212.

[6] T. Sasao, M. Matsuura, Y. Iguchi, and S. Nagayama, "Compact BDD Representations for Multiple-Output Functions," *VLSI-SOC 2001*, Montpellier, France, Dec. 3-5, 2001.

[7] C. Scholl, R. Drechsler, and B. Becker, "Functional simulation using binary decision diagrams," *ICCAD'97*, pp. 8-12, Nov. 1997.

[8] S. Yang, *Logic Synthesis and Optimization Benchmark User Guide Version 3.0*, MCNC, Jan. 1991.