On the Minimization of Average Path Lengths for Heterogeneous MDDs

Shinobu NAGAYAMA†
†Department of Computer Science and Electronics,
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Tsutomu SASAO†, ‡
‡Center for Microelectronics Systems,
Kyushu Institute of Technology
Iizuka 820-8502, Japan

Abstract

In this paper, we propose an exact and a heuristic minimization algorithms for the average path length (APL) of heterogeneous multi-valued decision diagrams (MDDs). In a heterogeneous MDD, each variable can take on the different number of values. To represent a binary logic function using a heterogeneous MDD, we partition the binary variables into groups, and treat them as multi-valued variables. By considering partitions of binary variables, we can obtain heterogeneous MDDs that represent logic functions more compactly and have smaller APLs than reduced ordered binary decision diagrams (ROBDDs). Experimental results using 21 *benchmark functions show that the APLs of the heterogeneous MDDs can be reduced by a half that of corresponding ROBDDs, on average, without increasing memory size.*

1. Introduction

Binary decision diagrams (BDDs) [5] and multi-valued decision diagrams (MDDs) [12] are extensively used in logic synthesis [9], logic simulation [1, 14], software synthesis [2, 11, 19], etc.. In the evaluation of logic function using BDDs and MDDs, the expected evaluation time is proportional to the average path length (APL) of BDDs and MDDs. Therefore, minimization of the APL reduces evaluation time of the logic function. Particularly, in logic simulation using decision diagrams [1, 14] and software synthesis [2, 11, 19], intensive minimization of APLs without increasing the memory size is very important to reduce the simulation time and to generate a fast and compact program code.

Minimization of APLs in BDDs has been considered in [8, 13, 21]. However, in a BDD, minimization of an APL often increases the memory size significantly, since a variable order that minimizes the APL is often different from the variable order that minimizes the number of nodes. In fact, the minimization of the APL for the BDD of the benchmark function $i10$ increases the memory size by 11 times of the original BDD size, as shown in Section 4.2. For a homogeneous MDD(k) (or MDD(k)), representing a logic function, the APL can be reduced by increasing the value of k (i.e., the size of group). However, the memory sizes of MDD(k)s increase exponentially with the value of k [18, 22]. On the other hand, for heterogeneous MDDs [19], APLs can be minimized without increasing the memory size. In [19], we proposed an APL minimization algorithm that considers only partitions of the binary variables, where the orderings of binary variables are fixed. In this paper, we propose an APL minimization algorithm that considers both partitions and orderings of binary variables. By experiments, we show that in heterogeneous MDDs, APLs can be reduced by a half that of corresponding BDDs, on the average, without increasing the memory size.

This paper is organized as follows. Section 2 shows the necessary terminology, definitions and theorems. Section 3 proposes an exact and a heuristic minimization algorithms for APLs. And, in Section 4, the experimental results show the efficiency of the algorithms.

2. Preliminaries

2.1. Partitions of Binary Variables

Definition 2.1 *Let $f(X)$ be a two-valued logic function, where $X = (x_1, x_2, \ldots, x_n)$ is an ordered set of binary variables. Let $\{X\}$ denote the unordered set of variables in X. If $\{X\} = \{X_1\} \cup \{X_2\} \cup \ldots \cup \{X_u\}$ and $\{X_i\} \cap \{X_j\} = \phi (i \neq j)$, then $(X_1, X_2, \ldots, X_u)$ is a* **partition** *of X. X_i is called a* **super variable**. *If $|X_i| = k_i$ $(i = 1, 2, \ldots, u)$ and $k_1 + k_2 + \ldots + k_u = n$, then a two-valued logic function $f(X)$ can be represented by the mapping $f(X_1, X_2, \ldots, X_u)$: $P_1 \times P_2 \times P_3 \times \ldots \times P_u \rightarrow B$, where $P_i = \{0, 1, 2, \ldots, 2^{k_i} - 1\}$ and $B = \{0, 1\}$.*

Definition 2.2 *A* **fixed-order partition** *of $X = (x_1, x_2, \ldots, x_n)$ is a partition $(X_1, X_2, \ldots, X_u)$, where*

$$X_1 = (x_1, x_2, \ldots, x_{k_1}),$$

$$
\begin{aligned}
X_2 &= (x_{k_1+1}, x_{k_1+2}, \ldots, x_{k_1+k_2}),\\
\ldots\\
X_u &= (x_{k_1+k_2+\ldots+k_{u-1}+1}, x_{k_1+k_2+\ldots+k_{u-1}+2}, \ldots, x_{n-1}, x_n),
\end{aligned}
$$

and $|X_i| = k_i$. That is, in the fixed-order partition of X, the variable order of X is fixed.

Definition 2.3 *When the variable order of $X = (x_1, x_2, \ldots, x_n)$ is not fixed, a partition of X is called a* **non-fixed-order partition** *of X.*

We assume that the given logic function is completely specified and has no redundant variables.

Example 2.1 *Consider (X_1, X_2), which is a fixed-order partition of X, where $X = (x_1, x_2, x_3, x_4, x_5)$ and each x_i is a binary variable. When $X_1 = (x_1, x_2)$ and $X_2 = (x_3, x_4, x_5)$, we have $k_1 = 2$, $k_2 = 3$, $P_1 = \{0, 1, 2, 3\}$, and $P_2 = \{0, 1, \ldots, 7\}$. Note that X_1 takes 4 values, and X_2 takes 8 values. So, a 5-variable logic function $f(X)$ can be represented by the multi-valued function $f(X_1, X_2)$: $P_1 \times P_2 \rightarrow B$. (End of Example)*

2.2. Heterogeneous MDD

In this paper, we use standard terminologies for BDDs, reduced ordered binary decision diagrams (ROBDDs) [5], MDDs, and reduced ordered multi-valued decision diagrams (ROMDDs) [12].

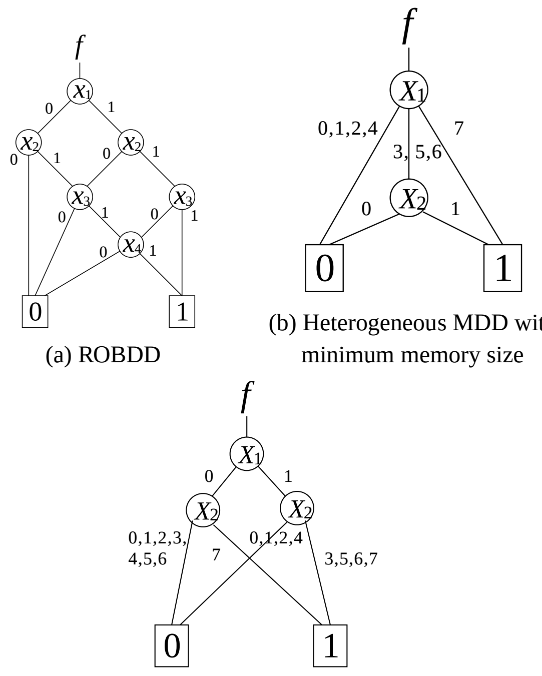
Definition 2.4 *When $X = (x_1, x_2, \ldots, x_n)$ is partitioned into $(X_1, X_2, \ldots, X_u)$, the ROMDD representing a logic function $f(X)$ is called a* **heterogeneous MDD**. *A heterogeneous MDD represents a mapping $f : P_1 \times P_2 \times \ldots \times P_u \rightarrow B$, where $P_i = \{0, 1, \ldots, 2^{k_i} - 1\}$ and $B = \{0, 1\}$. In a heterogeneous MDD, non-terminal nodes representing a super variable X_i have 2^{k_i} outgoing edges, where k_i denotes the number of binary variables in X_i.*

Definition 2.5 *In a decision diagram (DD), the* **number of nodes in the DD**, *denoted by $nodes(DD)$, includes only non-terminal nodes.*

Definition 2.6 *The* **width of the DD with respect to** *X_i, denoted by $width(DD, i)$, is the number of nodes in the DD corresponding to the super variable X_i. The number of nodes in the MDD with the partition $(X_1, X_2, \ldots, X_u)$ is given by*

$$nodes(MDD) = \sum_{i=1}^{u} width(MDD, i).$$

Example 2.2 *Consider the logic function $f = x_1x_2x_3 \vee x_2x_3x_4 \vee x_3x_4x_1 \vee x_4x_1x_2$. Fig. 2.1(a) and Fig. 2.1(b,c) represent the ROBDD and the heterogeneous MDDs for f, respectively. In Fig. 2.1(b), the binary variables $X = (x_1, x_2, x_3, x_4)$ are partitioned into (X_1, X_2),*



(a) ROBDD

(b) Heterogeneous MDD with minimum memory size

(c) Heterogeneous MDD with maximum memory size

Figure 2.1. BDD and heterogeneous MDDs

where $X_1 = (x_1, x_2, x_3)$ and $X_2 = (x_4)$. In Fig. 2.1(c), $X_1 = (x_1)$ and $X_2 = (x_2, x_3, x_4)$. (End of Example)

In this paper, we use Shared BDDs (SBDDs) [15] and Shared MDDs (SMDDs) to represent multiple-output functions $F = (f_0, f_1, \ldots, f_{m-1})$: $B^n \rightarrow B^m$, where $B = \{1, 0\}$, and n and m denote the number of input and output variables, respectively. In the following, BDDs and MDDs mean SBDDs and SMDDs, respectively, unless stated otherwise.

2.3. Memory Sizes

Definition 2.7 *The* **memory size of a DD** *is the number of words needed to represent the DD in memory, where we assume that a word is large enough to store an index or an edge pointer.*

In a memory, a non-terminal node requires an index and pointers to the succeeding nodes. Since each non-terminal node in a BDD has two pointers, the memory size needed to represent a BDD is

$$(2+1) \times nodes(\text{BDD}). \tag{2.1}$$

In a heterogeneous MDD, each super variable can take different domain. Therefore, the memory size for a heterogeneous MDD is given by

$$\sum_{i=1}^{u}(2^{k_i}+1) \times width(\text{heterogeneous MDD}, i).$$

Example 2.3 *The memory sizes to represent BDD and heterogeneous MDDs are as follows: for the BDD in Fig. 2.1(a),* 18*; for the heterogeneous MDD in Fig. 2.1(b),* 12*; and for the heterogeneous MDD in Fig. 2.1(c),* 21. *(End of Example)*

Definition 2.8 *Given a logic function f and the order of input variables, the* **fixed-order minimum heterogeneous MDD** *for f is the heterogeneous MDD with the minimum memory size among all possible fixed-order partitions of the variables.*

Definition 2.9 *Given a logic function f, the* **minimum heterogeneous MDD** *for f is the heterogeneous MDD with the minimum memory size among all possible non-fixed-order partitions of the variables.*

2.4. Average Path Lengths (APLs)

Definition 2.10 *In a DD, a sequence of edges and nodes leading from the root node to a terminal node is a* **path**. *The number of non-terminal nodes on the path is the* **path length**.

In this paper, we assume the following computational model:

1. DDs for logic functions are evaluated by traversing nodes from the root node to a terminal node according to values of the input variables.
2. MDDs are implemented directly, not simulated using the BDD package as described in [12].
3. Encoded input values are available, and their access time is negligible. For example, when $X_1 = (x_1, x_2, x_3, x_4) = (1, 0, 0, 1)$, $X_1 = 9$ is immediately available as an input to the algorithm.
4. Most computation time is devoted to accessing nodes.
5. The evaluation time for all MDD nodes are equal.

In this case, the time to evaluate a DD for a logic function is proportional to the number of non-terminal nodes on the path (i.e., path length). Furthermore, we assume that each binary variable occurs as a 0 with the same probability as a 1. Under these assumptions, we use the **average path length** or **APL** to estimate the evaluation time of different types of DDs.

In this paper, we use a Shared Decision Diagram (SDD) to represent multiple-output functions $F = (f_0, f_1, \ldots, f_{m-1})$. The APL of an SDD is the sum of the APLs of individual DDs for each function f_i [27].

Algorithm 3.1

```
1:  exhaustive_search (BDD, memory size limitation $L$) {
2:    $min\_APL$ = **minimize_APL** (1, $L$) ;
3:    for (all permutations of binary variables) {
4:      Generate the next variable order for BDD ;
5:      if ($L < nodes(BDD) + 2$) continue ;
6:      $memory$ = **minimize_memory** ( ) ;
7:      if ($L < memory$) continue ;
8:      $APL$ = **minimize_APL** (1, $L$) ;
9:      if ($APL < min\_APL$) {
10:       $min\_APL = APL$ ;
11:       Record the variable order for the BDD ;
12:       Record the partition of binary variables ;
13:     }
14:   }
15: }
```

Figure 3.1. Exact APL minimization algorithm for heterogeneous MDDs

3. APL Minimization

The APLs in heterogeneous MDDs depend on partitions and orderings of input variables X.

Example 3.1 *For the BDD in Fig. 2.1(a), the APL is* 3.125*; for the heterogeneous MDD in Fig. 2.1(b), it is* 1.375*; and for the heterogeneous MDD in Fig. 2.1(c), it is* 2.0. *(End of Example)*

The partition of X that minimizes the APL is the trivial partition, $X = X_1$, where $k_1 = n$. However, the memory size needed to represent the heterogeneous MDD for the trivial partition is nearly 2^n. Thus, such an heterogeneous MDD is impractical for most cases. Therefore, we find a partition with an order of X that minimizes the APL within the given memory size limitation. We formulate the APL minimization problem considering both partitions and orderings of binary variables as follows:

Problem 3.1 *Given a logic function f and a memory size limitation L, find a non-fixed-order partition of X that produces the heterogeneous MDD with the minimum APL and with the memory size equal to or smaller than L.*

3.1. Exact Minimization Algorithm

Fig. 3.1 shows pseudo-code to solve Problem 3.1. It uses a BDD for the given logic function as the internal representation. In the 2nd and 8th lines in Fig. 3.1, procedure **minimize_APL** [19, 22] finds an optimum fixed-order partition. Since it is recursive procedure, the top level for BDD (i.e. level = 1) is required as the initial argument. In the 5th line, a theorem in [20] is used to reduce the computation time.

Algorithm 3.2

```
1:  sifting_APL (BDD, $L$, #sifting rounds $R$) {
2:      $cost$ = minimize_APL (1, $L$) ;
3:      for ($r$ = 0; $r < R$; $r$++) {
4:          for ($\forall x_i \in X$) {
5:              $best\_p$ = current position of $x_i$ ;
6:              for (all positions $p$) {
7:                  Move $x_i$ to position $p$ ;
8:                  $memory$ = minimize_memory ( ) ;
9:                  Compute $L_{mem}$ ;
10:                 if ($L \leq L_{mem}$) break ;
11:                 if ($L < memory$) continue ;
12:                 $APL$ = minimize_APL (1, $L$) ;
13:                 if ($APL < cost$) {
14:                     $cost$ = $APL$ ;
15:                     $best\_p$ = $p$ ;
16:                     Record the partition of binary variables ;
17:                 }
18:             }
19:             Move $x_i$ to $best\_p$ ;
20:         }
21:     }
22: }
```

Figure 3.2. Heuristic APL minimization for heterogeneous MDDs

The procedure **minimize_memory** [20, 22] in the 6th line finds a fixed-order partition that produces the fixed-order minimum heterogeneous MDD. Algorithm 3.1 finds an optimum solution using exhaustive search, which enumerates all possible variable orders.

3.2. Heuristic Minimization

Although Algorithm 3.1 obtains an exact minimum solution, it is time-consuming for functions with many inputs; i.e. it enumerates all possible variable orders.

In this section, we show a heuristic APL minimization method for heterogeneous MDDs (Algorithm 3.2) using a sifting algorithm [24] and the fixed-order partition algorithm [19]. The sifting algorithm repeatedly performs the following basic steps:

1. Change the variable order.
2. Compute the cost.

Most sifting algorithms use the number of nodes in a BDD as the cost in their heuristics. Algorithm 3.2, however, uses the APL of a heterogeneous MDD as the cost. The APL for a heterogeneous MDD can be computed using a method similar to the APL of BDDs in [21]. Fig. 3.2 shows a pseudo-code for the heuristic algorithm. It also uses a BDD for the given logic function as the internal representation. In this algorithm, each variable x_i is sifted across all possible positions to determine its best position. First, x_i is sifted in one direction to the closer extreme (top or bottom). Then, x_i is sifted in the opposite direction to the other extreme. In the 10th line in Fig. 3.2, a property in [23] is used to find useful siftings of x_i. The L_{mem} denotes the memory size of fixed-order minimum heterogeneous MDD for logic function g or h, where $f(X) = g(h(X_1), X_2)$, X_1 contains the binary variables above x_i, and X_2 contains the binary variables below x_i. When x_i moves down to the bottom of the BDD, we use h for L_{mem}. On the other hand, when variable x_i moves up to the top of the BDD, we use g for L_{mem}. This lower bound for the memory size is similar to the one introduced for the number of nodes during the classical sifting [7], except for it is heuristic.

4. Experimental Results

Experiments were conducted in the following environment: CPU: Pentium4 Xeon 2.8GHz, L1 Cache: 32KB, L2 Cache: 512KB, Memory: 4GB, OS: redhat (Linux 7.3), and C-Compiler: gcc -O2.

4.1. Comparison of Memory Sizes and APLs for n-Variable Logic Functions

Table 4.1 compares the memory sizes and the APLs of BDDs and heterogeneous MDDs for n-variable logic functions. The BDDs and heterogeneous MDDs are optimized using four different algorithms: (1) exact minimization algorithm of the number of nodes in a BDD (column "Pivot"); (2) exact APL minimization algorithm for a BDD considering only the orderings of binary variables (column "Ord."); (3) fixed-order partition algorithm [19] considering only the partitions of binary variables, that minimizes the APL of a heterogeneous MDD, where the order of binary variables is the same as the BDD with the minimum nodes (column "Part."); and (4) Algorithm 3.1 for heterogeneous MDD considering both partitions and orderings of binary variables (column "O & P"). The memory size limitation L of the fixed-order partition algorithm and Algorithm 3.1 is set to the memory size of the BDD with the minimum nodes. The values in the table are the normalized averages of n-variable logic functions with the memory sizes and APLs of "Pivot" set to 1.00. Columns "Ord.", "Part.", and "O & P" show the relative values of the memory sizes and APLs to "Pivot". The BDDs and heterogeneous MDDs in this table do not use complemented edges.

For 4 and 5-variable logic functions, we calculated the exact averages over all functions. We did this by recognizing that the APL of a function in one NPN-equivalence class [17, 26] is identical to the APLs of other functions in the same class. Thus, it is sufficient to consider only

Table 4.1. Memory sizes and APLs for BDDs and heterogeneous MDDs for randomly generated n-variable logic functions

	Memory size				APL				
	BDD		MDD		BDD		MDD		
n	Pivot	Ord.	Part.	O & P	Pivot	Ord.	Part.	O & P	#samples
4	1.00	1.07	0.87	0.86	1.00	0.99	0.39	0.37	2^{16}(all functions)
5	1.00	1.07	0.91	0.91	1.00	0.98	0.28	0.27	2^{32}(all functions)
6	1.00	1.08	0.80	0.80	1.00	0.97	0.35	0.32	1,000
7	1.00	1.08	0.78	0.79	1.00	0.97	0.28	0.27	1,000
8	1.00	1.08	0.79	0.81	1.00	0.97	0.23	0.22	1,000
9	1.00	1.07	0.81	0.83	1.00	0.98	0.20	0.19	1,000
10	1.00	1.06	0.83	0.84	1.00	0.98	0.17	0.17	1,000

one function from each class and form a sum weighted by the size of each class. For larger n, there are too many NPN-equivalence classes. For $6 \leq n \leq 10$, we generated $1,000$ pseudo-random n-variable logic functions with different number of true minterms, and calculated the normalized averages for them.

For BDDs, APLs can be reduced up to 97% of BDDs with the minimum nodes. However, for heterogeneous MDDs, the APLs can be reduced up to 17% of BDDs with the minimum nodes without increasing the memory sizes. Table 4.1 shows that the relative values of APLs for heterogeneous MDDs decreases as the number of binary variables n increases. Algorithm 3.1 finds smaller APLs than the fixed-order partition algorithm; it finds exact minimum APLs of heterogeneous MDDs for the functions with up to 11 variables within a reasonable computation time.

4.2. Comparison of Memory Sizes and APLs for Benchmark Functions

Table 4.2 compares memory sizes and APLs of BDDs and heterogeneous MDDs for benchmark functions. Columns labeled “Init.” denote the BDDs obtained by the best known variable orders [29]. These are used as the initial BDDs for the algorithms in this experiment. Columns “Ord.” denote the BDDs obtained by the sifting algorithm for the APLs [21]. The sifting algorithm minimizes the APLs considering only the permutations of binary variables. Columns “Part.” denote the heterogeneous MDDs obtained by the fixed-order partition algorithm [19, 22], where the order of binary variables is fixed with that of the initial BDD. And, columns “O & P” denote the heterogeneous MDDs obtained by Algorithm 3.2. The memory size limitation L of the fixed-order partition algorithm and Algorithm 3.2 is set to the memory size of the BDD in “Init.”. In the sifting algorithm [21] and Algorithm 3.2, the number of rounds of sifting is set to two. The BDDs and heterogeneous MDDs in this table use complemented edges. The APLs in Table 4.2 may not be exact minimum since the algorithms are heuristic methods except for the fixed-order partition algorithm. The row labeled *Average of ratios* represents the normalized averages of memory size and APL, where the memory size and the APL of “Init.” are set to 1.00.

The sifting algorithm reduced APLs to 88% of “Init.”, on average, but increased the memory sizes by twice. Especially, for *C880*, *C1908*, *i10*, and *too_large*, the sifting algorithm increased the memory sizes significantly. The fixed-order partition algorithm and Algorithm 3.2 reduced APLs to 62% and 51% of “Init.”, on average, respectively, without increasing the memory size. For most benchmark functions, the fixed-order partition algorithm reduced APLs substantially (e.g. *C499*). However, for some logic functions (e.g. *apex6* and *i8*), the fixed-order partition algorithm could not reduce APLs of “Init.” by much. Even for such functions, Algorithm 3.2 (“O & P”) could reduce APLs substantially without increasing the memory size.

Example 4.1 *Consider the* 7*-variable* 2*-output balanced tree (BTREE) function. Fig. 4.1(a) shows the BDD with the fewest nodes for the BTREE function. For this BDD, the algorithm considering only the orderings of binary variables cannot reduce the APL. The fixed-order partition algorithm cannot reduce the APL without increasing the memory size. However, the algorithm considering both partitions and orderings of binary variables can reduce the APL without increasing the memory size, and obtains the heterogeneous MDD in Fig. 4.1(b), where* $X_1 = (x_1)$, $X_2 = (x_2, x_4, x_5)$, *and* $X_3 = (x_3, x_6, x_7)$. *(End of Example)*

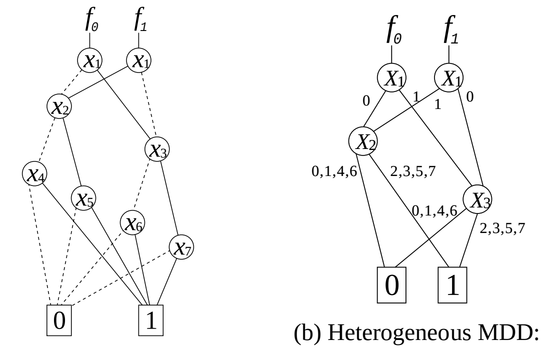
4.3. Comparison of Computation Time for Algorithms

Table 4.3 compares the computation time of the sifting algorithm [21], the fixed-order partition algorithm [19], and Algorithm 3.2. The values in Table 4.3 denotes the CPU times needed to obtain the BDDs and heterogeneous MDDs in Table 4.2, in seconds.

The sifting algorithm considering only the variable order “Ord.” and the fixed-order partition algorithm consid-

Table 4.2. Memory sizes and APLs for BDDs and heterogeneous MDDs for benchmark functions

			Memory size				APL			
			BDD		MDD		BDD		MDD	
Name	In	Out	Init.	Ord.	Part.	O & P	Init.	Ord.	Part.	O & P
C432	36	7	3189	3243	3180	3179	86.58	86.24	48.51	45.45
C499	41	32	77595	96315	77586	77589	813.64	641.16	215.64	192.52
C880	60	26	12156	54810	12155	12154	135.79	121.03	112.54	99.13
C1908	33	25	16575	56328	16570	16564	254.35	183.61	112.23	92.09
C2670	233	64	5319	8286	5317	5319	214.05	202.08	157.11	133.78
C3540	50	22	71481	74292	71472	71480	209.15	208.06	106.20	91.78
C5315	178	123	5154	5460	5154	5153	462.05	446.26	395.91	304.38
C7552	207	107	6633	6585	6633	6633	484.03	469.54	412.64	314.03
alu4	14	8	1047	1080	1019	1019	40.81	40.70	19.59	19.59
apex1	45	45	3735	4254	3734	3728	180.59	177.69	76.47	67.63
apex6	135	99	1491	1887	1491	1490	291.54	230.91	260.79	231.06
cps	24	102	2910	4656	2906	2906	290.25	235.39	164.67	151.81
dalu	75	16	2064	2970	2063	2064	102.67	78.81	45.78	28.09
des	256	245	8832	9177	8830	8831	1210.00	1080.38	810.38	687.50
frg2	143	139	2886	5070	2885	2884	624.69	322.17	536.64	348.60
i3	132	6	396	396	396	396	26.76	26.76	13.87	12.61
i8	133	81	3825	6954	3825	3825	302.54	270.82	302.54	207.54
i10	257	224	61977	685215	61977	61974	1084.96	776.10	817.63	614.53
k2	45	45	3735	4254	3728	3728	180.52	177.69	77.29	67.61
too_large	38	3	954	2361	953	954	13.16	11.52	6.55	6.24
vda	17	39	1431	1515	1413	1424	176.34	171.54	79.13	69.54
Average of ratios			1.00	2.03	1.00	1.00	1.00	0.88	0.62	0.51



(a) BDD: $APL = 6.0$

(b) Heterogeneous MDD: $APL = 4.0$

Figure 4.1. BDD and heterogeneous MDD for BTREE function

ering only the partition of binary variables "Part." require shorter CPU time than Algorithm 3.2 "O & P". Especially, the fixed-order partition algorithm can reduce APLs substantially in a short computation time for many benchmark functions. Thus, the fixed-order partition algorithm is the most efficient among these algorithms. However, the algorithm considering both partitions and orderings of binary variables (Algorithm 3.1 or 3.2) is also important, since for some logic functions, the fixed-order partition algorithm cannot reduce APLs as shown in the previous section. For example, in software synthesis, the fixed-order partition algorithm is useful for generating a program code for debugging. On the other hand, Algorithm 3.2 is useful for generating an optimized program code (final version).

Table 4.3. CPU times [sec] for APL minimization algorithms

Name	Ord. [21]	Part. [19]	O & P (Algorithm 3.2)
C432	0.23	0.01	1.04
C499	10.76	0.75	698.31
C880	4.54	0.02	22.09
C1908	1.44	0.09	27.38
C2670	2.21	0.14	1957.51
C3540	12.74	0.55	523.45
C5315	0.43	0.09	3663.57
C7552	1.35	0.09	2258.88
alu4	0.02	0.01	0.05
apex1	0.11	0.02	36.07
apex6	0.05	0.01	79.47
cps	0.09	0.01	0.80
dalu	0.15	0.05	132.41
des	0.91	0.87	60144
frg2	0.29	0.01	218.46
i3	0.01	0.01	95.69
i8	0.31	0.01	30.15
i10	160.91	2.69	71464
k2	0.11	0.03	33.99
too_large	0.07	0.01	0.31
vda	0.02	0.01	0.15

5. Conclusion and Comments

In this paper, we have proposed an exact and a heuristic minimization algorithm for APL in heterogeneous MDDs. Experimental results with 21 benchmark functions show that: 1) In heterogeneous MDDs, APLs can be reduced by a half of corresponding BDDs, on average, without increasing the memory size. 2) Algorithm 3.1 considering both partitions and orderings of binary variables finds heterogeneous MDDs with the minimum APLs for functions with up to 11 variables within a reasonable time. 3) The algorithm considering only partitions of binary variables (i.e. fixed-order partition algorithm) reduces APLs faster than algorithms considering variable order. However, for some logic functions, the fixed-order partition algorithm cannot reduce APLs by much. Therefore, such functions require an algorithm considering both partitions and orderings of binary variables (i.e. Algorithm 3.1 or 3.2).

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 funds from Ministry of Education, Culture, Sports, Science, and Technology (MEXT) via Kitakyushu innovative cluster project. Prof. Jon. T. Butler improved English presentation.

References

[1] P. Ashar and S. Malik, “Fast functional simulation using branching programs,” *ICCAD’95*, pp. 408–412, Nov. 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 embedded control applications,” *IEEE Trans. CAD*, Vol. 18, No. 6, pp.834-849, June 1999.

[3] B. Becker and R. Drechsler, “Efficient graph based representation of multivalued functions with an application to genetic algorithms,” *Proc. of International Symposium on Multiple Valued Logic*, pp. 40-45, May 1994.

[4] F. Brglez and H. Fujiwara, “Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN,” *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems*, June 1985, pp. 663-698.

[5] R. E. Bryant, “Graph-based algorithms for boolean function manipulation,” *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 677–691, Aug. 1986.

[6] J. T. Butler and T. Sasao, “On the average path length in decision diagrams of multiple-valued functions,” *33rd International Symposium on Multiple-Valued Logic*, pp. 383-390, Tokyo, Japan, May 16-19, 2003.

[7] R. Drechsler, W. Günther, and F. Somenzi, “Using lower bounds during dynamic BDD minimization,” *IEEE Trans. CAD*, Vol. 20 (1), pp. 51–57, Jan. 2001.

[8] R. Ebendt, S. Hoehne, W. Guenther, and R. Drechsler, “Minimization of the expected path length in BDDs based on local changes,” *Asia and South Pacific Design Automation Conference (ASP-DAC’2004)*, pp. 866-871, Yokohama, Japan, Jan. 2004.

[9] M. Fujita, Y. Matsunaga, and T. Kakuda, “On variable ordering of binary decision diagrams for the application of multi-level logic synthesis,“ *EDAC*, pp. 50–54, Mar. 1991.

[10] N. Ishiura, H. Sawada, and S. Yajima, “Minimization of binary decision diagrams based on exchanges of variables,“ *ICCAD*, pp. 472–475, Nov. 1991.

[11] Y. Jiang and B. K. Brayton, “Software synthesis from synchronous specifications using logic simulation techniques,” *Design Automation Conference*, pp. 319-324, New Orleans, LA, U.S.A, June 10-14, 2002.

[12] T. Kam, T. Villa, R. K. Brayton, and A. L. Sagiovanni-Vincentelli, “Multi-valued decision diagrams: Theory and Applications,” *International Journal on Multiple-Valued Logic*, Vol. 4, No. 1-2, pp. 9–62, 1998.

[13] Y. Y. Liu, K. H. Wang, T. T. Hwang, C. L. Liu, “Binary decision diagrams with minimum expected path length,“ *Proc. DATE 01*, pp. 708–712, Mar. 13-16, 2001.

[14] 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.

[15] S. Minato, N. Ishiura, and S. Yajima, “Shared binary decision diagram with attributed edges for efficient Boolean function manipulation,” *Proc. 27th ACM/IEEE Design Automation Conf.*, pp. 52–57, June 1990.

[16] D. M Miller, “Multiple-valued logic design tools,” *Proc. of International Symposium on Multiple Valued Logic*, pp. 2–11, May 1993.

[17] S. Muroga, *Logic Design and Switching Theory*, Wiley-Interscience Publication, 1979.

[18] S. Nagayama, T. Sasao, Y. Iguchi, and M. Matsuura, “Representations of logic functions using QRMDDs,” *32nd International Symposium on Multiple-Valued Logic*, pp. 261-267, Boston, Massachusetts, U.S.A., May 15-18, 2002.

[19] S. Nagayama and T. Sasao, “Code generation for embedded systems using heterogeneous MDDs,” *the 12th workshop on Synthesis And System Integration of Mixed Information technologies (SASIMI 2003)*, pp. 258-264, Hirosima, Japan, April 3-4, 2003.

[20] S. Nagayama and T. Sasao, “Compact representations of logic functions using heterogeneous MDDs,” *33rd International Symposium on Multiple-Valued Logic*, pp. 247-252, Tokyo, Japan, May 16-19, 2003.

[21] S. Nagayama, A. Mishchenko, T. Sasao, and J. T. Butler, “Minimization of average path length in BDDs by variable reordering,” *International Workshop on Logic and Synthesis*, pp. 207-213, Laguna Beach, California, U.S.A., May 28-30, 2003.

[22] S. Nagayama and T. Sasao, “Compact representations of logic functions using heterogeneous MDDs,” *IEICE Trans. on fundamentals*, Vol. E86-A, No. 12, pp. 3168-3175, Dec., 2003.

[23] S. Nagayama and T. Sasao, “Minimization of memory size for heterogeneous MDDs,” *Asia and South Pacific Design Automation Conference (ASP-DAC’2004)*, pp. 872-875, Yokohama, Japan, Jan. 2004.

[24] R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” *ICCAD’93*, pp. 42–47.

[25] T. Sasao and M. Fujita (ed.), *Representations of Discrete Functions*, Kluwer Academic Publishers 1996.

[26] T. Sasao, *Switching Theory for Logic Synthesis*, Kluwer Academic Publishers, 1999.

[27] T. Sasao, Y. Iguchi, and M. Matsuura, “Comparison of decision diagrams for multiple-output logic functions,” *International Workshop on Logic and Synthesis*, New Orleans, Louisiana, June 4-7, 2002, pp.379-384.

[28] T. Sasao, J. T. Butler, and M. Matsuura, “Average path length as a paradigm for the fast evaluation of functions represented by binary decision diagrams,” *International Symposium on New Paradigm VLSI Computing*, Sendai, Japan, Dec. 12-14, 2002.

[29] F. Somenzi, “CUDD: CU Decision Diagram Package Release 2.3.1,” University of Colorado at Boulder, 2001.

[30] S. Yang, *Logic synthesis and optimization benchmark user guide version 3.0*, MCNC, Jan. 1991.