On the Minimization of Longest Path Length for Decision Diagrams

Shinobu NAGAYAMA
Department of CSE,
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 consider the minimization of the longest path length (LPL) for binary decision diagrams (BDDs) and heterogeneous multi-valued decision diagrams (MDDs). Experimental results show that: (1) For many logic functions, node minimization of BDDs also minimizes the LPLs of BDDs. (2) When we use heterogeneous MDDs for representing logic functions and minimize the memory sizes of heterogeneous MDDs, we can reduce both memory sizes and LPLs to 86% *and* 72% *of corresponding BDDs, respectively. On the other hand, when the memory limitations are set to the memory sizes of BDDs, the LPLs can be reduced to* 54% *of BDDs.*

1 Introduction

Binary decision diagrams (BDDs) [4] and multi-valued decision diagrams (MDDs) [8] are extensively used to represent logic functions, and various optimization algorithms for BDDs and MDDs have been proposed. Most optimization algorithms for decision diagrams (DDs) minimize the number of nodes in DDs. However, logic simulation [1, 12] and software synthesis [2, 7, 16] require DDs with smaller path length, as well as fewer nodes. In the evaluation of logic functions using DDs, the evaluation time is proportional to the path length of DDs. Therefore, minimization of the path length reduces evaluation time of logic functions. The minimization of average path length (APL) proposed in [6, 11, 16, 18] reduces the average evaluation time of logic functions. While in logic simulation, the minimization of average evaluation time is very useful, in embedded system using Real-Time Operating System (RTOS) [2, 9, 22, 28] and Pass Transistor Logic (PTL) [5, 10, 25], the minimization of longest evaluation time is more important.

Since software programs on RTOS have to finish execution within a specified time, the accurate and fast estimation of the longest execution time is important for task scheduling [2, 9, 22, 28]. In software synthesis using DDs, the longest execution time of the generated software programs depends on the longest path lengths (LPLs) of DDs. Thus, the minimization of LPLs can reduce the longest execution time, and will allow to execute more tasks within a specified time. Similarly, critical paths in PTL circuits can be derived directly from BDDs representing logic functions by considering the LPLs of the BDDs. Thus, LPL minimization of BDDs can reduce the delay time of circuits [5, 10, 25].

In this paper, we consider the minimization of LPLs for BDDs and heterogeneous MDDs. Since the LPL of a BDD, as well as the number of nodes, depends on the variable ordering, we can reduce the LPL of the BDD by trying different variable orderings. On the other hand, in heterogeneous MDDs, both orderings and partitions of binary variables can be optimized for the LPL minimization [16, 17, 19, 20]. Our experimental results show that for many logic functions, node minimization of BDDs minimizes the LPLs of BDDs, as well. Also, we show that the LPLs of the heterogeneous MDDs can be reduced to 54% of the LPLs of corresponding BDDs, on average, without increasing memory size.

This paper is organized as follows: Section 2 shows the necessary terminology and definitions. Section 3 considers the LPL minimization for BDDs and shows experimental results for BDDs. Section 4 considers the LPL minimization for heterogeneous MDDs and compares heterogeneous MDDs with BDDs.

2 Preliminaries

In this paper, we use standard terminologies for BDDs, reduced ordered binary decision diagrams (ROBDDs) [4], MDDs, and reduced ordered multi-valued decision diagrams (ROMDDs) [8].

2.1 Heterogeneous MDD

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. Let $X_i \subseteq X$. If $\{X\} = \{X_1\} \cup \{X_2\} \cup \ldots \cup \{X_u\}$, $\{X_i\} \neq \phi$,* *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*

$$\begin{aligned} X_1 &= (x_1, x_2, \ldots, x_{k_1}), \\ X_2 &= (x_{k_1+1}, x_{k_1+2}, \ldots, x_{k_1+k_2}), \\ \ldots& \\ X_u &= (x_{k_1+k_2+\ldots+k_{u-1}+1}, x_{k_1+k_2+\ldots+k_{u-1}+2}, \ldots, x_{n-1}, x_n), \end{aligned}$$

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

In this paper, we will call a fixed-order partition simply a **partition** of X. And, we assume that the given logic function is completely specified and has no redundant variables.

Definition 2.3 *When $X = (x_1, x_2, \ldots, x_n)$ is partitioned into $(X_1, X_2, \ldots, X_u)$, the MDD 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 2.4 *In a DD for logic function f, the* **number of nodes in the DD**, *denoted by $nodes(DD)$, is the sum of all non-terminal nodes.*

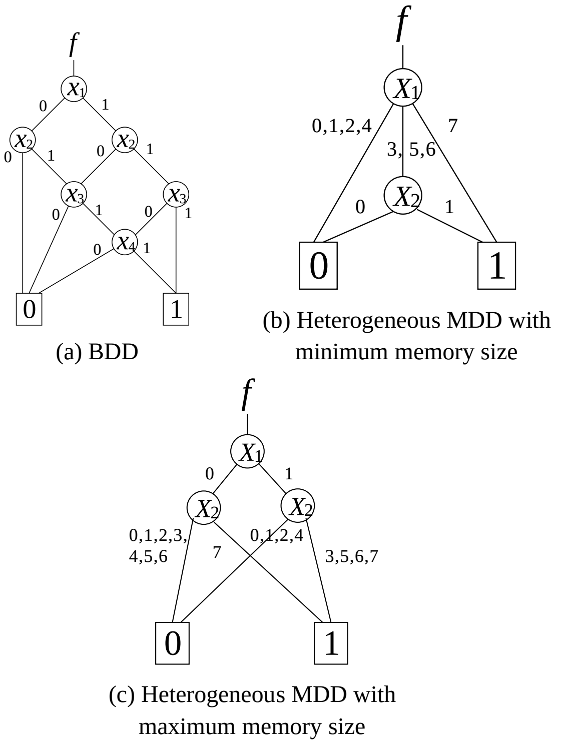
Definition 2.5 *The* **width of the DD with respect to** *X_i*[1], *denoted by $width(DD, i)$, is the number of nodes in the DD corresponding to the super variable X_i. The number of nodes in the MDD with the partition $(X_1, X_2, \ldots, X_u)$ is given by*

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

Example 2.1 *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 BDD 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), 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 decision diagrams (SDDs) [13] to represent multiple-output functions $F =$

[1]Note that this definition differs from that of "width of BDDs" in [14].



(a) BDD

(b) Heterogeneous MDD with minimum memory size

(c) Heterogeneous MDD with maximum memory size

Figure 2.1. BDD and heterogeneous MDDs

$(f_0, f_1, \ldots, f_{m-1})$: $B^n \rightarrow B^m$, 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.

2.2 Memory Size of Decision Diagram

Definition 2.6 *In a DD for logic function f, the* **memory size of the DD**, *denoted by $Mem(DD)$, 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, each 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

$$Mem(\text{BDD}) = (2 + 1) \times nodes(\text{BDD})$$

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

$$\begin{aligned} &Mem(\text{heterogeneous MDD}) \\ &\quad = \sum_{i=1}^{u} (2^{k_i} + 1) \times width(\text{heterogeneous MDD}, i). \end{aligned}$$

Example 2.2 *The memory sizes to represent BDD and heterogeneous MDDs are as follows: for the BDD in Fig. 2.1(a), it is* 18*; for the heterogeneous MDD in Fig. 2.1(b), it is* 12*; and for the heterogeneous MDD in Fig. 2.1(c), it is* 21. *(End of Example)*

2.3 Longest Path Length (LPL) of Decision Diagram

Definition 2.7 *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**.

Definition 2.8 **The longest path length (LPL) of a DD**, *denoted by $LPL(DD)$, is the length of the longest path.*

In an SDD for a multiple-output function $F = (f_0, f_1, \ldots, f_{m-1})$: $B^n \rightarrow B^m$, we need to evaluate each single-output function f_i one by one. Therefore, in this paper, we define the LPL of an SDD for a multiple-output function F as follows.

Definition 2.9 *The LPL of an SDD for a multiple-output function $F = (f_0, f_1, \ldots, f_{m-1})$: $B^n \rightarrow B^m$, denoted by $LPL_m(SDD)$, is given by*

$$LPL_m(SDD) = \sum_{i=0}^{m-1} LPL(DD_i),$$

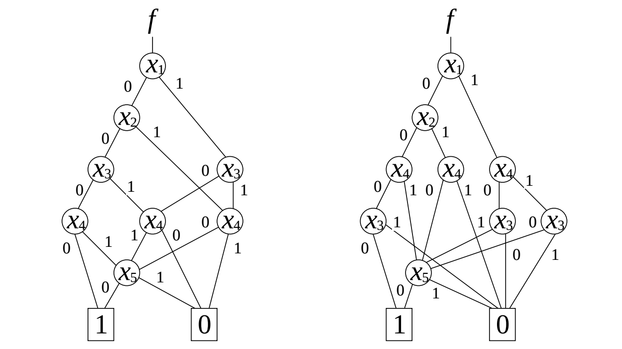
where DD_i represents a single output function f_i $(i = 0, 1, \ldots, m-1)$.

For a single-output logic function f, the least upper bound on LPL, denoted by $\lambda(f)$, is the number of variables on which the function depends. For a multiple-output logic function F, the least upper bound on LPL, denoted by $\lambda(F)$ is given by

$$\lambda(F) = \sum_{i=0}^{m-1} \lambda(f_i).$$

In this paper, we assume the following computational model:

1. The logic functions are evaluated by traversing DDs from the root node to a terminal node according to values of the input variables.
2. 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.
3. Most of the computation time is devoted to accessing nodes.
4. The evaluation time for all MDD nodes is the same.



(a) BDD with the fewest nodes (b) BDD with the smallest LPL

Figure 3.1. Relation between LPL and variable ordering for BDD

In this case, the longest time to evaluate a DD for a logic function is proportional to the LPL of the DD. Thus, in this model, we can fairly compare the LPLs of BDDs and MDDs.

Example 2.3 *For the BDD in Fig. 2.1(a), the LPL is* 4. *For the heterogeneous MDDs in Fig. 2.1(b, c), the LPLs are* 2. *(End of Example)*

3 LPL Minimization for BDDs

3.1 LPL Minimization Using Permutation of Binary Variables

Example 3.1 *Consider the logic function $f = \bar{x}_1 x_2 \bar{x}_4 \bar{x}_5 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_3 x_4 \bar{x}_5 \vee \bar{x}_1 \bar{x}_2 x_4 \bar{x}_5 \vee x_1 x_3 \bar{x}_4 \bar{x}_5$. Fig. 3.1 illustrates the BDD with the fewest nodes and the BDD with the smallest LPL for f. For the BDD in Fig. 3.1(a), $nodes(BDD) = 8$, $LPL(BDD) = 5$, and the variable order $= (x_1, x_2, x_3, x_4, x_5)$. For the BDD in Fig. 3.1(b), $nodes(BDD) = 9$, $LPL(BDD) = 4$, and the variable order $= (x_1, x_2, x_4, x_3, x_5)$.* *(End of Example)*

As shown in Example 3.1, since the LPL of the BDD depends on the variable ordering, the LPL minimization for a BDD is a problem of finding the optimum variable ordering for the BDD. We formulate the LPL minimization problem for a BDD as follows:

Problem 3.1 *Given a logic function $f(X)$, find a variable ordering for X that minimizes the LPL of the BDD.*

The heuristic algorithm for Problem 3.1, which uses the sifting algorithm [23], has been proposed in [5, 10, 25]. In this paper, we implement the simplest sifting algorithm for LPL minimization that uses the LPL as the cost function. We call it *LPL sifting*. This dedicated LPL minimization

Table 3.1. Comparison of the smallest LPLs and the least upper bounds on LPL of BDDs

n	$\lambda(f)$	Smallest LPL	#samples
4	3.98	3.89	2^{16}
5	5.00	4.99	2^{32}
6	5.92	5.92	1,000
7	6.99	6.99	1,000
8	7.99	7.99	1,000
9	8.99	8.99	1,000
10	9.99	9.99	1,000

algorithm requires more computation time than the original sifting and the APL sifting [6].

3.2 LPLs of BDDs for *n*-Variable Logic Functions

Table 3.1 compares the smallest LPLs with the least upper bounds on LPLs of BDDs for randomly generated n-variable logic functions. Note that all the n-variable logic functions are single output. In Table 3.1, the column labeled "n" denotes the number of variables. Column "#samples" denotes the number of sample functions used for each n-variable function. Columns "$\lambda(f)$" and "Smallest LPL" show the averages of the least upper bounds on LPL and the smallest LPLs for n-variable functions, respectively. Note that BDDs 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 LPL of a function in one NPN-equivalence class [15, 24] is identical to the LPLs of other functions in the same class. Thus, it is sufficient to consider only one function from each class and to form a sum weighted by the size of each class. For larger n, there are too many NPN-equivalence classes. Therefore, 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 averages for them.

For 4-variable functions, the smallest LPL of BDDs was 98% of $\lambda(f)$, on average. For 5-variable functions, the smallest LPL of BDDs was almost equal to $\lambda(f)$. For $6 \sim 10$-variable functions, the smallest LPLs of BDDs were exactly same as $\lambda(f)$.

For 91% of all 4-variable functions, the smallest LPLs were equal to $\lambda(f)$. Similarly, for 99% of all 5-variable functions; and for almost all $6 \sim 10$-variable functions used in this table, the smallest LPLs and $\lambda(f)$ were identical.

These experimental results show that the function in Example 3.1 is a rare case. From these experimental results, we obtain the following:

Observation 3.1 *For most randomly generated functions, the LPLs of BDDs are independent of the variable ordering.*

3.3 LPLs of BDDs for Benchmark Functions

Table 3.2 compares the numbers of nodes and LPLs of BDDs for 21 selected benchmark functions. The column labeled "$\lambda(F)$" denotes the least upper bounds on LPLs for the benchmark functions. Columns labeled "MinNodes" denote the BDDs obtained by the best known variable orders [26], that minimize the number of nodes. Columns "MinLPL" denote the BDDs obtained by the LPL sifting. For the LPL sifting, the number of rounds of sifting is set to two. The BDDs in this table use complemented edges. The numbers of nodes and LPLs in Table 3.2 may not be the exact minimum, since the algorithms are heuristic. The row labeled *Average of ratios1* represents the normalized averages of the numbers of nodes and LPLs, where the number of nodes and the LPL of "MinNodes" are set to 1.00. Similarly, the bottom row labeled *Average of ratios2* represents the normalized averages of LPLs, where each value in "$\lambda(F)$" is set to 1.00.

For these benchmark functions, the difference between $\lambda(F)$ and the LPLs obtained by LPL sifting is not so small. However, except for one function (*i8*), the difference between the LPLs of BDDs obtained by LPL sifting and the LPLs of BDDs in "MinNodes" is small. For *i*8, the LPL of BDD was reduced to 82% of "MinNodes", but the number of nodes increased by 172% of the original one. For *C*3540, although the difference of the numbers of nodes is large, the difference of LPLs is very small.

For 301 standard benchmark functions [3, 24, 27], we conducted similar experiments using BDDs without complemented edges. For also 301 benchmark functions, the difference between the least upper bounds on LPLs and the LPLs obtained by LPL sifting was not so small. On average, the LPL obtained by LPL sifting was 93% of the least upper bound on LPL. And, for 148 (49%) of 301 benchmark functions, LPLs minimized by LPL sifting were smaller than the least upper bounds on LPLs. On the other hand, the difference between the LPLs of BDDs obtained by LPL sifting and the LPLs of BDDs obtained by the original sifting was small. On average, the LPL of BDDs obtained by LPL sifting was just 99% of the LPL of BDDs obtained by the original sifting. For 211 (70%) of 301 benchmark functions, LPLs minimized by LPL sifting and LPLs of BDDs obtained by original sifting were identical. For function *c8*, the LPL of BDD was reduced to 91% of the LPL of BDDs obtained by the original sifting. It was the best case for LPL minimization in 301 benchmark functions.

From these experimental results, we obtain the following:

Observation 3.2 *For benchmark functions, LPLs of BDDs can be reduced by trying different variable orderings because the difference between the smallest LPL and the least upper bound on LPL is not so small. For many bench-*

Table 3.2. Numbers of nodes and LPLs of BDDs for 21 benchmark functions

				Number of nodes		LPL	
Name	In	Out	$\lambda(F)$	MinNodes	MinLPL	MinNodes	MinLPL
C432	36	7	225	1063	1063	225	225
C499	41	32	1312	25865	25865	1312	1312
C880	60	26	419	4052	5674	387	386
C1908	33	25	753	5525	5626	731	727
C2670	233	64	1057	1773	1790	491	490
C3540	50	22	713	23827	34996	454	452
C5315	178	123	2975	1718	1718	1429	1429
C7552	207	107	3496	2211	2237	2439	2433
alu4	14	8	70	349	349	70	70
apex1	45	45	814	1245	1245	543	543
apex6	135	99	759	497	510	651	640
cps	24	102	1637	970	970	1630	1630
dalu	75	16	635	688	688	272	272
des	256	245	2788	2944	2944	2211	2211
frg2	143	139	1763	962	2247	1626	1582
i3	132	6	132	132	132	132	132
i8	133	81	1260	1275	2195	1044	853
i10	257	224	5438	20659	61815	4634	4483
k2	45	45	814	1245	1245	543	543
too_large	38	3	107	318	403	107	104
vda	17	39	472	477	477	397	397
Average of ratios1			–	1.00	1.25	1.00	0.99
Average of ratios2			1.00	–	–	0.81	0.80

mark functions, the conventional sifting that minimizes the number of nodes in BDDs produces BDDs with reasonably small LPLs, comparable to LPLs obtained by LPL sifting.

From Observations 3.1 and 3.2, we conclude that for many logic functions, node minimization of BDDs minimizes the LPLs, as well.

4 LPL Minimization for Heterogeneous MDDs

4.1 LPL Minimization Using Both Permutation and Partition of Binary Variables

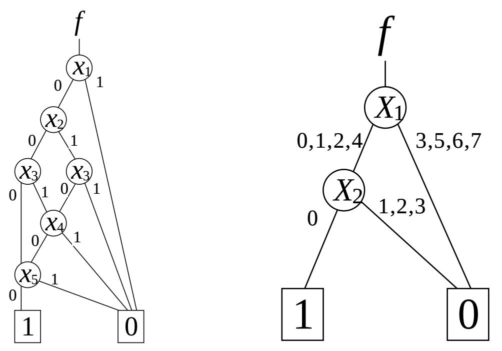
In this section, we consider both orderings and partitions of binary variables to minimize the LPL of a heterogeneous MDD. For any n-variable logic function, the trivial partition of X, where $X = X_1$ and $|X_1| = n$, produces a heterogeneous MDD with the smallest LPL (i.e., $LPL = 1$), independently of the variable ordering. However, since the memory size needed to represent the heterogeneous MDD for the trivial partition is nearly 2^n, such an heterogeneous MDD is too large in most cases. Therefore, we seek an ordering and a partition of X that minimizes the LPL within the given memory size limitation. We formulate the LPL minimization problem considering both orderings and partitions of binary variables as follows:

Problem 4.2 *Given a logic function $f(X)$ and a memory size limitation L, find an ordering and a partition of X that produces the heterogeneous MDD with the smallest LPL and with memory size equal to or smaller than L.*

Example 4.1 *Consider the logic function $f = \bar{x}_1\bar{x}_2\bar{x}_3\bar{x}_5 \vee \bar{x}_1\bar{x}_3\bar{x}_4\bar{x}_5 \vee \bar{x}_1\bar{x}_2\bar{x}_4\bar{x}_5$. Fig. 4.1(a) illustrates the BDD with the fewest nodes for f. Fig. 4.1(b) illustrates the heterogeneous MDD with the smallest LPL obtained by considering both orderings and partitions of binary variables, where the memory size limitation L is set to the memory size of the BDD in Fig. 4.1(a). In Fig. 4.1(b), $X_1 = (x_2, x_3, x_4)$ and $X_2 = (x_1, x_5)$. For the BDD in Fig. 4.1(a), $Mem(BDD) = 18$ and $LPL(BDD) = 5$. For the heterogeneous MDD in Fig. 4.1(b), $Mem(heterogeneous\ MDD) = 14$ and $LPL(heterogeneous\ MDD) = 2$. (End of Example)*

For logic functions with many binary variables, solving Problem 4.2 exactly within a reasonable time is difficult. That is, there may exist many different heterogeneous MDDs when both orderings and partitions of binary variables are considered for the optimization [20]. Thus, a heuristic algorithm is required for such logic functions. To develop an efficient heuristic algorithm, we modify Problem 4.2, based on a conjecture that reducing the number of variables also reduce the LPL, and reformulate the LPL minimization problem as follows.

Problem 4.3 *Given a logic function $f(X)$ and a memory size limitation L, find an ordering and a partition of X that*



(a) BDD with the fewest nodes (b) Heterogeneous MDD

Figure 4.1. Relation between LPL and partition of binary variables

Table 4.1. Memory sizes and LPLs of heterogeneous MDDs for n-variable logic functions

	Memory size		LPL		
n	BDD	MDD	BDD	MDD	#samples
4	1.00	0.87	1.00	0.32	2^{16}
5	1.00	0.87	1.00	0.31	2^{32}
6	1.00	0.87	1.00	0.34	1,000
7	1.00	0.81	1.00	0.29	1,000
8	1.00	0.73	1.00	0.26	1,000
9	1.00	0.68	1.00	0.23	1,000
10	1.00	0.67	1.00	0.21	1,000

produces a heterogeneous MDD with the fewest super variables X_i and with memory size equal to or smaller than L.

Although an optimum solution for Problem 4.3 is not always an optimum solution for Problem 4.2, the solution for Problem 4.3 will reduce the LPL of heterogeneous MDDs. Therefore, the heuristic algorithm for Problem 4.3 can be used for also Problem 4.2. We implement it and compare with the other algorithms. This heuristic algorithm is quite similar to the APL minimization algorithm in [21], except that the cost function is the number of super variables.

4.2 LPLs of Heterogeneous MDDs for n-Variable Logic Functions

Table 4.1 compares the memory sizes and the LPLs of BDDs and heterogeneous MDDs for the same logic functions as Table 3.1. The BDDs and heterogeneous MDDs are optimized using the following algorithms, respectively: 1) The exact minimization algorithm of the number of nodes in a BDD. 2) The exact LPL minimization algorithm for heterogeneous MDD considering both orderings and partitions of binary variables, where the memory size limitation L of this algorithm is set to the memory size of the BDD with the fewest nodes. The values in the table are the normalized averages of n-variable logic functions with the memory sizes and LPLs of BDDs set to 1.00. The columns "MDD" show the relative values of the memory sizes and LPLs for heterogeneous MDDs to BDDs. The BDDs and heterogeneous MDDs in this table do not use complemented edges.

As shown in Table 4.1, the LPLs of heterogeneous MDDs can be reduced to 21% of the LPLs of BDDs without increasing the memory size. Table 4.1 shows that the relative values of LPLs for heterogeneous MDDs decreases as the number of binary variables n increases.

Observation 4.1 *For single-output functions, LPLs of heterogeneous MDDs can be reduced substantially without increasing memory size.*

4.3 LPLs of Heterogeneous MDDs for 21 Benchmark Functions

Table 4.2 compares memory sizes and LPLs of BDDs and heterogeneous MDDs for the same benchmark functions as Table 3.2. Columns labeled "BDD" denote the BDDs obtained by the best known variable orders [26], that minimize the number of nodes. Columns "MinMem" denote the heterogeneous MDDs obtained by the memory minimization algorithm shown in [20]. Columns "MinLPL" denote heterogeneous MDDs obtained by the LPL minimization algorithm considering both orderings and partitions of binary variables, that minimizes the number of super variables. And, columns "MinAPL" denote the heterogeneous MDDs obtained by the APL minimization algorithm shown in [21]. The memory size limitation L of the LPL minimization and APL minimization algorithms for heterogeneous MDDs is set to the memory size of the BDD. And, for both the LPL minimization and APL minimization algorithms, the number of rounds of sifting is set to two. The BDDs and heterogeneous MDDs in this table use complemented edges. The memory sizes and LPLs in Table 4.2 may not be the exact minimum, since the algorithms are heuristic. The bottom row labeled *Average of ratios* represents the normalized averages of memory sizes and LPLs, where the memory size and the LPL of "BDD" are set to 1.00.

The memory minimization algorithm for heterogeneous MDDs reduced memory sizes to 86% and LPLs to 72% of corresponding BDDs, on average, respectively. The LPL minimization and APL minimization algorithms for heterogeneous MDDs reduced LPLs to 57% and 54% of BDDs, on average, respectively, without increasing memory size. Especially, for *C499*, the LPL of heterogeneous MDD was reduced to 27% of BDD. These experimental results show that the APL minimization algorithm for heterogeneous MDDs also can be used for the LPL minimization.

Table 4.2. Memory sizes and LPLs of heterogeneous MDDs for 21 benchmark functions

			Memory size				LPL			
				MDD				MDD		
Name	In	Out	BDD	MinMem	MinLPL	MinAPL	BDD	MinMem	MinLPL	MinAPL
C432	36	7	3189	2824	3176	3179	225	143	104	108
C499	41	32	77595	59739	77411	77589	1312	768	352	384
C880	60	26	12156	11812	12155	12154	387	329	255	254
C1908	33	25	16575	13493	16549	16564	731	465	268	301
C2670	233	64	5319	4650	5319	5319	491	397	330	280
C3540	50	22	71481	65029	71455	71480	454	357	269	239
C5315	178	123	5154	4582	5154	5153	1429	1273	1143	1035
C7552	207	107	6633	6199	6632	6633	2439	2003	1762	1395
alu4	14	8	1047	855	1019	1019	70	50	33	33
apex1	45	45	3735	3016	3734	3728	543	357	272	273
apex6	135	99	1491	1414	1491	1490	651	575	518	435
cps	24	102	2910	2533	2908	2906	1630	1106	743	835
dalu	75	16	2064	1548	2062	2064	272	181	132	150
des	256	245	8832	7288	8832	8831	2211	1478	1384	1227
frg2	143	139	2886	2671	2886	2884	1626	1298	1094	1095
i3	132	6	396	330	395	396	132	66	57	58
i8	133	81	3825	3662	3824	3825	1044	811	960	617
i10	257	224	61977	55766	61967	61974	4634	3434	3152	2902
k2	45	45	3735	3018	3733	3728	543	357	271	271
too_large	38	3	954	857	951	954	107	68	55	59
vda	17	39	1431	1088	1421	1424	397	280	189	199
Average of ratios			1.00	0.86	1.00	1.00	1.00	0.72	0.57	0.54

Observation 4.2 *When we use heterogeneous MDDs for representing logic functions and minimize the memory sizes of heterogeneous MDDs, we can reduce both memory sizes and LPLs to* 86% *and* 72% *of corresponding BDDs, respectively. On the other hand, when the memory limitations are set to the memory sizes of BDDs, the LPLs can be reduced to* 54% *of BDDs.*

5 Conclusion

In this paper, we have considered the LPL minimization for BDDs and heterogeneous MDDs. Experimental results show that: 1) For most randomly generated functions, the LPLs of BDDs are independent of the variable ordering. On the other hand, for many standard benchmark functions, the LPLs of BDDs can be reduced by checking different variable orderings. 2) For many logic functions, the difference between the smallest LPLs of BDDs and the LPLs of BDDs with the fewest nodes is small. Thus, for many logic functions, conventional algorithm for BDD node minimization minimizes the LPL, as well. In applications that use BDDs (e.g. PTL synthesis), we can use node minimization algorithm for LPL minimization. 3) The exact smallest LPLs of heterogeneous MDDs is much smaller than the exact smallest LPLs of BDDs. However, for logic functions with many binary variables, exact minimization is difficult. 4) For 21 benchmark functions, both memory sizes and LPLs of the heterogeneous MDDs can be reduced to 86% and 72% of corresponding BDDs, respectively. When the memory limitations are set to the memory sizes of BDDs, the LPLs of heterogeneous MDDs can be reduced to 54% of BDDs without increasing memory sizes. In applications that can use MDDs (e.g. software synthesis), we can reduce the LPLs substantially by using heterogeneous MDDs.

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] 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.

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

[5] P. Buch, A. Narayan, A. R. Newton, and A. Sangiovanni-Vincentelli, “Logic synthesis for large pass transistor circuits,” *International Conference on Computer-Aided Design*, pp. 663-670, Nov. 1997.

[6] 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.

[7] 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.

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

[9] M. Lindgren, H. Hansson, and H. Thane, “Using measurements to derive the worst-case execution time,” *7th International Conference on Real-Time Systems and Applications (RTCSA’00)*, pp. 15-22, Cheju Island, South Korea, Dec. 12-14, 2000.

[10] T. H. Liu, M. K. Ganai, A. Aziz, and J. L. Burns, “Performance driven synthesis for pass-transistor logic,“ *International Workshop on Logic Synthesis*, pp. 255–259, 1998.

[11] 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.

[12] 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.

[13] 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.

[14] S. Minato, “Minimum-width method of variable ordering for binary decision diagrams,” *IEICE Trans. on fundamentals*, Vol. E75-A, No. 3, pp. 392–399, Mar. 1992.

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

[16] 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, Hiroshima, Japan, April 3-4, 2003.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] S. Nagayama and T. Sasao, “On the minimization of average path lengths for heterogeneous MDDs,” *34rd International Symposium on Multiple-Valued Logic*, Toronto, Canada, May 19-22, 2004.

[22] T. Nolte, H. Hansson, and C. Norströ̈m, “Probabilistic worst-case response-time analysis for the controller area network,” *the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’03)*, pp. 200-207, Toronto, Canada, May, 2003.

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

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

[25] C. Scholl and B. Becker, “On the generation of multiplexer circuits for pass transistor logic,” *Design Automation and Test in Europe*, pp. 372-378, 2000.

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

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

[28] M. Zu and A. M. K. Cheng, “Real-time scheduling of hierarchical reward-based tasks,” *the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS’03)*, pp. 2-9, Toronto, Canada, May, 2003.