社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ヘテロジニアス MDD における平均パス長の最小化法について

永山　忍[†]　笹尾　勤[††,†††]

† 九州工業大学大学院 情報システム専攻　〒 820–8502 福岡県飯塚市川津 680–4
†† 九州工業大学 電子情報工学科　〒 820–8502 福岡県飯塚市川津 680–4
††† 九州工業大学 マイクロ化総合技術センタ　〒 820–8502 福岡県飯塚市川津 680–4
E-mail: †nagayama@aries02.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp

あらまし　本論文では, ヘテロジニアス MDD (Multi-valued Decision Diagram) の平均パス長最小化アルゴリズムおよび平均パス長縮小のための発見的アルゴリズムを提案する. ヘテロジニアス MDD では多値変数はそれぞれ異なる変域をもってよく, 二値変数の分割を考慮することによって, 論理関数を短い平均パス長でコンパクトに表現できる. ベンチマーク関数を用いた実験により, ヘテロジニアス MDD ではメモリ量を増やすことなく, ROBDD (Reduced Ordered Binary Decision Diagram) に比べ, 平均パス長を約半分に縮小できることを示す.

キーワード　ヘテロジニアス MDD, 二値変数の分割, ROBDD, メモリ量, 平均パス長 (APL).

Minimization of Average Path Lengths for Heterogeneous MDDs

Shinobu NAGAYAMA[†] and Tsutomu SASAO[††,†††]

† Graduate school of Computer Science and Information System, Kyushu Institute of Technology
Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
†† Department of Computer Science and Electronics, Kyushu Institute of Technology
Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
††† Center for Microelectronic Systems, Kyushu Institute of Technology
Kawazu 680–4, Iizuka, Fukuoka, 820–8502 Japan
E-mail: †nagayama@aries02.cse.kyutech.ac.jp, ††sasao@cse.kyutech.ac.jp

Abstract　In this paper, we propose an exact and a heuristic minimization algorithms of average path lengths (APLs) for heterogeneous multi-valued decision diagrams (MDDs). In a heterogeneous MDD, each multi-valued variable can take different domain. 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 usign 21 benchmark functions show that the APLs of the heterogeneous MDDs can be reduced into a half of corresponding ROBDDs, on average, without increasing the memory size.

Key words　heterogeneous MDD, partition of binary variables, ROBDD, memory size, average path length (APL).

1. はじめに

BDD (Binary Decision Diagram) [5] や MDD (Multi-valued Decision Diagram) [12] は, 論理合成 [9], 設計検証 [1], [14], ソフトウェア合成 [2], [11], [19] 等で広く利用されている. 論理関数の評価に BDD や MDD を利用しているアプリケーションでは, その実行時間が BDD や MDD の平均パス長 (APL: Average Path Length) に依存ため, BDD や MDD の APL の短縮は, アプリケーションの実行時間の短縮に繋がる. 特に, 同じ論理関数を幾度も評価する論理シミュレーション [1], [14] では, APL の短縮がシミュレーション時間の直接的な短縮に繋がる.

BDD の APL の縮小法は, 論文 [13], [21] で提案されている. しかしながら, BDD では, APL を縮小化する変数順序と節点数を最小化する変数順序が必ずしも一致しないため, APL 縮小化により節点数が増加することもある. 論文 [21] の実験では, ベンチマーク関数 $C880$ の APL を縮小すると, 節点数は 10 倍になった. 一方, 論文 [19] で提案したヘテロジニアス MDD では, メモリサイズを増加することなく APL を縮小できる. 論文 [19] では, 二値変数の順序を固定し, 二値変数の分割だけを考慮したアルゴリズムを提案した. 本論文では, 二値変数の順序と二値変数

の分割の両方を考慮した APL 縮小化アルゴリズムを提案し, ヘテロジニアス MDD では BDD よりメモリ量を削減でき, 同時に APL を短かくできることを示す.

本論文では, 第 2 章でヘテロジニアス MDD, 平均パス長およびその他の定義を行なう. 第 3 章では, ヘテロジニアス MDD の APL 最小化アルゴリズムおよび APL 縮小のための発見的アルゴリズムを提案する. 第 4 章では, ベンチマーク関数を用いた実験により, ヘテロジニアス MDD の有用性を示す.

2. 諸 定 義

本章では, ヘテロジニアス MDD (Multi-valued Decision Diagram) を定義し, 多出力関数の表現法を示す.

2.1 論理変数の分割

［定義 2.1］ 論理変数を $X=(x_1,x_2,x_3,\ldots,x_n)$ とする. X の変数の集合を $\{X\}$ で表す. $\{X\}=\{X_1\}\cup\{X_2\}\cup\ldots\cup\{X_u\}$ かつ $\{X_i\}\cap\{X_j\}=\phi\ (i\neq j)$ のとき, $(X_1,X_2,\ldots,X_u)$ を X **の分割**という. また X_i のことを**超変数**と呼ぶ. $|X_i|=k_i\ (i=1,2,\ldots,u)$ とすると, $k_1+k_2+\ldots+k_u=n$ である. このとき論理関数 $f(X)$ は $f(X_1,\ldots,X_u):P_1\times P_2\times P_3\times\ldots\times P_u\rightarrow B$ と表現できる. ここで $P_i=\{0,1,2,\ldots,2^{k_i}-1\}, B=\{0,1\}$ である.

［定義 2.2］ 次のような X の分割 $(X_1,X_2,\ldots,X_u)$ を $X=(x_1,x_2,\ldots,x_n)$ の**変数順序を固定した分割**という.

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

ただし, $|X_i|=k_i$ である.

変数順序を固定しない分割も同様に定義できる. また本論文で対象とする論理関数は, 冗長な変数を含まないものとする. 即ち, 冗長な変数があれば, 予め除去しておくものとする.

2.2 ヘテロジニアス MDD

BDD (Binary Decision Diagram), ROBDD (Reduced Ordered Binary Decision Diagram) の定義は [5] を, MDD (Multi-valued Decision Diagram), ROMDD (Reduced Ordered Multi-valued Decision Diagram) の定義は [12] を参照されたい.

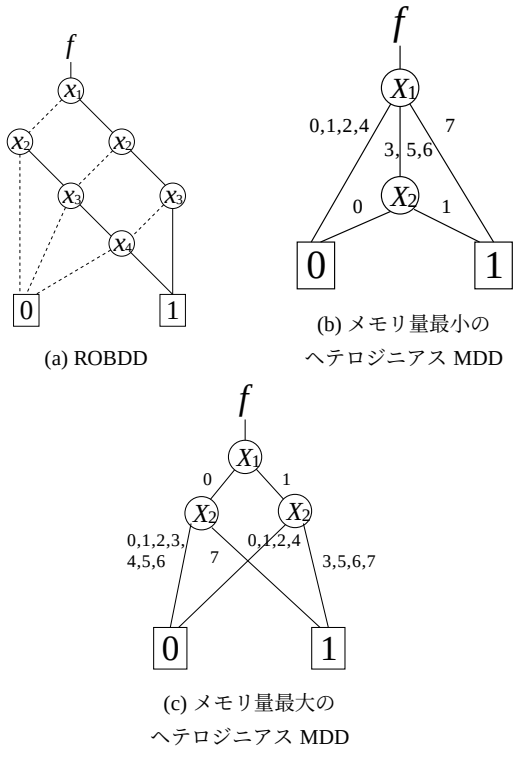
［定義 2.3］ 入力変数 $X=(x_1,x_2,\ldots,x_n)$ を $(X_1,\ldots,X_u)$ と分割したとき, 論理関数 $f(X)$ を表現する ROMDD を**ヘテロジニアス MDD** という. ヘテロジニアス MDD は写像 $f:P_1\times P_2\times\ldots\times P_u\rightarrow B$, ここで $P_i=\{0,1,\ldots,2^{k_i}-1\}, B=\{0,1\}$ を表現する. ヘテロジニアス MDD では, 変数 X_i を表現する非終端節点は 2^{k_i} 個の枝を持つ. ここで $|X_i|=k_i$ である.

［定義 2.4］ DD (Decision Diagram) において, 非終端節点の総数を **DD の節点数**といい, $nodes(\mathrm{DD})$ と表記する.

［定義 2.5］ DD において, 変数 X_i をインデックスとする節点数を **DD の** X_i **における幅**と定義し, $width(\mathrm{DD},i)$ で表す.

u 個の超変数からなるヘテロジニアス MDD の節点数は,

$$nodes(\text{ヘテロジニアス MDD})=\sum_{i=1}^{u}width(\text{ヘテロジニアス MDD},i)$$



(a) ROBDD

(b) メモリ量最小のヘテロジニアス MDD

(c) メモリ量最大のヘテロジニアス MDD

図 2.1 BDD とヘテロジニアス MDD

で計算できる.

［例 2.1］ 論理関数 $f=x_1x_2x_3\vee x_2x_3x_4\vee x_3x_4x_1\vee x_4x_1x_2$ を表現する ROBDD を図 2.1(a) に, ヘテロジニアス MDD を図 2.1(b), (c) に示す. 図 2.1(a) の ROBDD で実線は 1-枝, 破線は 0-枝を示す. また, 図 2.1(b) のヘテロジニアス MDD では $X_1=(x_1,x_2,x_3), X_2=(x_4)$, そして図 2.1(c) のヘテロジニアス MDD では $X_1=(x_1), X_2=(x_2,x_3,x_4)$ である. (例終り)

2.3 多出力関数の表現

多出力関数を, $F=(f_0,f_1,\ldots,f_{m-1}):B^n\rightarrow B^m, B=\{0,1\}$ と定義する. ここで, n は論理回路の入力数, m は出力数を表す. 本論文では, SBDD (Shared Binary Decision Diagram) [24] を用いて多出力関数を表現する. 以下 BDD および MDD は, 特に指定がない限り SBDD, SMDD を示す.

2.4 メ モ リ 量

［定義 2.6］ DD をメモリに格納するために必要なワード数を **DD のメモリ量**と定義する.

RODD (Reduced Ordered Decision Diagram) の各節点を表現するには, インデックスと各枝のアドレスが必要である. BDD の各節点は 2 本の枝をもつため, BDD の全節点を表現するためのメモリ量は,

$$(2+1)\times nodes(\mathrm{BDD})$$

である. ヘテロジニアス MDD では変数毎に枝数が異なってもよい. ヘテロジニアス MDD の全節点を表現するためのメモリ量は

$$\sum_{i=1}^{u}(2^{k_i}+1)\times width(\text{ヘテロジニアス MDD},i)$$

である.

［例 2.2］ 図 2.1(a) の BDD のメモリ量は 18, 図 2.1(b) のヘテロジニアス MDD のメモリ量は 12, そして図 2.1(c) のヘテロジニアス MDD のメモリ量は 21 である. (例終り)

［定義 2.7］ 関数 f が与えられたとき, 変数順序を固定した分割により得られるメモリ量が最小のヘテロジニアス MDD を関数 f の**最小ヘテロジニアス MDD** と呼ぶ.

［定理 2.1］ [20] 真に n ($n \geq 2$) 変数に依存する論理関数を $f(X)$ とする. $X = (x_1, x_2, \ldots, x_n)$ の変数順序を固定したとき, f を表現するヘテロジニアス MDD のメモリ量を Mem(ヘテロジニアス MDD : f) とすると,

$$Mem(\text{ヘテロジニアス MDD} : f) \geq nodes(\text{BDD}) + 2$$

が成立する. ただし, BDD は f を表現し, 変数順序は $(x_1, x_2, \ldots, x_n)$ である.

定理 2.1 は, 任意の論理関数で成立するため, ヘテロジニアス MDD の APL 最小化アルゴリズムで計算時間を削減できる.

［性質 2.1］ [22] 真に n 変数に依存する論理関数を $f(X)$ とする. $X = (x_1, x_2, \ldots, x_n)$ の変数順序を固定したとき, f を表現する最小ヘテロジニアス MDD のメモリ量を Mem_{min}(ヘテロジニアス $MDD : f$) とする. f の関数分解 $f = g(h(X_1), X_2)$ において, h および g を表現する最小ヘテロジニアス MDD のメモリ量をそれぞれ Mem_{min}(ヘテロジニアス $MDD : h$) および Mem_{min}(ヘテロジニアス $MDD : g$) とすると, 多くのベンチマーク関数において次の 2 つの関係が成立する.

（1） Mem_{min}(ヘテロジニアス $MDD : f$) $> Mem_{min}$(ヘテロジニアス $MDD : h$)

（2） Mem_{min}(ヘテロジニアス $MDD : f$) $> Mem_{min}$(ヘテロジニアス $MDD : g$)

性質 2.1 は, 任意の論理関数で常に成立するとは限らないため, 性質 2.1 を用いたアルゴリズムは, 最適解を保証しない. しかしながら, 性質 2.1 を用いることによって, 計算時間を大幅に削減できる.

2.5 平均パス長 (APL: Average Path Length)

［定義 2.8］ DD において, 根から終端節点までの経路を DD のパスという. このときパス上の非終端節点数をパス長という.

［定義 2.9］ DD において, パスが非終端節点 v_i を通過する確率を節点通過確率とよび, $NTP(DD, i)$ で表す.

本論文では, 以下のような計算モデルを仮定する.

（1） 各変数の値に従い, DD 上の節点を辿ることで論理関数の評価を行う.

（2） MDD は BDD パッケージを使わずに, 直接実現する [12].

（3） MDD の各変数に対して, 符号化された入力値が入力される. そのため入力値の変換時間は無視できる. 例えば, $X_1 = (x_1, x_2, x_3, x_4) = (1, 0, 0, 1)$ のときは, $X_1 = 9$ が入力値として与えられる.

（4） MDD を用いた論理関数の評価では, 節点のアクセスに主な時間を費やす.

（5） 節点の評価時間は全て等しい.

このとき, DD で表現した論理関数の評価時間は, アクセスした非終端節点の個数 (パス長) に比例する. また, 入力変数 x_i に 0 と 1 が等確率で入力されると仮定する. このような仮定の下で, DD の評価時間を見積もるために平均パス長を使う.

本論文では, 多出力関数の表現法に SDD (Shared Decision Diagram) を用いる. 多出力関数 $F = (f_0, f_1, \ldots, f_{m-1})$ を表現する SDD の平均パス長は, 各単一出力関数 f_i を表現する DD の平均パス長の合計で求められる [26].

［定理 2.2］ [26] DD の平均パス長は, 節点通過確率の合計に等しい.

3. APL 最小化

ヘテロジニアス MDD の APL は, 入力変数 X の分割の仕方と変数順序に依存する.

［例 3.1］ 図 2.1(a) の BDD の APL は, 3.125, 図 2.1(b) のヘテロジニアス MDD の APL は, 1.375, そして, 図 2.1(c) のヘテロジニアス MDD の APL は, 2.0 である. (例終り)

APL を最小にする X の分割は, $X = X_1$, $k_1 = n$ (1 つの超変数からなる分割) である. しかしこのときのヘテロジニアス MDD のメモリ量は真理値表のメモリ量 2^n にほぼ等しくなり実用的でない. そのため与えられたメモリ容量で APL が最小となる X の分割と変数順序を見つける必要がある. 入力変数の順序を考慮したヘテロジニアス MDD の APL 最小化問題を次のように定式化する.

［問題 3.1］ 論理関数 f とメモリ制限 L が与えられたとき, メモリ量が L 以下で, かつ APL 最小のヘテロジニアス MDD を構成する入力変数 X の分割と変数順序を求める.

3.1 APL 最小化アルゴリズム

問題 3.1 を解くための擬似コードをアルゴリズム 3.1 に示す. アルゴリズム 3.1 は, 論理関数の表現に BDD を使う. 初期 BDD の変数順序は $X = (x_1, x_2, \ldots, x_n)$ とする. アルゴリズム 3.1 の 2 行目, 8 行目の **minimize_APL** は, 変数順序を固定した場合の最適な分割を見つける [19]. この関数には, 引数として BDD の根節点における変数のインデックスとメモリ容量 L を与える必要がある. 6 行目と 8 行目の t は変数順序変更後の BDD の根節点における変数のインデックスを表す. 5 行目は定理 2.1 を使い計算時間を削減している. 6 行目の **minimize_memory** は, 変数順序を固定したメモリ量最小の分割を見つける [20]. このアルゴリズムは網羅的に解を探索し, 最適解を見つける.

［アルゴリズム 3.1］ (変数順序を考慮したヘテロジニアス MDD の APL 最小化)

```
 1: exhaustive_search (BDD, メモリ制限 L)
 2:     最小 APL = minimize_APL (1, L) ;
 3:     for (変数の全ての順列)
 4:         BDD の変数順序を変更 ;
 5:         if (L < nodes(BDD) + 2) continue ;
 6:         memory = minimize_memory (t) ;
 7:         if (L < memory) continue ;
 8:         APL = minimize_APL (t, L) ;
 9:         if (APL < 最小 APL)
10:             最小 APL = APL ;
11:             変数順序の記憶 ;
12:             分割の記憶 ;
```

［アルゴリズム 3.2］ (変数順序を考慮したヘテロジニアス MDD の APL 発見的縮小化)

```
 1:  sifting_APL (BDD, メモリ制限 L, sifting 回数 s)
 2:    コスト = minimize_APL (1, L) ;
 3:    for (r = 0; r < s; r++)
 4:      for (全ての x_i ∈ X)
 5:        for (全ての位置)
 6:          x_i を移動 ;
 7:          memory = minimize_memory (t) ;
 8:          L_mem を計算 ;
 9:          if (L ≦ L_mem) break ;
10:          if (L < memory) continue ;
11:          APL = minimize_APL (t, L) ;
12:          if (APL < コスト)
13:            コスト = APL ;
14:            変数順序の記憶 ;
15:            分割の記憶 ;
```

表 4.1　n 変数論理関数における BDD とヘテロジニアス MDD のメモリ量と APL

	メモリ量				APL			
	BDD		MDD		BDD		MDD	
n	Pivot	Ord.	Part.	O & P	Pivot	Ord.	Part.	O & P
4	1.00	1.07	0.87	0.86	1.00	0.99	0.39	0.37
5	1.00	1.07	0.91	0.91	1.00	0.98	0.28	0.27
6	1.00	1.08	0.80	0.80	1.00	0.97	0.35	0.32
7	1.00	1.08	0.78	0.79	1.00	0.97	0.28	0.27
8	1.00	1.08	0.79	0.81	1.00	0.97	0.23	0.22
9	1.00	1.07	0.81	0.83	1.00	0.98	0.20	0.19
10	1.00	1.06	0.83	0.84	1.00	0.98	0.17	0.17

3.2　APL 縮小のための発見的アルゴリズム

二値変数の変数順序と分割の両方を変化させ, ヘテロジニアス MDD を網羅的に調べる場合, 考慮すべきヘテロジニアス MDD は, 数多く存在する [22]. そのため, 最適なヘテロジニアス MDD を実用時間内に発見するのは困難である.

本節では, sifting [23] と分枝限定法 [19] を用いた APL 縮小のための発見的アルゴリズムを提案する. sifting は, BDD の変数順序最適化アルゴリズムであり, (1) 変数順序の変更, (2) コストの計算という 2 つのステップから構成される. BDD の最適化には多くの場合, コストとして BDD の節点数が使われる. 本論文では, コストとしてヘテロジニアス MDD の APL を用いる. 変数順序変更後の APL の計算法は, 論文 [21] で提案した手法を用いている. アルゴリズム 3.2 も, アルゴリズム 3.1 同様, 論理関数の表現に BDD を使う. 初期 BDD の変数順序は $X = (x_1, x_2, \ldots, x_n)$ とする. アルゴリズム 3.2 の 9 行目は性質 2.1 を使っている. 8 行目の L_{mem} は, x_i が BDD の終端節点へ (根節点へ) 移動するときの x_i より根節点側 (終端節点側) の部分グラフにおける, メモリ量最小のヘテロジニアス MDD のメモリ量を表す. これは, 論文 [8] で提案された下界の応用である.

4. 実験結果

以下の計算機環境で実験を行なった. CPU: Pentium4 Xeon 2.8GHz, 1 次キャッシュ: 32KB, 2 次キャッシュ: 512KB, 主メモリ: 4GB, OS: redhat (Linux 7.3), C-コンパイラ: gcc -O2.

4.1　n 変数論理関数におけるメモリ量と APL の比較

表 4.1 は, n 変数論理関数を表現する BDD とヘテロジニアス MDD を, そのメモリ量および APL の比率の平均によって比較している. 表中の “Pivot” は, 節点数最小の BDD を表す. 実験は, この BDD のメモリ量および APL を 1.00 として他の DD の比率を算出した. “Ord.” は, 変数順序最適化によって得られる APL 最小の BDD を表す. 表中の “MDD” はヘテロジニアス MDD を表し, “Part.” は, 論文 [19] で提案したアルゴリズム (以下, **分割アルゴリズム**) により得たヘテロジニアス MDD である. 分割アルゴリズムは, 変数順序を固定し, 最適な変数の分割を得る. “O & P” は, アルゴリズム 3.1 により得た, メモリ制限下の APL 最小のヘテロジニアス MDD を表す. アルゴリズム 3.1 は, 変数順序と変数の分割の両方を考慮し, APL 最小のヘテロジニアス MDD を得る. 分割アルゴリズムおよびアルゴリズム 3.1 は, 節点数最小の BDD のメモリ量をメモリ制限 L とした. また分割アルゴリズムの変数順序は節点数最小の BDD の変数順序で固定した. 本表の DD は否定枝を用いていない.

4 変数論理関数および 5 変数論理関数では, 全ての関数を NPN 同値類 [17], [25] に分類し, 各 NPN 同値類から NPN 代表関数を生成した. そして, 全関数における平均を算出した. $6 \leqq n \leqq 10$ の各 n 変数論理関数では, 異なる真理値表濃度 [25] で 1,000 個の論理関数をランダムに生成し, 平均を算出した.

BDD では, 変数順序を最適にしても APL の大幅な縮小は望めない. 一方, ヘテロジニアス MDD では, 短い APL でなおかつ, コンパクトに論理関数を表現できる. 表 4.1 では, n (二値変数の数) の増加にともなって, ヘテロジニアス MDD の APL の相対値が小さくなっている. また, ヘテロジニアス MDD では, 変数の分割だけを考慮しても APL を大幅に縮小できるが, 変数順序の変更を考慮することで, さらに APL を縮小できる. 変数の順序と分割の両方を考慮したアルゴリズム 3.1 は, 実用時間内に, 11 入力程度の論理関数まで最小解を求めることができた.

4.2　ベンチマーク関数におけるメモリ量と APL の比較

表 4.2 は, 入力数の大きいベンチマーク関数に対して, BDD とヘテロジニアス MDD のメモリ量および APL を比較している. 表中の “Init.” は, この実験に用いた初期 BDD を表し, その変数順序は [28] から得た. “Ord.” は, 論文 [21] の APL 縮小化アルゴリズム (以下, sifting アルゴリズム) により得た BDD である. sifting アルゴリズムは変数順序だけを考慮して, APL 縮小化を行なった. 表中の “MDD” はヘテロジニアス MDD を意味し, “Part.” は, 分割アルゴリズムにより得たヘテロジニアス MDD である. 分割アルゴリズムは, 変数の分割だけを考慮して APL を縮小した. “O & P” は, アルゴリズム 3.2 により得たヘテロジニアス MDD である. アルゴリズム 3.2 は, 変数順序と変数の分割の両方を考慮して APL を縮小した. 分割アルゴリズムおよびアルゴリズム 3.2 は, 初期 BDD のメモリ量をメモリ制限 L として, 最適化を行なった. また, sifting アルゴリズムおよびアルゴリズム 3.2 の sifting 回数は, 2 とした. 本表での BDD およびヘテロジニアス MDD は否定枝を用いている. 分割アルゴリズム以外のアルゴリズムは発見的手法であるため, 厳密な最小解を得る保証がない. “比率の平均” は, 初期 BDD のメモリ量および

表 4.2　ベンチマーク関数における BDD とヘテロジニアス MDD のメモリ量と APL

			メモリ量				APL			
			BDD		MDD		BDD		MDD	
関数名	入力数	出力数	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
比率の平均			1.00	2.03	1.00	1.00	1.00	0.88	0.62	0.51

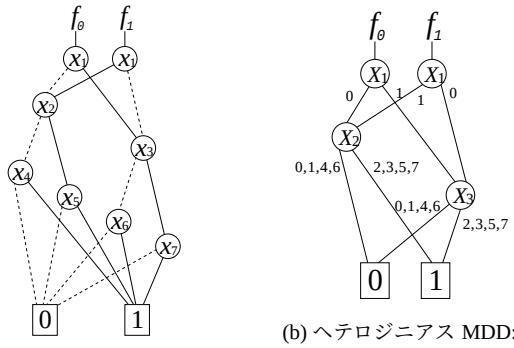
APL を 1.00 としたときの各 DD のメモリ量および APL の比率の算術平均を表す.

BDD の場合, sifting アルゴリズムを用いると, APL を平均して 88% に縮小できる. しかしながら, メモリ量は 2 倍に増加する. 特に, *C880*, *C1908*, *cps*, *frg2*, *i8*, *i10*, *too_large* ではメモリ量が大幅に増加している. sifting アルゴリズムは, メモリ量を意識せずに APL のみを縮小するため, メモリ量が大幅に増加することもある. 一方, ヘテロジニアス MDD の場合, 分割アルゴリズムおよびアルゴリズム 3.2 はメモリ制限 (初期 BDD のメモリ量) を超えずに APL を縮小できる. この実験で, 分割アルゴリズムおよびアルゴリズム 3.2 は, それぞれ平均して 62%, 51% に APL を縮小できた. 多くの関数は, 変数の分割を考慮するだけで APL を大幅に縮小できる (例えば *C499*). しかしながら, *apex6* や *i8* のように変数の分割だけでは, APL があまり減らない関数も存在する. アルゴリズム 3.2 は, 変数順序と変数の分割の両方を考慮しているため, 変数の分割だけでは APL があまり減らない関数でも, メモリ量を増加させることなく APL を大幅に縮小できた.

［例 4.1］ 7 変数 2 出力の BTREE 関数 [7] を考える. 図 4.1(a) は, BTREE 関数を表現する節点数最小の BDD を示している. この BDD は, 変数順序を変更しても, APL を縮小できない. また変数の分割だけを考慮してもメモリ量を増加せずに APL を縮小することはできない. しかしながら, 変数順序と変数の分割の両方を考慮することによって APL を縮小でき, 図 4.1(b) のヘテロジニアス MDD を得る. ここで $X_1 = (x_1)$, $X_2 = (x_2, x_4, x_5)$, $X_3 = (x_3, x_6, x_7)$ である. (例終り)

4.3 計算時間の比較

表 4.3 は, sifting アルゴリズム, 分割アルゴリズム, アルゴリズム 3.2 に必要な計算時間を比較している. 表 4.3 は, 表 4.2 の BDD およびヘテロジニアス MDD を得るための計算に必要な CPU 時間を表している.



(a) BDD: $APL = 6.0$　　(b) ヘテロジニアス MDD: $APL = 4.0$

図 4.1　BTREE 関数の BDD とヘテロジニアス MDD

sifting アルゴリズムと分割アルゴリズムは, アルゴリズム 3.2 より計算時間が短い. 特に分割アルゴリズムは, 多くのベンチマーク関数において, 短時間で APL を大幅に縮小できるため, 効率的なアルゴリズムといえる. しかし例 4.1 のように, 変数の分割だけでは APL を縮小できない場合があるため, 変数順序と変数の分割の両方を考慮することも重要である.

5. 結論とコメント

本論文では, ヘテロジニアス MDD の平均パス長最小化アルゴリズムと平均パス長縮小のための発見的手法を提案した. 提案した平均パス長最小化アルゴリズムと平均パス長縮小のための発見的手法をプログラム実装し, ベンチマーク関数を用いた実験により以下のことを示した. 1) ヘテロジニアス MDD は, メ

表 4.3　APL 最小化のための計算時間 [sec] の比較

関数名	Ord.	Part.	O & P
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
平均	9.37	0.26	6732.75

モリ量を増加させることなく平均パス長を BDD の半分近くまで縮小できる. 2) アルゴリズム 3.1 は, 変数順序と変数の分割の両方を考慮することによって, 実用時間内に, 11 入力程度の論理関数まで最小解を見つけることができる. 3) 変数の分割だけを考慮するアルゴリズムは高速に平均パス長を縮小できる. しかしながら, 分割だけでは平均パス長があまり小さくならない場合もあるため, 変数順序と変数の分割の両方を考慮することも重要である.

謝　　辞

本研究は一部, 日本学術振興会: 科学研究費および文部科学省: 北九州地域知的クラスター創成事業補助金による.

文　　献

[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] J. T. Butler, T. Sasao, and M. Matsuura, “Average path length of binary decision diagrams,” (draft).

[8] 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.

[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,” *Multiple-Valued Logic*, 1988, 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, “Minimization of memory size for heterogeneous MDDs,” *Asia and South Pacific Design Automation Conference (ASP-DAC’2004)*, Yokohama, Japan, Jan. 2004 (accepted for publication).

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

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

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

[26] 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.

[27] 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.

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

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