BDD を用いた多出力論理関数の評価法に関する検討

松浦宗寛 [1], 井口幸洋 [2], 笹尾勤 [1,3]

[1] 九州工業大学 情報工学部
[2] 明治大学理工学部情報工学科
[3] 九州工業大学 マイクロ化総合技術センター

あらまし: データ構造に BDD for ECFN(encoded characteristic function for non-zero outputs) を用いた多出力論理関数の評価法を提案する. ECFN を用いると MTBDD, BDD for CF(characteristic function), および SBDD よりもコンパクトにデータを実現でき, さらに, SBDD よりも 2 倍以上高速に評価できることを実験的に示す. また, MTBDD に比べると評価時間は 5.7 倍となるが, 0.3% の節点数で, また, BDD for CF に比べると評価時間は 1.14 倍となるが, 0.5% の節点数のメモリで評価できることも示す.

和文キーワード: 多出力論理関数, 二分決定グラフ (BDD), MTBDD, BDD for CF, BDD for ECFN

A Fast Method to Evaluate Multiple-Output Logic Functions using BDDs

Munehiro MATSUURA[1], Yukihiro IGUCHI[2], Tsutomu SASAO[1,3]

[1]Department of Computer Science and Electronics, Kyushu Institute of Technology
[2]Department of Computer Science, Meiji University
[3]Center for Microelectronic Systems, Kyushu Institute of Technology

Abstract: We propose a fast method to evaluate multiple-output logic functions using BDDs for ECFNs (Encoded Characteristic Function for Non-zero outputs). Evaluation using BDDs for ECFN outperforms ones using MTBDDs, BDDs for CF, and SBDDs with respect to the required size of memory and evaluation speed. The number of nodes of BDDs for ECFNs are smaller than ones of corresponding MTBDDs (Multi-Terminal BDDs), BDDs for CFs (Characteristic Functions), and SBDDs (Shared BDDs).

Experimental results using standard benchmark functions show that

1) evaluation time using BDDs for ECFNs is less than a half of corresponding SBDDs,
2) evaluation time using BDDs for ECFNs is 5.7 times of corresponding MTBDDs, however the number of nodes of BDDs for ECFNs is only 0.3% of MTBDDs,
3) evaluation time using BDDs for ECFNs is 1.14 times of corresponding BDDs for CFs, however the number of nodes of BDDs for ECFNs is only 0.5% of BDDs for ECFNs.

Key words: Multiple-output functions, Binary decision diagram, MTBDD, BDD for CF, BDD for ECFN

1 はじめに

多出力論理関数の表現には，MTBDD(multi-terminal BDD), SBDD(shared BDD), 特性関数 (characteristic function) を表す BDD(これを BDD for CF と書く) などを用いる方法が提案されており，それぞれ利害得失がある [1, 3]. BDD for CF をデータ構造として利用する場合, 評価時間は $O(n+m)$ となる．ここで, n は入力変数の個数, m は出力変数の個数である．また, MTBDD では, 評価時間は $O(n+m)$ に, SBDD では $O(n \cdot m)$ となる．高速化が目標の場合, BDD for CF や MTBDD が有利だが, 多くの場合, 表現する BDD が大きくなりすぎる．その時は, SBDD を使うことになるが, 評価時間が長くかかる.

我々は, SBDD よりコンパクトなデータ構造として BDD for ECFN(encoded characteristic function for non-zero outputs) を提案した [6]．本稿では，このデータ構造を用いて論理関数の評価を行うと, SBDD を用いた場合に比べ高速にできることを示す. 実験より SBDD に比べ評価時間は 41% で，また，節点数は 81% で実現できることを示す．また，MTBDD に比べ 5.7 倍の評価時間はかかるが 0.3% の節点数で, BDD for CF の 1.14 倍の評価時間はかかるが 0.5% の節点数で実現できることも示す.

2 BDD を用いた論理関数の評価法

論理関数の表現方法としてブランチング・プログラムが知られている [1, 3]．図 1にブランチング・プログラムの生成法を示す．与えられた論理関数を図 1(a) に示すような BDD(binary decision diagram) で表現し，各非終端節点を *If then else* のコードに置換すると, 図 1(b) のブランチング・プログラムが生成できる．これをコンパイルし，計算機で実行すれば論理関数を評価できる．この時，ある入力に対する論理関数値をブランチング・プログラムで評価する時間は，実行される *If then else* 文の個数に比例する．*If then else* 文の個数は, BDD を根から辿ったパス長 (経路長) に等しい.

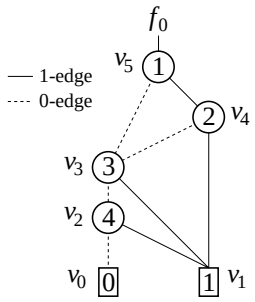
例 2.1 図 *1*の *BDD* が表す論理関数 f_0 に対し，$(x_1, x_2, x_3, x_4) = (1, 0, 1, 1)$ を加えると図 *2*に示すように *3* 本の枝を経由して終端節点に到達し $f_0(1, 0, 1, 1) = 1$ と評価できる．図からこの関数では，最小パス長は *2* であり，最大パス長は *4* であることがわかる．(例終り)

このように, 評価時間は入力値に依存する．すべての入力値を加えたときの平均評価時間は平均パス長を求めればよい.

定義 2.1 根から終端節点まで入力変数の値に従って *BDD* を辿るとき, 節点 v_i を通過する確率を**節点通過確率**と呼び $P(v_i)$ で記す．また，v_i から出る *0* 枝 *(1* 枝*)* に進む確率を**枝通過確率**と呼び $P(e_{i_0})(P(e_{i_1}))$ で記す.

各入力変数が 0 または 1 をとる確率は等しく 1/2 であると仮定すると, $P(e_{i_0}) = P(e_{i_1}) = P(v_i)/2$ である.

定理 2.1 節点通過確率は，そこに入ってくる枝通過確率の合計に等しい.



(a) BDD の例

v_5: *if*($x_1 == 0$) *goto* v_3;
else goto v_4;
v_4: *if*($x_2 == 0$) *goto* v_3;
else goto v_1;
v_3: *if*($x_3 == 0$) *goto* v_2;
else goto v_1;
v_2: *if*($x_4 == 0$) *goto* v_0;
else goto v_1;
v_1: *return(1);*
v_0: *return(0);*

(b) BDD より生成したブランチング・プログラム

図 1: ブランチング・プログラムの生成法

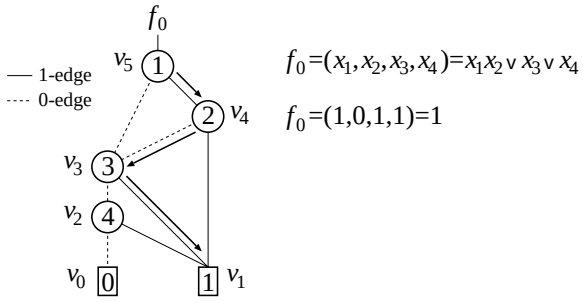


図 2: BDD を用いた論理関数の評価法

定理 2.2 平均パス長は，すべての枝通過確率の合計に等しい．平均パス長は，すべての節点通過確率の合計に等しい.

例 2.2 図 *3*の *BDD* の平均パス長を考える．$P(v_5) = 1$ であり，$P(e_{5_0}) = P(e_{5_1}) = 1/2$．$P(v_4) = 1/2$．$P(e_{4_0}) = P(e_{4_1}) = 1/4$．$P(v_3) = P(e_{5_0}) + P(e_{4_0}) = 1/2 + 1/4 = 3/4$．$P(e_{3_0}) = P(e_{3_1}) = 3/8$．$P(v_2) = 3/8$．$P(e_{2_0}) = P(e_{2_1}) = 3/16$．従って，*BDD* の平均パス長は，$P(v_5) + P(v_4) + P(v_3) + P(v_2) = 1 + 1/2 + 3/4 + 3/8 = 21/8 = 2.625$ となる．(例終り)

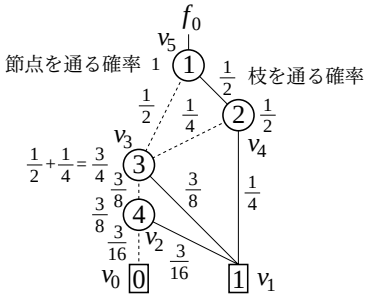


図 3: BDD の平均パス長

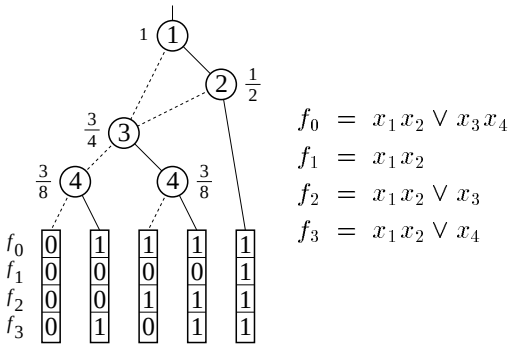


図 4: MTBDD の平均パス長

BDD の平均パス長を求めるアルゴリズムを次に示す.

アルゴリズム 2.1 *(BDD の平均パス長の計算)*

1) 根の節点通過確率を *1* とする.
2) 節点通過確率が決定した節点について, そこから出る各枝の枝通過確率を節点通過確率の $1/2$ とする.
3) 非終端節点について, そこに入ってくる枝通過確率がすべて決定すれば, その合計値を節点通過確率とする.
4) 上記 *1)–3)* をすべての非終端節点の節点通過確率が決まるまで行う.
5) すべての非終端節点の節点通過確率の合計を平均パス長とする.

3 多出力論理関数を表現する BDD

2節では一出力論理関数の場合について考えたが, 取り扱いたい論理関数は一般に多出力である. 多出力論理関数を扱う方法に MTBDD, SBDD, および BDD for CF を用いる方法がある. 本節では, これらについて例を用いて簡単に説明する.

3.1 MTBDD を用いた論理関数の評価法

図 4に 4 入力 4 出力の論理関数を MTBDD で表現した例を示す. 終端節点には, すべての出力の出力値を格納してあるので, MTBDD では, 根から各入力変数の値に従って枝を終端節点まで辿るだけですべての出力値を求められる. 例えば 1 ワード 32 ビットの整数型変数に出力値を蓄えるならば, 終端節点で 32 出力までは 1 回のメモリアクセスで, 64 出力までは 2 回のメモリアクセスで出力が求まる. 一般に, 評価時間は $O(n+m)$ である. 入出力の個数が 100 程度とし, 上のように出力をワード毎に格納する場合, ほぼ $O(n)$ と考えてよい.

BDD の場合と同様, アルゴリズム 2.1を用いて平均パス長を求めれば平均評価時間を計算できる.

MTBDD を用いると, すべての出力値が 1 回の探索で求まるので評価を高速にできる. しかし, MTBDD では終端節点数が最大で 2^m 個となるので, 節点数が膨大となり, 多くの実用的な論理関数ではメモリサイズの制限で利用不可能となる.

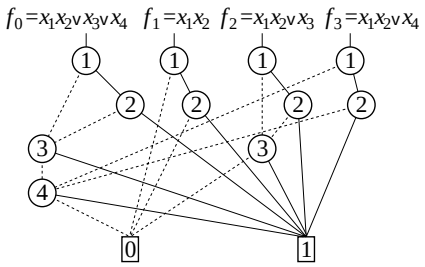


図 5: SBDD の例

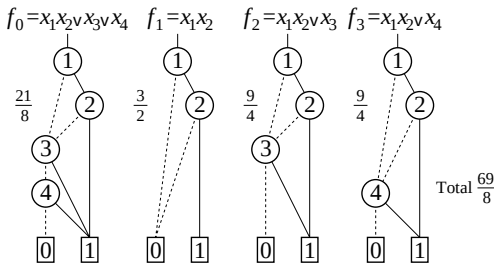


図 6: SBDD の平均パス長

例 3.1 図 *4の MTBDD* の平均パス長は, すべての非終端節点の節点通過確率の和 $1+\frac{1}{2}+\frac{3}{4}+\frac{3}{8}+\frac{3}{8}=3$ と計算できる. (例終り)

3.2 SBDD を用いた論理関数の評価法

SBDD は, 各出力ごとに BDD を作り, 共有可能な部分を共有し, 節点数削減を狙った BDD である. n 入力 m 出力の論理関数では, m 個の BDD を共有する. 評価は, 各出力毎に根から入力変数に従って枝を辿ればよい. 従って, 評価時間は $O(n \cdot m)$ となる.

図 4の論理関数を SBDD で実現した例を図 5に示す. すべての出力値を求めるには, f_0, f_1, f_2, f_3 を表す BDD それぞれに対して評価する必要がある. 従って, 平均パス長は図 5の SBDD を図 6の BDD として, 各 BDD の平均パス長を求め合計すればよい.

例 3.2 図 *5の SBDD* の平均パス長は $\frac{21}{8}+\frac{3}{2}+\frac{9}{4}+\frac{9}{4}=\frac{69}{8}=8.625$ となる. また, 補助変数を用いて出力部分も *BDD* で表現すると図 *7の SBDD* が得られる. この場合, 平均評価時間は $8.625+6=14.625$ となる. (例終り)

3.3 BDD for CF を用いた論理関数の評価法

Characteristic function(特性関数, 以後 CF と略記) を用いた, 高速な論理関数の評価法が提案されている [1]. n 入力 m 出力論理関数の CF は, $n+m$ 入力 1 出力論理関数となり, もとの論理関数の入力と出力の組合せが存在するとき特性関数 CF は 1 となり, それ以外の場合は 0 となる.

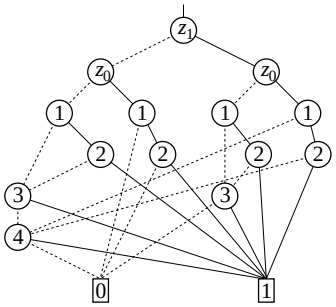


図 7: 補助変数を用いた多出力関数の表現

表 1: 真理値表から CF の真理値表を求めた例

(a) 真理値表

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) CF の真理値表

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

例 3.3 表 *1(a)* に二出力関数 $f_0 = x_1 x_2$, $f_1 = x_1 \vee x_2$ の真理値表を示す．また，*CF* の真理値表を *(b)* に示す．$(x_1, x_2, f_0, f_1) = (0,0,0,0)$ の組合せは，表 *1(a)* に存在するので，*(b)* の対応する行の出力 CF には *1* が記入される．次に $(x_1, x_2, f_0, f_1) = (0,0,0,1)$ の組合せは，表 *1(a)* に存在しないので，*(b)* の対応する行の出力 CF には *0* が記入される．(例終り)

n 入力 m 出力関数を表現する CF の真理値表は 2^{n+m} 行で，その内 2^n 行には 1 が，残りの行には 0 が記入されている．

CF を表現する BDD(これを以後 BDD for CF と呼ぶ) を使用して，高速に論理関数の評価を行うことができる． 図 4, 5で用いた論理関数の CF を求め，それを BDD で実現した BDD for CF の例を図 8に示す． f_0, f_1, f_2, f_3 のインデックスをもつ非終端節点は補助変数に対応する．BDD for CF では，補助変数と入力変数の非終端節点が混在している．ここで，補助変数はその出力に依存するすべての入力変数に比べ，葉の側にあることに注意．また，補助変数の 0 枝，または，1 枝のいずれか一方は必ず終端節点 0 に接続されていることにも注意されたい．BDD for CF 上で論理関数を評価する方法を例を用いて説明する．

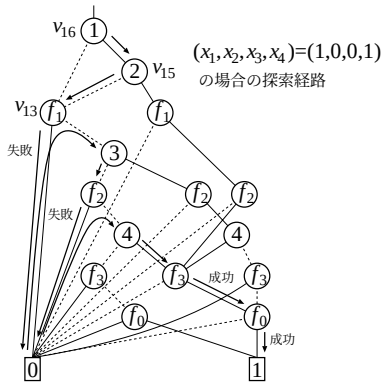


図 8: BDD for CF を用いた論理関数の評価

例 3.4 *(BDD for CF を使用した論理関数の評価法)*
図 *8の BDD for CF* を使用して $(x_1, x_2, x_3, x_4) = (1,0,0,1)$ に対する出力値を求める． 根である v_{16} のインデックスは x_1．x_1 の値は *1* であるので *1* 枝に進み，インデックスが x_2 である節点 v_{15} に到達する．x_2 は *0* であるので *0* 枝に進み，v_{13} に到る．v_{13} は補助変数に対応する．補助変数に到達したときは，*0* 枝か *1* 枝のどちらかに進む．もし *0* 枝を進んで終端節点 *0* に到達したら，その出力は *0* でない *(*つまりその出力は *1* である*)* ことを示しているので，バックトラックして *1* 枝を進むことにする．反対に，もし *1* 枝を進んで終端節点 *0* に到達したら，その出力は *1* でない *(*つまりその出力は *0* である*)* ことを示しているので，バックトラックして *0* 枝を進むことにする．

この例では，*1* 枝を進んでみて終端節点 *0* に到達したのでバックトラックして *0* 枝側に進み，$f_1 = 0$ が決定した． もし *0* 枝に進んでいたら，終端節点 *0* に到達しないので，これだけで $f_1 = 0$ が決定する． この場合は，バックトラック分の探索をしないで済むことに注意されたい．

出力値が *0* をとる確率と *1* をとる確率が等しいと仮定すると，どちらの枝に最初に分岐しても平均評価時間は同じである． ここでは，補助変数に到達した場合，*1* 枝を最初に探索した場合の例を示す． 以後，同様に探索を行い，*10* 本の枝を通過することで $(f_0, f_1, f_2, f_3) = (1,0,0,1)$ が求まる．(例終り)

このように BDD for CF を用いて論理関数を評価する場合，平均評価時間は他の BDD と同様，平均パス長で求められる．ただし補助変数に対応する非終端節点 v_i に到達した場合のみ変更を加える必要がある．0 枝か 1 枝のどちらかは必ず終端節点 0 につながっている，この終端節点 0 につながる枝を通過する可能性はその非終端節点の節点通過確率の $1/2$ となり，もう一方の枝は必ず通過するので，結局，これら 2 つの枝の枝通過確率の合計は $1.5 \cdot P(v_i)$ となる．従って，平均評価

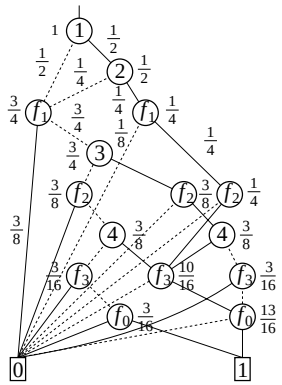


図 9: BDD for CF の平均評価時間の計算

時間を計算するアルゴリズムは次のようになる.

アルゴリズム 3.1 *(BDD for CF の平均評価時間の計算)*

1) 根の節点通過確率を *1* とする.
2) 節点通過確率が決定した非終端節点 v_i について,
  a) v_i が補助変数を表す時*:* 終端節点 *0* につながる枝の枝通過確率を $P(v_i)/2$ とし, 他方の枝の枝通過確率を $P(v_i)$ とする.
  b) v_i が入力変数を表す時*:*$P(e_{i_0}) = P(e_{i_1}) = P(v_i)/2$ とする.
3) 非終端節点について, そこに入ってくる枝通過確率がすべて決定すれば, その合計値をその節点の節点通過確率とする.
4) 上記 *1)–3)* をすべての非終端節点と終端節点 *0* の通過確率が決まるまで行う.
5) すべての非終端節点の節点通過確率の合計と, 終端節点 *0* につながる枝の枝通過確率との合計を平均評価時間とする.

例 3.5 図 *8*の *BDD for CF* の平均評価時間を求める過程を図 *9*に示す. 平均評価時間は, 非終端節点の節点通過確率の和と終端節点 *0* につながる枝の枝通過確率の和で計算でき, $1+\frac{1}{2}+(\frac{3}{4}+\frac{3}{8})+(\frac{1}{4}+\frac{1}{8})+\frac{3}{4}+(\frac{3}{8}+\frac{3}{16})+(\frac{3}{8}+\frac{3}{16})+(\frac{1}{4}+\frac{1}{8})+\frac{3}{8}+\frac{3}{8}+(\frac{3}{16}+\frac{3}{32})+(\frac{10}{16}+\frac{10}{32})+(\frac{3}{16}+\frac{3}{32})+(\frac{3}{16}+\frac{3}{32})+(\frac{13}{16}+\frac{13}{32}) = \frac{288}{32} = 9$ となる. (例終り)

このように BDD for CF は $O(n+m)$ の計算時間で評価できるので, SBDD を用いる場合より高速となる. しかし, CF の変数の合計が $n+m$ となるので節点数が大きくなり, 関数の規模が大きくなると BDD for CF を構成できなくなる.

4 BDD for ECFN を用いた多出力論理関数の表現と評価方法

我々は SBDD よりも少ない節点で論理関数を表現できる方法として BDD for ECFN(encoded characteristic function for non-zero outputs) を提案した [6]. 本節では, BDD for ECFN を概説する.

表 2: 4 出力関数の符号化法

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

4.1 出力符号化問題

ECFN は次のような写像を表現する: $F : B^n \times B^u \to B$, ここで $u = \lceil \log_2 m \rceil$. $F(a,b) = 1$ iff $f_{v(b)}(a) = 1$, ただし, $v(b)$ は, 2 値ベクトル b によって表現された整数.

例 4.1 *4* 出力関数の *ECFN* は次のとおり*:* $F = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$. (例終り)

ECFN は, FPGA の設計, 論理エミュレーション, 組込みシステムにも用いられている. BDD for ECFN は, SBDD を一般化したものと考えられる.

定義 4.1 $z^0 = \bar{z}$, $z^1 = z$.

定義 4.2 m 出力関数 $f_i (i = 0, 1, \ldots, m-1)$ に対して *ECFN* は

$$F = \bigvee_{i=0}^{m-1} z_{u-1}^{b_{u-1}} z_{u-2}^{b_{u-2}} \cdots z_0^{b_0} f_i$$

である. ここで $\boldsymbol{b} = (b_{u-1}, b_{u-2}, \ldots, b_0)$ は, 整数 i の *2* 値表現であり, $u = \lceil \log_2 m \rceil$.

z_0, z_1, ... , z_{u-1} は補助変数であり, 上記の定義では整数 i は 2 値ベクトル $\boldsymbol{b}$ に素直に変換しただけであった. しかし, 出力の符号化法を工夫することで表現を簡単にできる.

定義 4.3 *BDD* の節点数 *nodes(BDD)* とは*, BDD* の総節点数 *(*非終端節点数および終端節点数の合計*)* である.

例 4.2 *4* 出力関数 $F = (f_0, f_1, f_2, f_3)$ を考える. ここで, $f_0 = 0$, $f_1 = x_1$, $f_2 = x_2$, $f_3 = x_1 \vee x_2$ とする. 表 *2*の *Encoding1* で生成した *ECFN* F_1 は, $F_1 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_2 \vee z_1 z_0 f_3$. この場合, $F_1 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_2 \vee z_1 z_0(x_1 \vee x_2) = z_0 x_1 \vee z_1 x_2$ となる. 表 *2*の *Encoding2* で生成した *ECFN* F_2 は, $F_2 = \bar{z}_1\bar{z}_0 f_0 \vee \bar{z}_1 z_0 f_1 \vee z_1\bar{z}_0 f_3 \vee z_1 z_0 f_2$. この場合, $F_2 = \bar{z}_1\bar{z}_0 0 \vee \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0(x_1 \vee x_2) \vee z_1 z_0 x_2 = \bar{z}_1 z_0 x_1 \vee z_1\bar{z}_0 x_1 \vee z_1 x_2$. となる. このように符号化法により得られる関数も異なる. (例終り)

4.2 変数順序の最適化

図 7に示した SBDD は, BDD for ECFN でもある. 以下では, このような BDD を単に SBDD と記す. 図 10に示すように SBDD では補助変数は必ず根の側に存在し, 入力変数はその下に存在する. BDD for ECFN では, 補助変数と

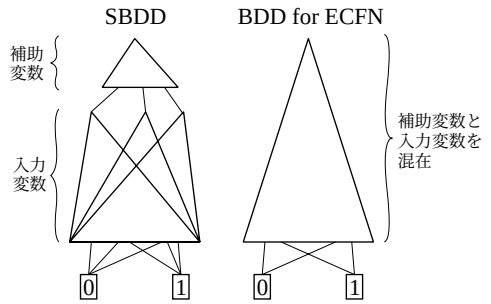


図 10: SBDD と BDD for ECFN

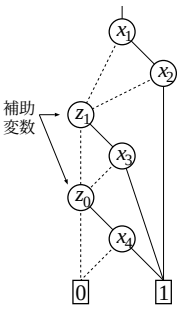


図 11: 変数順序を最適化した BDD for ECFN

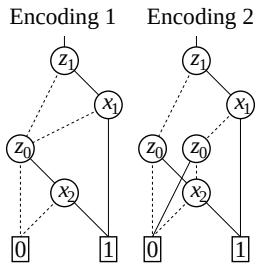


図 12: 出力符号化による節点数の変化

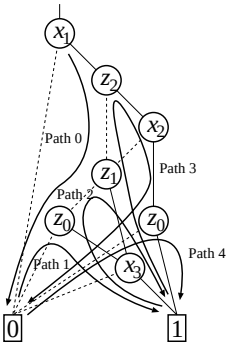


図 13: BDD を辿ることで全ての出力値を求める方法

入力変数の順序に制限はないので, 混在させ変数順序の最適化を行うことで節点数の削減が可能となる.

定理 4.1

$$nodes(BDD\ for\ ECFN) \leq nodes(SBDD)$$

例 4.3 図 *7*の *SBDD* を *BDD for ECFN* と考え, 変数順序の最適化を行うと図 *11*の最適化された *BDD for ECFN* が得られる. (例終り)

このように, BDD for ECFN を用いると, SBDD よりもコンパクトにできる. 出力符号化と変数順序の両方を考慮することによって, より節点数の少ない BDD for ECFN が得られる [5, 6].

例 4.4 例 *4.2*の *Encoding1* と *2* とで得られた F_1 および F_2 を表す *BDD for ECFN* を図 *12*に示す. このように, 節点数が *6* と *7* となり出力符号化法によっても節点数が異なる. (例終り)

4.3 高速評価アルゴリズム

図 11の BDD for ECFN で注目すべきことは節点数だけでなく高速評価の可能性である. 例えば, 図 11が表す関数に $(x_1, x_2, x_3, x_4) = (1, 1, 0, 0)$ を加えたとき, 1 回の探索だけで f_0, f_1, f_2, f_3 の全てが 1 であることがわかる. 一方, SBDD を用いると m 出力関数の場合, 必ず m 回の探索が必要となる. このように, BDD for ECFN では, 一回の探索で複数出力の評価が可能な場合がある.

例 4.5 図 *13*のような *BDD for ECFN* を考えよう. 注目すべきことは, パス上に必ずしも全ての補助変数が出現しないことである. この *BDD* に対して $(x_1, x_2, x_3) = (1, 1, 1)$ を加えた場合, パス $0 \sim 4$ の *5* 個のパスに従って探索するだけで *8* 出力分の値 $(0, 1, 1, 1, 0, 1, 0, 1)$ が求まる. 実際, パス *0* では f_0 が, パス *1* では f_1, パス *2* では f_2, f_3 が同時に, パス *3* では f_4, f_6 が同時に, パス *4* では f_5, f_7 が同時に求まる. (例終り)

このように探索中, 補助変数を表す節点で 0 枝と 1 枝の両方を順次探索していくことで全ての出力を無駄なく探索し, 全ての出力値を求められる.

一つの入力に対して全ての出力を求める場合, 無駄のない評価アルゴリズムが必要となる. このアルゴリズムを次に示す.

アルゴリズム 4.1 *(ECFN* を用いた論理関数の評価*)*

$eval()\{$
　$u = \lceil \log_2 m \rceil;$
　$evalp(root, u);$
　出力リストの括弧を展開*;*
$\}$

```
evalp(p, lastIndex){
    if (p == 非終端節点){
        if (p == 入力変数の節点){
            if (x[p → index] == 0)
                evalp(p → 0枝, lastIndex);
            else
                evalp(p → 1枝, lastIndex);
        } else if (p == 補助変数の節点) {
            currentIndex ← 現在の節点の index;
            printf (2^(lastIndex−currentIndex−1));
            printf ("(");
            evalp(p → 0枝, currentIndex);
            evalp(p → 1枝, currentIndex);
            printf (")");
        } else {/* 終端節点 */
            printf (2^currentIndex);
            printf ("(");
            printf (現在いる終端節点の値);
            printf (")");
            return ():
        }
    }
}
```

平均評価時間は，BDD のときと同様に平均パス長から計算できる．ただし，補助変数が BDD 中に散在するので (z_2, z_1, z_0) を順次 $(0,0,0)$, $(0,0,1)$, $(0,1,0)$, ... , $(1,1,1)$ と変化させ変化分から葉に近いところのみのパス長を加算すれば求まる．

例 4.6 図 *11の BDD for ECFN* の平均評価時間を求める手順を図 *14(a)*〜*(e)* に示す．平均評価時間は，$1+\frac{1}{2}+\frac{6}{4}+\frac{3}{4}+\frac{18}{8}+\frac{9}{8}=\frac{57}{8}=7.125$ と計算できる．（例終り）

5 実験結果

論理関数の評価を行う際重要なのは，BDD の大きさと評価時間である．ここでは，ベンチマーク関数に対して BDD を構成し，その大きさと平均評価時間を求める．

MCNC ベンチマーク関数に対して MTBDD，BDD for CF，SBDD，BDD for ECFN を構成し，BDD の大きさと平均評価時間を求めたものを表 3に示す． 各決定グラフの変数順序最適化には，文献 [4] のアルゴリズムを，BDD for ECFN を作成する際の出力の符号割当は，文献 [5] のアルゴリズムを用いた．表中の Name は関数名，In は入力数，Out は出力数，Nodes は節点数，Eva は平均評価時間を示す．BDD for ECFN の Min の列は ECFN を出力を表す補助変数と入力変数を混在して最適化した場合，Fast の列は補助変数を終端側に集めた場合の結果である． 最終行の Ratio は SBDD の結果を 1.0 とした時の比率の平均である．また，SBDD の平均評価時間は，すべての出力値を計算するために必要な評価時間を表す．

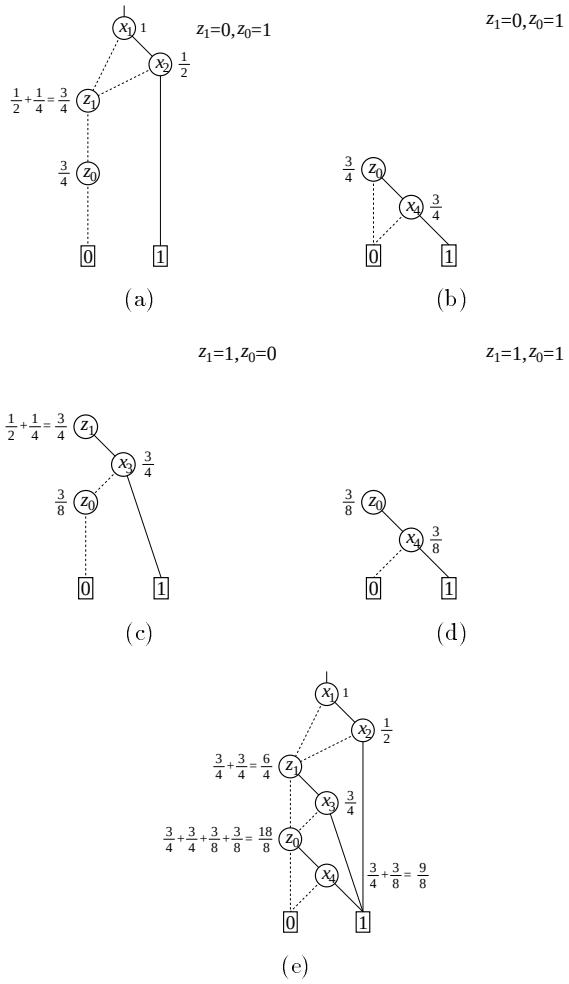


図 14: BDD for ECFN の平均評価時間

例えば，ts10 の場合 MTBDD の節点数は Nodes=589837，平均評価時間は Eva=5.5，BDD for CF では Nodes=589826，Eva=31.5，SBDD では Nodes=163，Eva=88.0，BDD for ECFN では Nodes=83，Eva=14.5 であり，BDD for ECFN を使用すれば，SBDD に比べ約 51% の節点で表現でき，評価速度は約 6 倍となる．MTBDD と比べると節点数は 0.014%，評価速度は 0.38 倍である． 一方 mainpla の場合，MTBDD の節点数が最小で Nodes=632 であり，平均評価時間は Eva=4.2，BDD for CF では Nodes=3114，Eva=85.2，SBDD では Nodes=1857，Eva=277.5，BDD for ECFN では Nodes=1018，Eva=49.2 となっており，MTBDD を用いると最も高速に評価できる．

総合的に見ると，MTBDD では，出力を表す補助変数が不要で，入力を決定すれば全ての出力が決定するので，平均評価時間は最短で，SBDD の 0.072 倍である． しかし，節点数は 271 倍必要となり，実用上問題がある．BDD for CF では，平均評価時間は SBDD の 0.36 倍，節点数は 163 倍必要となる． BDD for ECFN では，変数順序を最適化した場合，節点数は約 81%，平均評価時間は 0.41 倍であり，都合が良い．BDD for ECFN は SBDD と異なり，出力を表す補助変数の

表 3: 決定グラフの大きさと平均評価時間

Name	In	Out	MTBDD		BDD for CF		SBDD		BDD for ECFN			
									Min		Fast	
			Nodes	Eva	Nodes	Eva	Nodes	Eva	Nodes	Eva	Nodes	Eva
apex1	45	45	942	8.8	3594	76.3	1324	269.8	1087	31.8	1297	29.6
apex3	54	50	537	6.6	2449	81.6	986	286.6	708	28.2	751	25.8
duke2	22	29	662	6.4	997	49.9	366	150.3	346	22.5	846	15.1
e64	65	65	131	2.0	260	99.5	194	256.0	194	256.0	246	12.4
exep	30	63	1170	7.8	3030	102.3	675	255.7	660	38.0	1293	14.5
k2	45	45	929	8.8	3594	76.3	1321	269.8	1167	29.1	1303	32.2
mainpla	27	54	632	4.2	3114	85.2	1857	277.5	1018	49.2	1016	49.1
mark1	20	31	4138	6.4	745	52.9	119	115.7	117	109.2	4358	28.9
misex2	25	18	118	4.9	184	31.9	100	75.6	98	18.9	138	15.7
opa	17	69	241	4.4	1778	107.9	428	322.2	364	37.8	484	35.7
pdc	16	40	19178	10.2	5852	70.2	596	215.4	590	181.1	20630	43.7
rckl	32	7	65	2.0	135	12.5	198	100.6	67	4.8	67	4.8
seq	41	35	378	3.3	1197	55.8	1284	168.0	493	28.2	498	28.2
shift	19	16	196095	15.5	3746	39.5	78	86.0	62	55.0	196095	32.1
spla	16	46	11100	9.7	7522	78.1	628	226.6	604	99.9	12016	26.1
t2	17	16	304	7.5	484	31.7	145	72.9	140	44.9	357	16.8
table5	17	15	436	8.0	677	30.5	685	114.1	476	10.6	476	10.7
ts10	22	16	589837	5.5	589826	31.5	163	88.0	83	14.5	589837	7.8
vg2	25	8	420	11.8	471	23.8	90	48.9	82	45.7	425	15.2
x1dn	27	6	214	9.8	245	21.2	139	41.0	139	41.0	228	16.7
x6dn	39	5	195	4.1	225	11.6	235	41.2	193	13.4	196	4.6
x9dn	27	7	292	9.8	237	20.3	139	50.4	139	50.4	311	14.0
xparc	41	73	3844	3.6	4773	113.1	1947	304.8	1232	21.8	4510	19.0
Ratio			271.3	0.072	162.5	0.36	1.0	1.0	0.81	0.41	271.7	0.163

Min: 補助変数の位置を任意にした場合　　Fast: 補助変数の位置を終端側に集めた場合
Eva: 平均評価時間　　Ratio: SBDD を 1.0 とした時の比率の平均

位置を任意の場所にできるので, 節点数は少なくできる. 出力を表す補助変数が終端側に近い程, 平均評価時間は短くなる. そこで, 補助変数を終端側に集めた場合について実験を行った結果, 節点数は約 272 倍, 平均評価時間は 0.163 倍となった. 節点数は MTBDD と同程度であるが, 平均評価時間は MTBDD の約 2.3 倍である.

6 まとめ

多出力論理関数の評価に適した 4 つの BDD の形式を述べた. 次に, その節点数と平均パス長に関する特徴を述べた. 最後に, SBDD に対して BDD for ECFN の節点数は 19%, 評価時間は 59% 削減できることを実験的に示した.

謝辞

本研究は, 一部, 日本学術振興会科学研究費補助金による.

参考文献

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

[2] Y. Iguchi, T. Sasao, M. Matsuura, and A. Iseno "A hardware simulation engine based on decision diagrams," *Asia and South Pacific Design Automation Conference (ASP-DAC'2000)*, Jan. 26-28, Yokohama, Japan.

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