2分決定グラフのための変数順決定アルゴリズムとその評価†

藤田昌宏†† 藤沢久典††
松永裕介†† 角田多苗子††

Bryant は論理関数を効率的に処理する手法として，場合分けの変数順をすべてのパスで固定した2分決定グラフ (BDD) を提案し，BDD の効率的な演算アルゴリズムを示した．64ビットの ALU が表現できる等，BDD は従来の論理関数表現手段と比べ，極めて強力であることが示されたが，場合分けの変数順によって，BDD の大きさが大きく変化するという問題点がある．そこで本稿では，BDD の変数順についての各種実験結果と，それらから得られた変数順決定アルゴリズムについて述べる．まず，ベンチマーク回路に対する実験結果から，人が描く回路図上に現れる変数を上から順に順序付けしたものが極めてよい結果をもたらすことを示す．次に，これの根拠となる定理を幾つか示し，それらの定理に基づく変数順決定アルゴリズムとその評価を示す．この変数順決定アルゴリズムは回路図から得られる順よりもよい結果をもたらし，100入力，3,000ゲート以上の回路を BDD で表現できるようになった．

1. はじめに

論理関数をコンパクトに表現する手法として，2分決定グラフ (Binary Decision Diagram, 以下 BDD と略す) が Bryant によって提案されている[1]．これは，従来の決定木に対し，決定木の根から葉へのすべてのパスについて，変数順を固定することで，サブグラフの共有（もとは決定木だが共有することでグラフとなる）を可能としたものである．例えば，論理式 $X1X2+X3'$ は図1のような BDD で表現できる（以下，本稿では，変数は英字1字に数字を付けたもので表す．論理積記号は省略し，論理和記号には『+』を用い，否定は否定をとるべき式の後ろに『'』を付ける．BDD は変数順が固定されていることから，Ordered BDD と呼ばれることもある)．図で，○内は変数番号を示し，辺に付いている 0，1 は，各々その変数が値0または1をとる場合を表している．また，□は定数を示す．

このように，変数順が固定されていると，reduce と呼ばれるグラフの冗長性を削除する操作を施すことにより，もとの論理が同じであれば，対応する BDD も同形になることが示され[1]，また，2つの BDD が同形であるか否かの判定は，単に，2つの BDD をそれぞれたどればよく（BDD のノード数に比例する時間で済む．実際には，BDD の生成時間に比べ，無視できる程度)，論理検証にも容易に応用することができる．さらに，Bryant は BDD 間の演算を行う apply，compose 操作，BDD の冗長性を削除する reduce 操作のアルゴリズムを示し，実際に64ビットの ALU の検証に適用することで，その有効性を示した．

しかし，BDD の大きさは，論理が同じでも変数順によって大きく変化することが分かっている．例えば，同じ論理 $X1X2+X3X4+X5X6$ に対し，最適変数順で BDD に変換したものを図2(a)に最悪変数順で BDD に変換したものを(b)に示す．この例の場合，2入力 AND ゲートの数を増やしていくと（それに比例して変数の数も増加する)，最適順では，変数の数に比例するグラフの大きさで済むが，最悪順では，変数の数に対し指数的にグラフの大きさが増大する．このように変数順により BDD の大きさは大きく変化する．

もちろん，2つの論理式の論理的同一性判定問題は，NP 完全問題に属することから，どのような変数順を選んでも，グラフの大きさが変数の数に対し指数的に大きくなる論理が存在する．実用的な回路では，組合せ回路で実現された乗算器がそうである．しかし，実回路への適用結果によると，よい変数順を用いれば，他の手法，例えば，積和形で表現する方法では検証できない回路も検証できており，実用的価値は高いと言える．

最適変数順探索問題も NP 完全問題であることが分かっており[2]，変数の数に対し指数の手間がかかると考えられる．実際に扱われる回路は変数の数が 50 以上であることもあり，最適変数順を求めることは，特

† An Ordering Algorithm for Binary Decision Diagrams and Its Evaluation by MASAHIRO FUJITA, HISANORI FUJISAWA, YUSUKE MATSUNAGA and TAEKO KAKUDA (FUJITSU LABORATORIES LTD.).

†† (株)富士通研究所

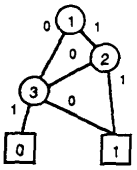
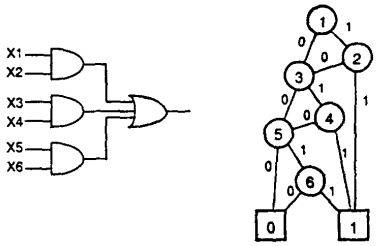
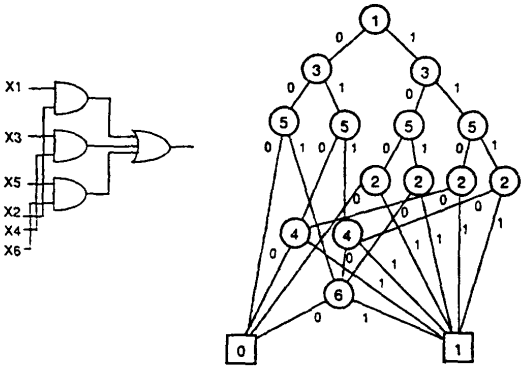


図 1 $X1X2+X3'$ に対する BDD
Fig. 1 BDD for $X1X2+X3'$.



(a) 最適順
(a) Best ordering.

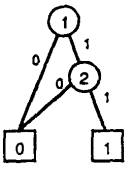


(b) 最悪順
(b) Worst ordering.

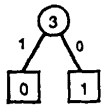
図 2 変数順の影響
Fig. 2 Variable ordering: best and worst.

殊な回路を除き，ほぼ不可能である．したがって，よい変数順を決定できる発見的手法の開発が，実際的に極めて重要となる．本論文では，この変数順決定アルゴリズムを示すとともに，ベンチマーク回路に対する適用結果からその有効性を示し，さらに，変数順決定アルゴリズムの有効性の理論的根拠について述べる．

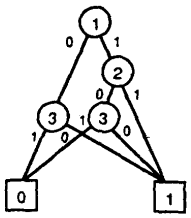
2章では，BDD の演算の簡単な説明と，BDD の応用について述べる．3章では，ベンチマーク回路に対し，元の順，乱数順，回路図を描いた時に現れた順，の各変数順を用いて実験した結果から，回路図を描いた時に現れた順が極めて有効であることを示す．さらに，回路図を描いた時に現れた順が有効であることの根拠について考察する．考察の結果，回路図上でネットの交差をできるだけ少なくするような配置から得



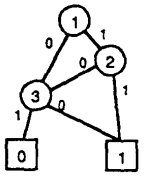
(a) $X1X2$ に対する BDD
(a) BDD for $X1X2$.



(b) $X3'$ に対する BDD
(b) BDD for $X3'$.



(c) $X1X2$ と $X3'$ に和演算を apply
(c) Apply "or" to ($X1X2$) and $X3'$.



(d) reduce 操作後
(d) After "reduce" operation.

図 3 apply と reduce 操作
Fig. 3 "Apply" and "reduce" operations for BDD.

られる変数順がよいことが示される．4章では，3章の結果に基づく発見的変数順決定アルゴリズムと，そのベンチマーク回路に対する適用結果を示す．最後に5章で結論を述べる．

2. BDD の演算と応用

Bryant は，BDD 間の各種演算を行うための効率的な操作アルゴリズムを示した[1]．これらの操作では，扱う BDD を一度辿っていきながら，各ノードに必要な演算を施しているため，BDD の大きさ（ノード数）に比例する時間で処理することができる（複数の BDD を操作する場合には，各 BDD の大きさの積に比例）．

まず，apply 操作によって，BDD 間に各種演算（論理演算等）を施すことできる．例えば，図3(a)，(b)に示す2つの BDD に論理和演算を apply した結果を図3(c)に示す．一般に，apply の結果には，冗長性があるので，reduce 操作によってその冗長性を削除し，サブグラフの共有を行う．図3(c)に reduce 操作を施した結果を図3(d)に示す．reduce 操作の結果は，正規形であり，2つの論理の比較は，単に2つの BDD が同形であるかを調べればよい．この apply，reduce 操作の繰り返しによって，任意の論理式（組合せ回路）に対応する BDD をボトムアップに作成していくことができる．

また，2つの論理関数の合成を計算する compose

表 1 ISCAS ベンチマーク回路
Table 1 ISCAS benchmark circuits[8].

Circuits	Inputs	Outputs	Levels	Gates	Max gate for single output	Max inputs for single output
sn 181	14	8	9	34	55	14
c 432	36	7	7	203	145	36
c 499	41	32	11	275	102	41
c 880	60	26	24	469	130	45
c 1355	41	32	24	619	322	41
c 1908	33	25	24	619	322	41
c 2670	233	140	32	1566	828	122
c 3540	50	22	47	1741	1433	50
c 5315	178	123	49	2608	937	67
c 6288	32	32	123	2480	2327	32
c 7552	207	108	43	3827	1096	194

操作も apply と同様に用意されており，論理関数を効率的に操作することができるため，BDD の応用範囲は極めて広い．2つの組合せ回路の論理を比較する論理照合は，回路に対する BDD を作成し，同形か否かの判定を行えばよい．また，記号シミュレーション[4]では，シミュレーション結果の解析が重要であるが，BDD は正規形なので，容易に解析することができる．

以上の論理検証以外でも，テスト生成[3]や論理回路簡単化等へも容易に応用することができる．回路の構造から生じるドントケアを利用しながら回路変換と冗長性除去を行うことで論理回路を簡単化するトランスダクション法[5),6]も，内部表現として，ドントケアも定数値に含めた BDD を利用することで，極めて効率的に実行することができる[7]．

以上のように，BDD は極めて強力な論理関数表現手段であるが，変数順によってその大きさが大きく変化するという問題がある．本稿では，以下，この変数順決定アルゴリズムについて考察し，実用的なアルゴリズムを示す．

3. BDD のための変数順に関する実験・考察

変数順の影響を調べるために，テスト生成で用いられているベンチマーク回路[8]を用いて次の3種類の変数順を評価した．

(1) もとの変数順（プログラムの都合上，もとのファイルに現れる変数順の逆順になっている）

表 2 ISCAS ベンチマーク回路に対する BDD の作成
Table 2 BDD generation of ISCAS benchmark circuits.

Circuits	Original order		Random order		Manual order	
	Time	Max nodes	Time	Max nodes	Time	Max nodes
sn 181	4	339	4	578	4	197
c 432	55	1146	N. F.	>100000	27	442
c 499	464	9020	N. F.	>100000	367	4661
c 880	N. F.	>100000	N. F.	>100000	43	3421
c 1355	388	9020	N. F.	>100000	N.T.	N.T.
c 1908	483	2912	>50000	N. F.	900	4505
c 2670	N. F.	>100000	N. F.	>100000	N.T.	N.T.
c 3540	N. F.	>100000	N. F.	>100000	N.T.	N.T.
c 5315	1070	11807	N. F.	>100000	N.T.	N.T.
c 6288	N. F.	>100000	N. F.	>100000	N.T.	N.T.
c 7552	N. F.	>100000	N. F.	>100000	N.T.	N.T.

>100000: Number of nodes exceeds 100000, >50000: CPUTIME exceeds 50000 seconds, N. F.: Not finished, N. T.: Not tried, Machine: SUN 3/260, All outputs are processed one by one.

(2) 乱数順

(3) 人手でネットリストから回路図を描き，その回路図上で上から順次現れた変数順

ベンチマーク回路の規模を**表 1** に，結果を**表 2** に示す．なお，ベンチマーク回路は多出力であるが，表2では，各出力ごとに回路を切り出して順に処理している（上の(3)でも回路図は各出力ごとに切り出したものに対し描いている）．処理時間は合計であり，ノード数は各出力の BDD のノード数の最大値である．使用計算機は SUN 3/260 で，CPUTIME が 50,000 秒を越えるか，途中の BDD のノード数が 100,000 を越えた場合には処理を打ち切った．表中，N. F. は処理を打ち切ったことを示し，N. T. は試行しなかったことを示す（大きな回路に対し，回路図を描くことは困難なので，試行を諦めた）．この表から分かるよう

に乱数順では大規模回路を扱うことができないが，回路図に現れた順からは極めて良好な結果が得られている．また，元の順も乱数に比較するとかなりよい結果が得られている．これは，元の回路データも何らかの回路図上の配置情報でソートされているからとも推測でき，この実験から回路図を描いた時に現れる順がよいと言える．では，なぜ，回路図を描いた時に現れる順がよいのであろうか．本章では，以下この根拠について考察していく．

考察の準備として，幾つかの定義をする．ここでは，組合せ回路のみを考え，回路に閉路はないとする．回路の入力数を n，出力数を m，ゲート数を g とする．各入力は，$X1, \cdots, Xn$ で表し，各出力を $Z1, \cdots, Zm$ で表す．回路の各入力と各ゲートには，次に述べる制限を満たすインデックス付け T がされているとし，インデックス1から $(n+g)$ が付いているとする．ここで制限とは，ゲートにインデックスを付ける際には，そのゲートのすべての入力に既にインデックスが付いている場合に限り，かつ，逆に，あるゲートのすべての入力にインデックスが付いている場合には，次にそのゲートにインデックス付けすることである．このような制限を加えても，入力間の順序には制限はないため，すべての変数順を考慮することができる．インデックス値 r による回路の分割とは，回路を $C1$ と $C2$ に分け，$C1$ にはインデックス値が r 未満の入力またはゲートが属するようにし，$C2$ にはインデックス値が r 以上の入力またはゲートが属するようにすることである．この際，$C1$ から $C2$ へのネットは現れるが，インデックス付けに対する制限から $C2$ から $C1$ へのネットは現れない．この $C1$ から $C2$ へのネットの数を $w(r, T)$ で表し，それぞれのネットを $Y1, \cdots, Yw(r, T)$ とする．また，$C1$ に属する入力変数を $X1, \cdots, Xi$ とし，$C2$ に属する入力変数を $Xi+1, \cdots, Xn$ で表す．以上を図示すると**図4**のようになる．さらに，$w(r, T)$ の r に関する最大値を $w(T)$ とする．

Berman は，BDD の大きさと回路図上の幅との関係に関して，次の定理を示した．

〈定理1〉(Berman[9])

与えられた T に対し，BDD の大きさは

$$n \cdot m \cdot 2^{w(T)} + m$$

で押さえられる．

(略証)

回路をインデックス r で分割したとすると，$C2$ からみて，$C1$ の出力値は $2^{w(r,T)}$ 通りしかない．したがって，$C1$ に属する入力変数が決定する関数の種類は $2^{w(r,T)}$ 通り以下である．これは，BDD 上では，変数 Xi に対するノードから出ているすべてのエッジの指しているノードのうち，異なるものの数（変数 Xi の直下のノード数）は

$$2^{w(r,T)}$$

個以下であることを意味する．$2^{w(r,T)}$ の r に関する最大値が $2^{w(T)}$ であることから，任意の変数の直下のノード数が $2^{w(T)}$ で押さえられ，結果として，BDD 全体のノード数が $n \cdot m \cdot 2^{w(T)} + m$ で押さえられる（各出力ごと別々に BDD を作るとしている．m を最後に加えているのはルートノード分である）（略証終わり）．

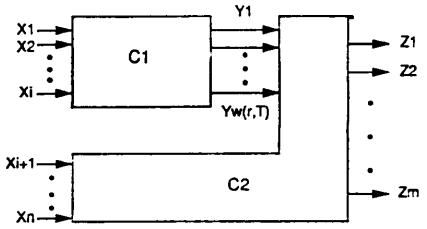


図4　インデックス値 r による回路の分割
Fig. 4　Partition of a circuit by index r.

このように，$w(r, T)$ を小さくするようなインデックス付け T を使えば，より小さな数で BDD の大きさを押さえることができ，結果として，より小さな BDD が得られると期待できる（上の定理は BDD の大きさの上限であり，必ずしもより小さい BDD を保証するものではない）．実際には，上の定理より強く BDD の大きさを押さえることができる．上の略証の仮定から分かるように，変数 Xi の直下のノード数は $2^{w(r,T)}$ 通り押さえられる．証明では単純にこの値の r に関する最大値をとっているが，実際には，入力変数の分割 $\{X1, \cdots, Xi\}$ と $\{Xi+1, \cdots, Xn\}$ を満たす範囲の r の集合を Ri とすると，$wi(T) = \max_{k \in Ri}(w(k, T))$ で押さえられる．したがって，次が成立する．

〈定理1系〉

与えられた T に対し，BDD の大きさは

$$m \cdot \sum_{i=1}^{n} 2^{wi(T)} + m$$

で押さえられる．

今，信号は必ず左から右へ流れる回路図が与えられたとする．前に示した制限の下で，インデックス付けを行うことを考えるが，インデックス付けに自由度がある場合には（入力のインデックス付けや，すべての入力がインデックス付けされているゲートが複数存在

する場合），必ず，回路図上で上に配置されているものを先にインデックス付けるようにする．このインデックス付け T に対し，定理1系を適用することにより，その回路図上で上から現れた入力変数の順に対する BDD の大きさの上限を得ることができる．その際，$\sum 2^{wi(T)}$ が小さくなる回路図がより小さい値で，BDD の大きさを押さえることができることになる．$wi(T)$ は，回路図上では，回路図をある断面で切った時に，その断面を通過するネット数である．したがって，$\sum 2^{wi(T)}$ を小さくするためには，各断面を通過するネット数をそれぞれ小さくするとともに，（指数の和であることから）その最大値を小さくすることが重要であると言える．このように断面を通過するネットワーク数をより小さくするような回路図が，よりよい変数順を与えると期待できる．一般に，ネットの交差数を小さくするような配置に基づく回路図のほうがより，この断面を通過するネット数を小さくすると考えられる．したがって，ネットの交差数が小さい回路図から得られる変数順が，より小さい値で BDD の大きさを押さえることが期待できる．人手が描く回路図は，回路を見やすくする必要から，ネットの交差数が小さくなっていると考えられる．

以上は，表2で回路図から得られる順がよい結果になるわけを完全ではないにしても，ある程度説明している．したがって，人手の回路図から得られる順に近い変数順を生成するアルゴリズムを考えることが重要となる．しかし，これは回路素子の配置問題であり，最適に近い解を得ることは，簡単ではなく，得られるアルゴリズムも計算時間のかかるものとなってしまうことが予想される．そこで，次章では，より単純で高速なアルゴリズムを考える．

4. BDD のための変数順決定アルゴリズムとその評価

4.1 BDD のための変数順決定アルゴリズム

前章で，人が描いた回路図上の変数順がよいことが示された．もし，人による回路図を入力したものを扱う場合には，その回路図上での変数順をそのまま用いればよいことになる．しかし，このような変数順を高速に自動生成することは簡単ではない．そこで，以下の2つの定理の結果を踏まえて，より単純なアルゴリズムを考える．

〈定理2〉

外部入力も含めて完全な（AND, OR, NAND, NOR, NOT のみからなる）樹枝状回路（すべての入力とゲートのファンアウト数が1）に対しては，出力から入力への回路を深さ優先で辿っていった際に見つかる入力の順が，BDD の最適変数順の1つである．

（略証）

完全な樹枝状回路なので，各ゲートの BDD を apply と reduce 操作によって作っていく際，操作を施す各 BDD に現れる変数の集合が，変数順に対してオーバラップがなければ，個々の BDD を適当な操作でつなぐだけで出力の BDD が得られる．その際，元の個々の BDD の大きさ（つまり，BDD のノード数）が変数順を変化させた時の最小（つまり，元の個々の BDD のノード数が，冗長変数のない n 変数関数に対する BDD のノード数の最小値である $n+2$）であれば，変数にオーバラップはないので BDD の一意性により，出力の BDD の大きさも最小となる（出力の BDD を作る際に新たにノードは増えないからである．例については，**図5**参照．AND ゲートの各入力の BDD をそれぞれA，Bとすると，出力に対する BDD は，図に示すように，各 BDD の定数ノードへの辺を結合することにより作成できる．図で，$A0$, $A1$, $B0$, $B1$ はそれぞれ，BDD の変数ノードから定数ノードへの辺の集合を表しているとする）．このようなオーバラップを生じない変数順の1つは定理のような深さ優先のスキャンで得られる（略証終わり）．

〈定理3〉

外部入力も含めて，ただ1箇所のみファンアウトが2である（AND, OR, NAND, NOR, NOT のみからなる）回路に対しては，原則として，出力から入力へ回路を深さ優先で辿っていくが，ファンアウトが2のゲートまたは入力が現れた時には，そのゲートまたは入力を先に辿るようにした際に見つかる変数順が，BDD の最適変数順の1つである．

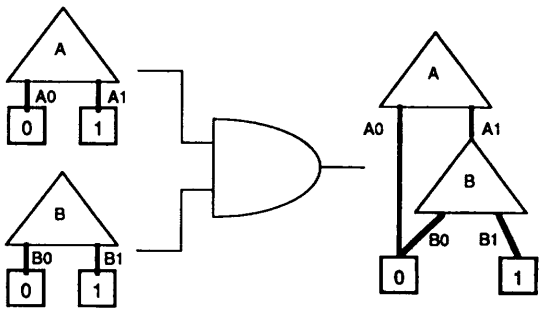
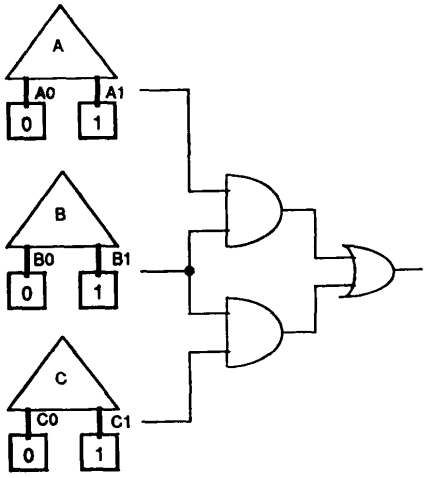


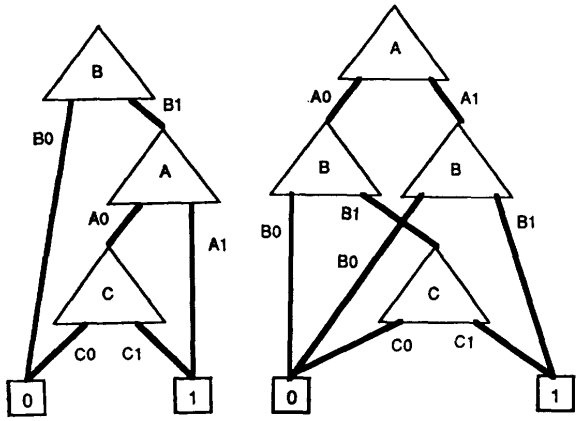
図5 木状回路に対する BDD の構築
Fig. 5 Construction of BDD for tree circuits.

(略証)

定理3と異なるのは，ファンアウトが2である際の扱いのみである．ファンアウトが2である部分に対応するBDDは2箇所で使われるため，出力のBDDでは，上側に置かないと出力のBDDでも2箇所現れる(**図6**参照．この回路の出力に対するBDDは，Bに現れる変数を先に順序付けると(b)のようになりBは1箇所にしか現れないが，Bに現れる変数を後に順序付けると(c)のようにBは2箇所に現れる)．したがって，ファンアウト2の部分は先に辿るようにする(略証終わり)．



(a) ファンアウトが2である点が1箇所しかない回路
(a) A circuit which has only one 2-fanout point.



(b) Bに含まれる変数を先に順序付けした場合の出力に対するBDD
(b) B is ordered first.

(c) Bに含まれる変数を先に順序付けしなかった場合の出力に対するBDD
(c) B is not ordered first.

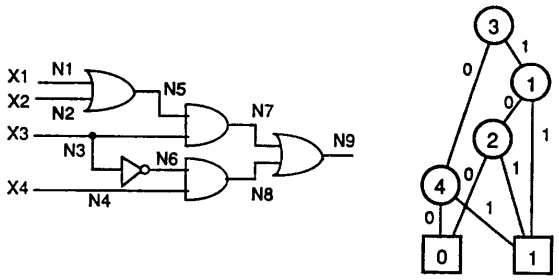
図6 ファンアウトが2である点が1箇所しかない回路に対するBDDの構築
Fig. 6 Construction of BDD for circuits which have only one 2-fanout point.

定理2，3から，少なくとも完全樹枝状回路に近い回路では，基本的に出力から入力へ深さ優先で辿った時に現れる順がよいことになる．定理3ではさらに，ファンアウトの大きいゲートや入力を先に辿るほうが(樹枝状回路に近い場合には)よいことを示している．これらは，ファンアウトの多い一般回路にもそのまま通用するわけではないが，基本的に深さ優先で辿ることは，3章の結果である，人手の回路図に近い順になるとも考えられる．そこで我々は**図7**に示すように，基本的に出力から入力へ深さ優先で辿るが，入力変数の部分のみファンアウトが1のものを比較的後に置く，変数順決定アルゴリズムを考えた[10]．入力変数のうち，ファンアウトが1のものだけ考慮したのは，その部分は比較的樹枝状回路に近いと考えたからである．**図8**(a)の回路に対し，図7のアルゴリズムを適用した実行トレースを図8(b)に示す．FANOUT2UPは各ゲートに対し，そのゲートより入力側でファンアウトが2以上の入力を発見したか否かのフラッグであり，FANOUT1LISTにはファンアウトが1の入力が一時蓄えられる．これは，次のファンアウト2以上の入力を発見した時に，その変数の順のすぐ後に付け加えられる．各入力に対するファンアウトの計算は，そ

```
procedure makeOrder(O); /* O is an output */
intlist FANOUT1LIST;
intarray MARK;
begin
   foreach gate or input T do MARK[T] = off;
   FANOUT1LIST := NIL;
   makeOrder0(O);
   ORDER := append(ORDER,FANOUT1LIST);
end;

function makeOrder0(G);
int FANOUT2UP;
begin
   FANOUT2UP := undef;
   foreach I of fanins of the gate G in an appropreate order* do
   begin
      if MARK[I] <> off then
      begin
         FANOUT2UP := MARK[I];
         continue;
      end;
      if I is a primary input then
         if I's fanout > 1 or FANOUT2UP <> undef then
         begin
            if I's fanout > 1 then FANOUT2UP := I;
            if I is not in ORDER then ORDER := append(ORDER,I);
            ORDER := append(ORDER,FANOUT1LIST);
            FANOUT1LIST := NIL
         end else FANOUT1LIST := append(FANOUT1LIST, I);
      else FANOUT2UP := makeOrder(I);
   end;
   if FANOUT2UP <> undef then ORDER := append(ORDER,FANOUT1LIST);
   MARK[G] := FANOUT2UP;
   return(FANOUT2UP);
end;
```

図7 BDDのための変数順序付けアルゴリズム
Fig. 7 Ordering algorithm.



(a) 例題の回路とその BDD
(a) A sample circuit and its BDD.

Traverse of circuit	FANOUT2UP	FANOUT1LIST	Order
N9	undef	NIL	NIL
N7	undef	NIL	NIL
N5	undef	NIL	NIL
N1	undef	X1	NIL
N2	undef	X1,X2	NIL
N5	undef	X1,X2	NIL
N3	3	X1,X2	X3
N7	3	NIL	X3,X1,X2
N9	3	NIL	X3,X1,X2
N8	undef	NIL	X3,X1,X2
N6	undef	NIL	X3,X1,X2
N3	3	NIL	X3,X1,X2
N8	3	NIL	X3,X1,X2
N4	undef	X4	X3,X1,X2
N8	3	NIL	X3,X1,X2,X4
N9	3	NIL	X3,X1,X2,X4

(b) 図7のアルゴリズムの実行トレース
(b) Trace of the ordering algorithm in Fig. 7.

図 8 図7に示したアルゴリズムの実行例
Fig. 8 Sample trace of the ordering algorithm in Fig. 7.

の出力がバッファやインバータ等の1入力のゲートに接続されている場合には，そのゲートの出力のファンアウトも加えるようにする．図7のアルゴリズムをベンチマーク回路に適用した結果を**表3**に示す．表2と同じように，1出力ごとに BDD を生成した際の処理時間の合計と BDD のノード数の最大値である．図7のアルゴリズムは，1出力用であり，多出力への適用については4.2節で述べる．表3から分かるように，人手の回路図からの順と同等以上の結果が得られており，少なくとも1出力の回路に対しては，基本的に深さ優先で辿ることの有用性を示している．これらの結果は，例えば論理照合への応用を考えると，他の手法[11),12)]と比べ，はるかに高速であり，また，大規模回路が扱えると言える．

c 432 のみ結果が悪いが，これは，c 432 には多入力のゲートが幾つかあるため，単純に深さ優先で辿ると多入力のゲートが変数順をシャッフルしてしまうからと考えられる．これらの解決については，4.3 節で検討する．

4.2 変数順決定アルゴリズムの多出力への拡張

論理照合への応用では，BDD の変数順は各出力ごとに異なってもよいが，論理回路簡単化やテスト生成への応用を考えると，全出力同じ変数順で BDD が作成されている必要がある．

表 3 図7のアルゴリズムによる変数順を用いた BDD の作成
Table 3 BDD generation by the algorithm of Fig. 7.

Circuits	Algorithm of Fig. 7 order	
	Time	Max nodes
sn 181	3	213
c 432	712	6196
c 499	382	4661
c 880	28	3359
c 1355	505	4661
c 1908	389	3076
c 2670	169	14763
c 3540	6310	53460
c 5315	249	3441
c 6288	N.F.	>100000
c 7552	192	2096

>100000: Number of nodes exceeds 100000, N.F.: Not finished, Machine: SUN3/260, All outputs are processed one by one.

図7のアルゴリズムは，多出力の回路に対しても，適切な順で各出力に適用していけば，回路全体に対する変数順を得ることができる．その際，最も複雑な論理になりそうな出力を先に処理する（最も複雑な論理になりそうな出力に有利になるように変数順を決定する）のがよいと考えられる．そこで，出力の処理順位を次のように決める．

- 最も多くの入力に関係している出力を優先する．
- 同点の場合には，より多くのゲートに関係している出力を優先する（さらに同点の場合は任意）．

以上のようにして，ベンチマーク回路に対し，変数順を決定して，BDD を作成した結果を**表4**に示す（c 1355 について試行しなかったのは，c 1355 が c 499 と同じ論理であったため）．BDD は各ゲートに対して1度作成するのみでよいので，各出力から切り出した回路にオーバラップがかなりある場合には，1出力ごとに処理した場合よりも手間はかなり少なくなる．しかし，すべての出力に対し，同じ変数順を用いるため，各出力ごとには必ずしもよい変数順とはならない可能性もある．結果を見ると，ほとんどの回路で表3より2倍以上の速度向上が見られる．しかし，c 7552 に対しては，以上の方法では，全出力に対するよい変数順を見つけられなかったし，c 2670 でも速度向上は他に比べると小さくなっている．このように，回路規模が

表 4 図7のアルゴリズムによる変数順を用いたBDDの作成（多出力同時処理）
Table 4 BDD generation by the algorithm of Fig. 7 with multiple output handling.

Circuits	Algorithm of Fig. 7 order Time
sn 181	2
c 432	265
c 499	156
c 880	17
c 1355	N.T.
c 1908	221
c 2670	168
c 3540	3807
c 5315	127
c 6288	N.F.
c 7552	N.F.

N.F.: Not finished, N.T.: Not tried, Machine: SUN 3/260, Same ordering is used for all outputs in a circuit to generate BDD.

大きくなればなるほど，全出力に対するよい変数順を決定することは困難になっていく．

4.3 入力変数のグループ化

表2と表3を比較すると，c 432 のみ表3のほうが結果が悪くなっている．これは，c 432 には多入力のゲートがあるため，これらにより，深さ優先で辿る際に順序が乱されるからである．単純な解決方法は，これら多入力のゲートを辿らないか，辿るのを後にすればよい（実際，こうすると表2の人手順と対等の順になる）．しかし，多入力のかわりに，樹枝状の回路があっても同じように，順は乱れるので，他の解決方法が必要である．

一般に，多入力ゲートが多くあると，回路構造は複雑になりやすいが，そのような回路でも，例えば**図 9** に示すように，入力が一種のグループを構成しているとも考えられる構造になっている場合も多いと考えられる（実際，c 432 もこの種の構造になっている）．このような回路構造に対しては，1つのグループ内は近くに順序付けるべきであるが，単純に出力から辿ることを行うと，多入力のゲートのため，順が乱されてしまう．そこで，ここでは，深さ優先で辿る前に，入力から少し辿り，グループを先に作ることを考える．各グループ間の順序は，深さ優先で回路を辿ることで決定する．

では，どのようにグループ化すればよいのであろうか．3章の定理3から，ファンアウトが2以上の場所が1箇所のみの場合には，それを先に辿ればよかった．これを以下のように，グループ化に応用する．

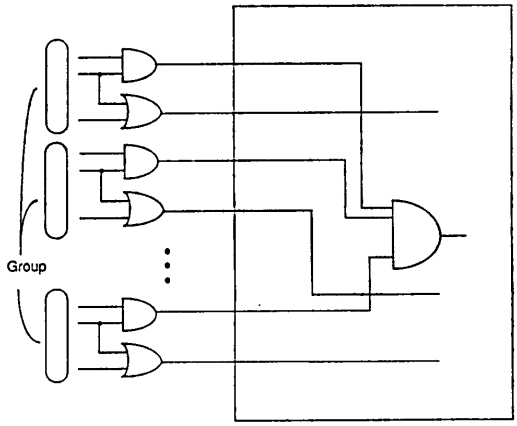


図 9 多入力ゲートをもつ回路
Fig. 9 A circuit which has many fanin gates.

（1） 回路を入力から辿り，入力以外のファンアウトが1である回路（入力以外樹枝状の回路）を切り出す．これは，入力から順に回路を辿り，ファンアウトが2以上のゲートを発見したところで切り出しをやめればよい．

（2） 切り出された回路の入力で，まだどのグループにも属していないもののうち，最もファンアウトの大きい入力を選び，その入力から出力へ回路を辿っていき，その入力に直接または，間接に接続されている出力を列挙する．次に，それらの出力各々について，回路を逆に出力から入力へ辿り，それらの出力に直接または，間接に接続されている入力を列挙する．ここで得られた入力全体を1つのグループとする．以上の操作をグループに属していない入力がなくなるまで続ける．

（3） (2)でできたグループにファンアウトが2以上の入力が1つ以下しかないグループについてのみ，図7のアルゴリズムにより順序を決め，グループとして登録する．

（4） 深さ優先に辿ることで順序を決めていくが，もし，グループ化された入力を見つけた時はグループ内の入力全体をその順とする．

上の(3)でグループに制限を付けているのは，定理3が適用できる場合のみグループ化するためである．以上をベンチマークで回路に適用し，実験した．結果を**表 5** に示す．実験では，各ベンチマーク回路に対し，最も多くの入力やゲートに関係する出力から回路を切り出したもの（1出力の回路）を用いた．

表5から分かるように，比較的小さい回路に対しては，この入力のグループ化はうまく働いており，有用

表 5 入力グループ化による変数順を用いた BDD の作成
Table 5 BDD generation by first grouping inputs.

Circuits	Without input grouping		With input grouping	
	Time	Node #	Time	Node #
test 181	1	213	1	204
test 432	195	5976	6	385
test 499	12	4661	9	3124
test 880	9	3359	9	2565
test 1355	18	4661	15	3124
test 1908	30	3076	36	3224
test 2670	66	14763	99	635
test 3540	27	3441	N.F.	100000
test 5315	1810	44387	N.F.	100000
test 7552	11	2096	N.F.	100000

N.F.: Not finished, All circuits have single output extracted from the benchmark circuits (Output which relates to the largest number of inputs is used.), Machine: SUN 3/260.

であると言える．しかし，大きな回路になってくると，この入力のグループ化は単純すぎ，ほとんどグループ化されなかったり，非常に大きなグループができたりして，うまく働かないことが分かる．また，中規模の c2670 では，出力のノード数は小さくなっているが，処理時間が増大している．これは，内部のゲートに対する BDD が大きくなっているからだと考えられ，入力のグループ化があまりうまく作用していないといえ，今後さらに検討する必要があろう．

5. おわりに

本稿では，まず，BDD の効果的な変数順決定について考えた．実験，考察の結果，人が描く回路図に現れる順が効果的であることが分かった．そこで，これに基づく変数順決定アルゴリズムを示し，評価を行った．その結果，100 入力，3,000 ゲート以上の回路の論理を BDD で表現することができた．

本稿で示した変数順決定アルゴリズムは，実回路に対し，かなりよい順を生成することが示せた．結果として，BDD は他の論理の表現手段，例えば積和形表現と比較して，より大きな回路を表すことができることが分かった．しかし，VLSI 等に使用される回路の規模は今回実験で使用したものに比べかなり大きく，実用規模の回路に対し，本稿の手法をそのまま適用できるとは言い難い．したがって，回路分割等，問題をより小さくして扱う手法は依然として重要であるが，その際にも，BDD のより大きな回路が扱えるという特徴は生かすことができると考える．

また，BDD 自体の改良，例えば，複数の BDD の共有による BDD の操作の効率化[3),4)]等も提案されており，今後はこれらも取り込み，更なる改良を行っていきたい．

謝辞 人手回路図作成に協力して頂いた横山優子氏に深謝する．

参考文献

1) Bryant, R. E.: Graph-based Algorithms for Boolean Function Manipulation, *IEEE Trans. Comput.*, Vol. C-35, No. 8, pp. 667-691 (1986).
2) Friedman, S. J. and Supowit, K. J.: Finding the Optimal Variable Ordering for Binary Decision Diagrams, *24th ACM/IEEE DAC*, pp. 348-355 (1987).
3) Cho, K. and Bryant, R. E.: Test Pattern Generation for Sequential MOS Circuits by Symbolic Fault Simulation, *26th ACM/IEEE DAC*, pp. 418-423 (1989).
4) 北嶋，高木，矢島：論理関数のグラフ表現を用いた記号シミュレーション，情報処理学会設計自動化研究会資料，87-DA-40, pp. 111-116 (1987).
5) Muroga, S., Kambayashi, Y., Lai, H. C. and Culliney, J. N.: The Transduction Method-Design of Logic Networks Based on Permissible Functions, *IEEE Trans. Comput.*, Vol. C-38, No. 10, pp. 1404-1424 (1989).
6) 藤田：トランスダクション法に基づく多段論理回路簡単化機能をもつ論理合成システムとその評価，情報処理学会論文誌，Vol. 30, No. 5, pp. 613-623 (1989).
7) Matsunaga, Y. and Fujita, M.: Multi-level Logic Optimization Using Binary Decision Diagrams, *ICCAD '89*, Santa Clara, pp. 556-559 (1989).
8) Beglez, F. and Fujiwara, H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran, *ISCAS '85*, (1985).
9) Berman, C. L.: Circuit Width, Register Allocation, and Reduced Function Graphs, IBM Research Report, No. RC 14129 (1988).
10) Fujita, M., Fujisawa, H. and Kawato, N.: Evaluations and Improvements of a Boolean Comparison Method Based on Binary Decision Diagrams, *ICCAD '88*, Santa Clara, pp. 2-5 (1988).
11) Hachtel, G. D. and Jacoby, R. M.: Verification Algorithms for VLSI Synthesis, *IEEE Trans. Computer-Aided Design*, Vol. CAD-7, No. 5, pp. 616-640 (1988).
12) Ma, H. T., Devadas, S. and Sangiovanni-Vincentelli, A. L.: Logic Verification Algorithms and Their Parallel Implementation, *24th ACM/IEEE DAC*, pp. 283-290 (1987).

(平成元年 9 月 13 日受付)
(平成 2 年 1 月 16 日採録)

藤田 昌宏（正会員）

昭和31年生．昭和55年東京大学工学部電気工学科卒業．昭和60年同大学院工学系研究科情報工学専門課程博士課程修了．工学博士．同年(株)富士通研究所入社．以来，論理設計CADシステムの研究開発に従事．並列論理プログラミング言語や並列処理技術，特にその論理設計CADに興味を持つ．

藤沢 久典（正会員）

昭和37年生．昭和60年東京大学工学部物理工学科卒業．昭和62年同大大学院工学系研究科物理工学専門課程修士課程修了．同年(株)富士通研究所に入社．以来，論理検証および論理合成CADの研究に従事．

松永 裕介（正会員）

昭和37年12月29日生．昭和60年早稲田大学理工学部電子通信学科卒業．62年同大学院(電気工学専攻)修士課程修了．同年(株)富士通研究所に入社．VLSIのCAD，特に論理合成に関する研究に従事している．電子情報通信学会会員．

角田多苗子（正会員）

昭和33年生．昭和57年東京大学教養学部基礎科学科卒業．同年(株)富士通研究所に入社．VLSIのCAD技術に関する研究に従事している．