Vol. 35 No. 5 情報処理学会論文誌 May 1994

二分決定グラフによる探索型組合せ問題の解法での組合せ的爆発抑制法

奥 乃 博†

二分決定グラフ (BDD) で組合せ問題の全解を同時に求めることを検討する．この問題での課題は，計算途中で生じる組合せ的爆発を避け，同じ計算機資源のもとでできるだけ大きな問題が解ける手法を開発することである．従来から知られていた BDD の問題点は，最終ノード数を最小にする最適変数順序を求めることである．本稿では，組合せ問題に BDD を適用する場合には最大ノード数が重要であること，および，それが変数順序だけでなく制約組合せ順序の影響を受けることを指摘する．次に，最大ノード数を最小にする制約順序と変数順序の最適解を見つけるアルゴリズム CCVO を提案する．さらに，最大ノード数が大き過ぎて計算ができない場合には，オンライン版分割統治法を使用する解法を提案し，その分割ヒューリスティックを提案する．これらの手法により，12-Queens を 128 Mbyte 主記憶で解くことができ，また，分割統治法により 13-Queens と 4 次の魔法陣を解くことができた．二分決定グラフでは途中結果がすべて保存されているので，オンライン型分割統治法による解法は，オフライン型分割統治法よりも高速である．

Reducing Combinatorial Explosions in Solving Search-Type Combinatorial Problems with Binary Decision Diagrams

Hiroshi G. Okuno†

We consider finding the complete solutions of combinatorial problems simultaneously by BDDs (Binary Decision Diagrams) with the aim of developing a methodology to avoid combinatorial explosions in building the BDDs. Since the order of variables affects the number of nodes in the BDD, the known problem is to find an optimal variable order that minimizes the number of nodes. However, the constraint order and its associated variable order are important in reducing combinatorial explosions, which has not been explicitly pointed out so far. We propose an algorithm called Correlation-based Constraint and Variable Ordering (CCVO), which computes a constraint and variable order that reduces the number of maximum nodes. We also propose the on-line divide-and-conquer method and some heuristics for decomposing the problem into a set of subproblems. With 128 Mbytes of memory, the 12-Queen problem is solved by the optimal constraint and variable order obtained by CCVO. The on-line divide-and-conquer is also used to solve the 13-Queen problem and the 4-order Magic Square. The on-line divide-and-conquer is much faster than the off-line version due to the compact representation of BDD. Since these results with BDDs have not been reported before, the proposed methodology makes it possible to solve larger problems with the same computing resources.

1. はじめに

最近，ブール関数がコンパクトに表現でき，かつ，効率的な処理が可能な手法として二分決定グラフ (Binary Decision Diagram, 以下 BDD と略す)[1),3)] が注目を浴びている．論理関数に対する BDD は，変数順序を固定するとユニークに定まるので，論理関数のカノニカル表現となる．その結果，VLSI CAD の分野では，BDD は標準的に使用され，回路設計，検証，テスト生成などに応用されている[5)]．BDD のサイズは変数順序によって大きく変化するので，大規模な回路を扱うためには，BDD のサイズを最小にする最適な変数順序を見つけることが重要な課題である．しかし，最適変数順序を求めることは NP 完全の問題であり，さまざまな回路に対して最適あるいは準最適な変数順序を見つけるアルゴリズムやヒューリスティックが提案されている．

† 日本電信電話 (株) 基礎研究所
NTT Basic Research Laboratories, Nippon Telegraph and Telephone Corporation

BDD は回路だけではなく，組合せ問題や最適化問題，さらには人工知能にも適用されている[11),12),15),16)]．これらの応用では，回路とは異なった問題が生ずる．組合せ問題や最適化問題では探索空間で解の占める割合が極めて小さい．たとえば，組合せ問題の例として魔法陣を解く問題を考えよう．n 次の魔法陣とは $n\times n$ の盤面に1から n^2 の数を並べたときに，縦・横・斜めに並んだ数の和がすべて等しいようなものである．3次の魔法陣の対称解を除いた解は

2	9	4
7	5	3
6	1	8

だけである．4次の魔法陣では16個の数を使用するので，探索空間は 16!，約 2.1×10^{13} となる．対称解を排除すると探索空間は1/8に縮小される．しかし，4次の魔法陣の解は880種しかないので，探索空間で解の占める割合は極めて小さい[13)]．魔法陣を BDD で解こうとすると，計算途中で BDD が膨大になって BDD が構築できず，解が求まらないという組合せ的爆発が発生しやすい．本稿で取り組む組合せ的爆発は，最終結果の BDD が主記憶に収まるにもかかわらず，計算途中で BDD が膨大になり解集合が表現できない場合である．

組合せ問題に BDD を応用するときに生ずるような組合せ的爆発は，回路設計において回路を BDD で表現しようとする場合には一般には発生しない．というのは，BDD を構築しようとする対象の回路はすでに何らかの最適化が施されているので，冗長な部分があまり残っていないことが多いからである．さらに，回路図にしたがって BDD を構築していけば，回路全体が BDD で表現可能である場合には，計算途中で BDD が爆発することはほとんどない[4)]．しかし，組合せ問題では，変数順序だけでなく，BDD を構築するときに，問題を記述する制約条件の論理積を取っていく順序によっても組合せ的爆発が発生する．また，回路の検証でも，同様の問題が発生することがある．

組合せ問題は，従来はバックトラック型プログラムで解かれてきた．バックトラック型プログラムは，計算途中で遭遇した選択点の情報をスタック等に蓄えておけば十分であり，それ以外の中間状態を保持する必要がないので，高速に解を数え上げることができる．しかし，バックトラック型プログラムは選択木の縦型探索であるので，得られた解の構造を調べたり，解同士の比較を行ったりすることは簡単ではない．さらに，バックトラックにより以前の選択点に後ろ戻りするときにすでに計算した結果を捨ててしまうので，同じ計算を何度も繰り返すという冗長計算を行う可能性もある．また，バックトラック型プログラムでは途中結果が保存されていないので，得られた解に対してさらに制約条件を満たす解を求めたい場合には，プログラムを変更し，再実行しなければならないことが多い．

一方，全解を同時に求める横型探索は，上記のような高機能な処理が容易に実現できる．横型探索では中間状態が保存されているので，システムはすでにその探索を行ったかをチェックすれば，冗長計算を避けることができる．しかし，中間状態をうまく保存しないとデータがメモリに入り切らずオーバーフローして計算が続行できなくなることはいうまでもない．BDD は論理関数のコンパクトな表現であるので，中間状態を効率よく保存しておくことができ，横型探索に適していると期待される．

組合せ問題は，一般に再帰関係型と探索型に大別することができる．

（1）再帰関係型組合せ問題

解が再帰関係（recurrence relation）で記述できるような組合せ問題である．たとえば，n 個の中から j 個のものを選ぶ方法を列挙する問題を考えよう．論理関数 $F_{i,j}$ を j 個の変数 $x_1, x_2, \cdots, x_j$ から i 個を選ぶ方法を表現する論理式とすると，$F_{i,j}$ は再帰関係

$$\begin{cases} F_{0,0}=1 & \\ F_{0,j}=\neg x_j\wedge F_{0,j-1} & (j\geq1) \\ F_{j,j}=x_j\wedge F_{j-1,j-1} & (j\geq1) \\ F_{i,j}=x_j\wedge F_{i-1,j-1}\vee\neg x_j\wedge F_{i,j-1} & (1<i<j) \end{cases}$$

によって表現することができる（∧は∨よりも優先順位が高い）．

再帰関係で表現できる組合せ問題の BDD は，i, j を1から順次増やして再帰関係に従った論理演算を繰り返せば求まる．BDD を構築するときにはむだなノードが生成されないので，システムで利用可能なメモリ量にほぼ相当するサイズの BDD まで構築することができる．また，BDD の構築も極めて高速にできる．実際，仙波はさまざまな再帰関係型組合せ問題の解を BDD を用い

て高速に求めている[11),12)].

再帰関係型組合せ問題は組合せ的爆発が発生しないので，本稿では検討の対象とはしない．

（2）**探索型組合せ問題**

与えられた制約条件から問題空間を探索し，解を見つけるような組合せ問題である．たとえば，巡回セールスマン問題，N-Queens 問題，魔法陣などがある．全探索空間に対して解の占める密度が極端に小さい．たとえば，4次の魔法陣では解の密度は 3.3×10^{-10} である．

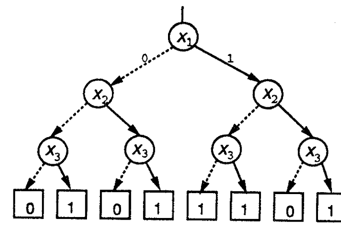
本稿では，探索型組合せ問題を解く場合の組合せ的爆発をできるだけ避けるための方策について，さまざまな側面から議論する．

本稿の構成を以下に示す．2章で，二分決定グラフ（BDD）について説明し，BDD 使用上の問題点を指摘する．3章で，組合せ問題に BDD を応用して，変数順序と制約順序の問題点を指摘する．4章で，制約順序と変数順序を決定するアルゴリズムを提案する．5章で，組合せ的爆発のためにそのままでは解けない問題に分割統治法を適用するための分割ヒューリスティックを述べる．6章で，実験結果を示し，評価を行い，7章で，まとめを行う．

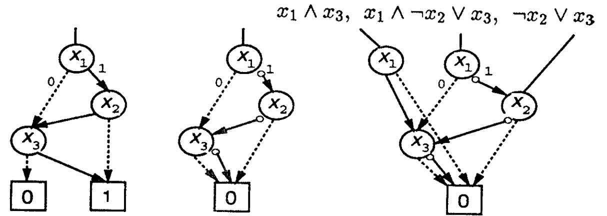
2. 二分決定グラフ（BDD）

2.1 論理関数表現法

BDD は，Akers が提案し[1)]，Bryant が効率の良い BDD 処理アルゴリズムを考案した[3)] コンパクトな論理関数の表現法である．一般に，論理関数は変数の取る値によって木構造で表現できる．たとえば，論理式 $x_1\wedge\neg x_2\vee x_3$ の値は，変数をノードとし，各変数の 0, 1 の値を枝とすると，木表現になる（**図1**の(a)）．この木表現に変数順序 x_1, x_2, x_3（x_1 が最大順位，BDD で最上位のノードとなる）を導入し，かつ，終端ノード，重複ノードを共有化し，冗長ノードを削除することによって既約化した木構造を**既約順序つき二分決定グラフ**（Reduced Ordered BDD, ROBDD）と呼ぶ（図1の(b)）[3)]．ROBDD では，さまざまな属性を有した枝を導入し，ノード数を減らす工夫がされている．その一つは，否定枝である．図1(c)の小さな丸は否定枝を示し，否定枝の値は0と1が交換される．これによって，1のノードは不要となる．湊は，



(a) $x_1\wedge\neg x_2\vee x_3$ の木表現
(a) Tree representation



(b) ROBDD での表現 (c) 否定枝の導入 (d) SBDD での表現
(b) ROBDD (c) Negated nodes (d) SBDD

図1 論理式の木表現と ROBDD, SBDD
Fig. 1 Tree representation, ROBDD and SBDD of a logic formula.

ROBDD を論理関数間で共有できるようにした**共有二分決定グラフ**（Shared BDD, 以下 SBDD と略す）を提案している[7)]．図1(d)には，3つの論理関数 $x_1\wedge x_3$, $x_1\wedge\neg x_2\vee x_3$, $\neg x_2\vee x_3$ で共有される SBDD を示す．

以下では，ROBDD や SBDD をすべて BDD と略すことにする．

BDD の主な特徴は

（1）変数順序を固定すると論理関数の標準形となる，

（2）多くの実用的な関数を比較的コンパクトなノード数で表現できる，

（3）論理演算に対する演算ノードが数に比例する時間で行える，

とまとめることができる．これらの特徴によって，BDD は論理設計や検証，テスト生成，記号シミュレーションなどの多岐にわたる応用で使用されている．

BDD は，最初に木表現（図1の(a)）を作成し，次にそれを既約化することにより構築されるのではない．論理式を順番に適用することによって逐次的に構築される．論理関数間の論理演算は，それぞれの論理関数に対応する BDD 同士の *apply* 演算によって実現される．apply 演算は，再帰呼び出しによって次のように実現される．

apply (bdd 1, bdd 2, operation)

= {
- bdd 1 の根の変数順位＝bdd 2 の根の変数順位ならば，各々の枝を用いて operation を実行し，0枝と1枝を求め，ノードを作成．
- bdd 1 の根の変数順位＞bdd 2 の根の変数順位ならば，apply (bdd 1 の 0 枝, bdd 2, operation) と apply (bdd 1 の 1枝, bdd 2, operation) をそれぞれ0枝，1枝とするノードを作成．
- 上記以外ならば，apply (bdd 2, bdd 1, operation) を実行．
}

BDD に対して論理演算を実行する部分は，二つの BDD に対応するシャノン展開に対して論理演算を施すことである．たとえば，関数 f と g とのシャノン展開を

$$f=\neg x\wedge f|_{x=0}\vee x\wedge f|_{x=1}$$

$$g=\neg x\wedge g|_{x=0}\vee x\wedge g|_{x=1}$$

とすると，f と g との論理積は

$$f\wedge g=\neg x\wedge(f|_{x=0}\wedge g|_{x=0})\vee x\wedge(f|_{x=1}\wedge g|_{x=1})$$

で計算される．

apply 演算を効率よく実行するために BDD 処理系では2種類の工夫が行われている[9]．

- BDD ノードのハッシュ表による管理
- apply 演算の結果のハッシュ表によるキャッシュ

後者のキャッシュはうまく機能すれば apply 演算で同じ計算を省略できるので，冗長計算が省け，効率よく BDD を構築することができる．

表 1 BEM-II の演算
Table 1 BEM-II operators.

演　算	意　味	演　算	意　味
! *expr*	論理否定	~*expr*	ビット反転
expr1 * *expr2*	算術積	*expr1*+*expr2*	算術和
expr1−*expr2*	算術差	*expr1*≪*expr2*	算術左シフト
expr1≫*expr2*	算術右シフト	*expr1*>*expr2*	算術不等号条件
expr1<*expr2*	算術不等号条件	*expr1*>=*expr2*	算術不等号条件
expr1<=*expr2*	算術不等号条件	*expr1*==*expr2*	算術等号条件
expr1!=*expr2*	算術非等号条件	*expr1*&*expr2*	論理積（ビット/論理的）
expr1 ↑ *expr2*	排他的論理和	*expr1* | *expr2*	論理和

表 2 BEM-II のコマンド
Table 2 BEM-II commands.

コマンド	意　味
symbol *variable...*	変数順序の指定
FUNCTION=*formula*	関数記号の定義
? [/option] (*formula*|*function*)	出力（/option は, /size, /count, /mincover など）

2.2 算術論理式計算システム BEM-II

BDD を組合せ問題に適用するときに，組合せ問題をどのように記述するのかという課題がある．BDD のライブラリを用いてC言語等の低レベルで書くのか，コマンドインタープリタを持つシステムで記述するのか，あるいは，算術式まで許すコマンドインタープリタを持つシステム（例，BEM-II）で書くのか，など，さまざまな代替案が考えられる．

本稿では，BDD システムとして湊が開発した算術論理式計算システム BEM-II (Boolean Expression Manipulator-II）を使用した[8]．BEM-II は，SBDD をもとにしたシステムであり，入力として論理式だけでなく，算術演算を含む論理式が許される．**表1**に BEM-II で提供されている演算子をまとめる．算術演算を使用すると，変数 x1, x2, x3, x4 のどれか一つだけ選択するという択一型制約条件は

x 1+x 2+x 3+x 4==1

と書くことができる．この式は，BEM-II により

x1 & !x2 & !x3 & !x4 | !x1 & x2 & !x3 & !x4 |
!x1 & !x2 & x3 & !x4 | !x1 & !x2 & !x3 & x4

と展開される．BEM-II でのコマンドを**表 2** に示す．BEM-II で書いた魔法陣のプログラムを**図 2** に示す．

3. BDD による組合せ問題解法での検討課題

BDD によって組合せ問題に応用するときの課題として以下の3点がある：

（1）所与の問題のコーディング方法
（2）変数順序
（3）制約条件を表現する論理式適用の順序—**制約順序**と呼ぶ．

具体的な課題を明らかにするために，図2に示した魔法陣のプログラムを例に取り，説明をしよう．

3.1 組合せ問題のコーディング方法

魔法陣のマス目に置かれる数は，ビットベクトルでコーディングする（代替案として，マス目に置かれる数を2進数でコーディングする方法もある）．

- $P_{i,j}$—i 行 j 列のマス目に置かれる数
- $x_{i,j,k}$—i 行 j 列のマス目に

```
# 変数順序の宣言
symbol x11 x12 x13 x14 x15 x16 x17 x18 x19
symbol x21 x22 x23 x24 x25 x26 x27 x28 x29
. . .
symbol x91 x92 x93 x94 x95 x96 x97 x98 x99

# マス目に置く数の定義
P11 =  x11*1 | x12*2 | x13*3 | x14*4 | x15*5 | x16*6 | x17*7 | x18*8 | x19*9 ..... (P1)
P12 =  x21*1 | x22*2 | x23*3 | x24*4 | x25*5 | x26*6 | x27*7 | x28*8 | x29*9 ..... (P2)
. . .
P33 =  x91*1 | x92*2 | x93*3 | x94*4 | x95*5 | x96*6 | x97*7 | x98*8 | x99*9 ..... (P9)

# 一つのマス目には一つの数しか置けない
C =     (x11 + x12 + x13 + x14 + x15 + x16 + x17 + x18 + x19 == 1) ........ C =     (C1)
C = C & (x21 + x22 + x23 + x24 + x25 + x26 + x27 + x28 + x29 == 1) ........ C = C & (C2)
. . .
C = C & (x91 + x92 + x93 + x94 + x95 + x96 + x97 + x98 + x99 == 1) ........ C = C & (C9)

# 一つの数は盤の中に一つしか置けない
C = C & (x11 + x21 + x31 + x41 + x51 + x61 + x71 + x81 + x91 == 1) ........ C = C & (D1)
C = C & (x12 + x22 + x32 + x42 + x52 + x62 + x72 + x82 + x92 == 1) ........ C = C & (D2)
. . .
C = C & (x19 + x29 + x39 + x49 + x59 + x69 + x79 + x89 + x99 == 1) ........ C = C & (D9)

C = C & (P11 + P12 + P13 == 15)    # 各行の和が15 ............................ C = C & (S1)
C = C & (P21 + P22 + P23 == 15) & (P31 + P32 + P33 == 15) ........... C = C & (S2) & (S3)

C = C & (P11 + P21 + P31 == 15)    # 各列の和が15 ............................ C = C & (T1)
C = C & (P12 + P22 + P32 == 15) & (P13 + P23 + P33 == 15) .......... C = C & (T2) & (T3)

C = C & (P11 + P22 + P33 == 15)    # 各対角線の和が15 ......................... C = C & (X)
C = C & (P13 + P22 + P31 == 15)    ............................................. C = C & (Y)

# 対称解の除去
C = C & (P11 < P13) & (P11 < P31) ....................................... C = C & (K) & (L)
C = C & (P13 < P31) & (P11 < P33) ....................................... C = C & (M) & (N)

? /size C                          # BDD のサイズを出力
? /count C                         # BDD の解の個数を出力
? C                                # BDD の積和論理形を出力
```

図 2 BEM-II による3次の魔法陣プログラム
Fig. 2 3-Order magic square program with BEM-II.

数 k が置かれているかを示す変数(ビットベクトルの要素).

$$x_{i,j,k}=\begin{cases}1 & (i\text{ 行 }j\text{ 列のマス目に数 }k\text{ が置かれているとき})\\ 0 & (i\text{ 行 }j\text{ 列のマス目に数 }k\text{ が置かれていないとき})\end{cases}$$

- 各マス目に置かれる数 $P_{i,j}$ はビットベクトル要素とそのビットベクトル要素が表す数値の積の論理和である(図2の(P1)〜(P9)).

$$P_{i,j}=x_{i,j,1}*1\vee x_{i,j,2}*2\vee\cdots\vee x_{i,j,n^2}*n^2$$

制約条件は以下のようにコーディングする.

- 各マス目にはどれか一つの数が置かれるという択一型制約条件は,ビットベクトル要素が一つだけ1になるという算術論理式で表現する(図2の(C1)〜(C9)).
- 各数がどれかのマス目に置かれるという択一型制約条件は,同じ数値を表すビットベクトル要素の集合でどれか一つだけが1になるという算術論理式で表現する(図2の(D1)〜(D9)).
- 各行,各列,各対角線の数の和は等しいという制約条件は,それぞれの集合に属するマス目の和が総和の $n(n^2+1)/2$ に等しいという算術論理式で表現する(図2の(S1)〜(S3),(T1)〜(T3),(X),(Y)).
- 対称解を除去する制約条件は,四隅に置かれる数の不等式で表現する(図2の(K)〜(N)).

図2に示した3次の魔法陣を解くプログラムが上記のコーディングを使用していることは容易にわかるであろう.

3.2 変数順序と制約順序

BDD の変数順序によって,(1)最終的な BDD のノード数(サイズ),(2)BDD を計算する途中に必要なノード数,が大きく変わることが知られている.最終ノード数を最小にする最適変数順序を求める問題は,計算量の観点からは困難な問題であることがわかっている.論理設計の分野では,最適変数順序や準最適変数順序を求めるさまざまな方法が提案されている[4].その方法は,(一般の問題に共通のものではなく)

固有の問題に即した制限，規則を使うものである．

計算途中で必要なノード数は変数順序だけでなく，BDD 間の組合せの順序によっても大きく変化する．たとえば，3次の魔法陣の解法では，同じ変数順序であっても制約順序によって最大ノード数が2桁も変わる場合がある．組合せ問題をコーディングした論理式からそのまま BDD を生成すると容易に組合せ的爆発が生ずる．問題に応じた制約順序を求める BDD を応用する上で重要な課題である．

4. 制約順序・変数順序決定アルゴリズム

4.1 CCVO アルゴリズムの記述

本節では，ほぼ最適な制約順序と変数順序の組を求めるアルゴリズム **CCVO** (Correlation-based Constraint and Variable Ordering) を説明する．まず，定義を与え，次に，設計方針を述べる．

- **制約グループ**：制約条件をグループ分けしたものである．たとえば，魔法陣プログラムでの制約グループは盤の制約，数の制約，列の和の制約などである．
- **制約間の相関関係表**：以下で定義される制約条件 C_1 と C_2 に対する**相関度**の表である．

$$correlation\ (C_1, C_2) = \frac{C_2 \text{ のうち } C_1 \text{ に現れる変数の個数}}{C_2 \text{ に現れる変数の個数}}$$

さらに，制約条件 C_2 と制約グループ $\boldsymbol{C}$ との相関関係を定義する．C_2 の変数がすべてどれかの $C_1 \in \boldsymbol{C}$ に現れるとき，制約条件 C_2 は制約グループ $\boldsymbol{C}$ に対して**完全相関**にあるといい，C_2 の変数のうちどの $C_1 \in \boldsymbol{C}$ にも現れない変数があるときには，**不完全相関**にあるという．制約グループ間の相関関係についても同様に定義する．

3次の魔法陣プログラムにおける相関関係表を**表 3** に示す．制約グループ $\boldsymbol{C}$ (制約条件 C 1～C 9) はすべての制約グループに対して完全相関にある．制約グループ $\boldsymbol{D}$ (制約条件 D 1～D 9) は $\boldsymbol{S, T, X, Y, K, L, M, N}$ に対して不完全相関にある．$\boldsymbol{C}$ は $\boldsymbol{S, T}$ の制約グループによってそれぞれの3つのサブグループに分割される．一方，制約グループ $\boldsymbol{D}$ は他のどの制約条件によっても一切サブグループには分割されない．サブグループへの分割は相関度が1の制約条件から見つける．

次に，制約順序と変数順序を求めるための基本方針を示す．それは以下のようにまとめることができる．

- 択一型制約条件を優先する．
- 他の制約条件，制約グループと完全相関な制約グループを優先する．
- ノード数の少ない制約条件を優先する．
- サブグループに分割される制約グループは，サブグループと分割する制約条件を組で考え，その組を優先する．
- すでに実行された制約条件と相関関係の高い制約条件のうち，変数の個数の少ないものを優先する．
- 制約条件の順序が決まれば，それに含まれる変数について変数順序を決める．

これで準備が終了した．次に，制約順序と変数順序とを求めるアルゴリズム CCVO を示す．

(1) すべての制約条件を制約条件プールに入れる．

(2) 制約条件を分類する：択一型制約グループを集め，完全相関であるものを *Comp* に並べ，不完全相関のものを *Incomp* に並べる．いずれも現れる変数の少ないものを前に置く．サブグループに分割可能な択一型制約グループについては，同じノード数の他のグループより前に並べる．このとき，サブグループを分割する制約条件を付加する．分割可能な組合せが複数通りある場合には，どれか一つを選択する．
　残りの制約条件を変数の個数の少ない順に *Other* に並べる．また，どれかの制約グループとの相関度の和が1となった制約条件を保持する *Ready* を用意する．さらに，変数順序を保持する *Var* と，制約順序を保持する *Order* を用意する．

(3) *Comp* が空でない限り，以下のことを繰り返す．

　(a) *Comp* から先頭の（サブ）グループを取り出す．
　　グループに含まれる制約条件から *Var* に含まれていない変数を抽出し，それを出現順に *Var* に付加する．グループ内の制約条件について以下のことを繰り返す．

　　i. 一つの制約条件 *C* を *Order* に入れ，制約条件プールから抜く．

　　ii. *Incomp*, *Other* のうち，それまでに制約順序が定まったことによって，どれかの制約グループとの相関度の和が1になったような制約

表 3 3次の魔法陣での制約条件間の相関
Table 3 Correlation between constraints in 3-order magic square.

C_1	C									D	S			T			X	Y	K	L	M	N
C_2	1	2	3	4	5	6	7	8	9	1 ～ 9	1	2	3	1	2	3						
C1										$\frac{1}{9}$ … $\frac{1}{9}$	1	0	0	1	0	0	1	0	1	1	0	1
C2										$\frac{1}{9}$ … $\frac{1}{9}$	1	0	0	0	1	0	0	0	0	0	0	0
C3										$\frac{1}{9}$ … $\frac{1}{9}$	1	0	0	0	0	1	0	1	1	0	1	0
C4										$\frac{1}{9}$ … $\frac{1}{9}$	0	1	0	1	0	0	0	0	0	0	0	0
C5					**1**					$\frac{1}{9}$ … $\frac{1}{9}$	0	1	0	0	1	0	1	1	0	0	0	0
C6										$\frac{1}{9}$ … $\frac{1}{9}$	0	1	0	0	0	1	0	0	0	0	0	0
C7										$\frac{1}{9}$ … $\frac{1}{9}$	0	0	1	1	0	0	0	1	0	0	1	0
C8										$\frac{1}{9}$ … $\frac{1}{9}$	0	0	1	0	1	0	0	0	0	0	0	0
C9										$\frac{1}{9}$ … $\frac{1}{9}$	0	0	1	0	0	1	1	0	0	0	0	1
D1	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$		$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$
⋮	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	**1**	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$
D9	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{9}$		$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$	$\mathbf{\frac{2}{9}}$
S1	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	0	0	0	$\frac{1}{9}$ … $\frac{1}{9}$				$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{2}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$
S2	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	0	0	0	$\frac{1}{9}$ … $\frac{1}{9}$		**1**		$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	**0**	**0**	**0**	**0**
S3	0	0	0	0	0	0	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{9}$ … $\frac{1}{9}$				$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	**0**	**0**	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$
T1	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	0	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$				$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{2}{3}}$	$\mathbf{\frac{2}{3}}$	$\mathbf{\frac{1}{3}}$
T2	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$		**1**		$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	**0**	**0**	**0**	**0**
T3	0	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	0	0	$\frac{1}{3}$	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$				$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	0	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$
X	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	$\mathbf{\frac{1}{3}}$	0	$\mathbf{\frac{2}{3}}$
Y	0	0	$\frac{1}{3}$	0	$\frac{1}{3}$	0	$\frac{1}{3}$	0	0	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	$\mathbf{\frac{1}{3}}$	1	$\mathbf{\frac{1}{3}}$	0	$\mathbf{\frac{1}{3}}$	**0**
K	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	0	0	0	$\frac{1}{9}$ … $\frac{1}{9}$	1	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$	1	$\mathbf{\frac{1}{2}}$	0	$\mathbf{\frac{1}{2}}$
L	$\frac{1}{2}$	0	0	0	0	0	$\frac{1}{2}$	0	0	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1	0	0	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$	1	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$
M	0	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	0	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	**0**	1	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$	1	**0**
N	$\frac{1}{2}$	0	0	0	0	0	0	0	$\frac{1}{2}$	$\frac{1}{9}$ … $\frac{1}{9}$	$\frac{1}{2}$	0	$\frac{1}{2}$	$\frac{1}{2}$	0	$\frac{1}{2}$	1	0	$\mathbf{\frac{1}{2}}$	$\mathbf{\frac{1}{2}}$	0	1

ただし，**1** は対角線成分のみが1である部分行列を示し，**ボールド体**は C_2 が $\{C_1, \ldots\}$ に対して不完全相関にあることを示す.

条件があった場合には，それらをすべて *Ready* の最後に付ける.

(b) サブグループに分割した制約条件がある場合には，それを *Order* に入れ，*Other* と制約条件プールから抜く.

(4) *Ready* のうち *Incomp* に含まれるものがあれば，以下のことを繰り返す.
先頭から一つ取り出し，*Order* に入れ，制約条件プールから抜く．まだ，順序の定まっていない変数があれば，*Var* の最後に付加する.
この制約順序が定まったことによって，どれかの制約グループとの相関度の和が1になったような制約条件があった場合には，それらをすべて *Ready* の最後に付加する.

(5) *Incomp* から順に制約条件 C を取り出し，*Order* に入れ，制約条件プールから抜く．まだ順序の定まっていない変数があれば，*Var* の最後に付加する.
Other のうち，それまでに制約順序が定まったことによって，どれかの制約グループとの相関度の和が1になったような制約条件があ

った場合には，*Ready* の最後に付加する．

(6) *Ready* から一つずつ取って，*Order* に入れ *Other* と制約条件プールから抜く．まだ変数順序が定まっていない変数が現れたら，*Var* の最後に付加する．
Other のうち，それまでに制約順序が定まったことによって，どれかの制約グループとの相関度の和が1になったような制約条件があった場合には，*Ready* の最後に付加する．

(7) *Other* について以下のことを繰り返す．
(a) *Other* を並べ換える．各制約条件のソートのキーは，すでに *Order* に入っている制約条件（制約順序が定まっている）との相関度のうちの最大値を使用する．
(b) *Other* の先頭要素を一つ取り出し，それを *Order* に入れ，制約条件プールから抜く．まだ，変数順序が定まっていない変数が現れたら，*Var* の最後に付加する．

3次の魔法陣プログラムに対しては，以下のような結果が得られる．

- 制約順序：C1 C2 C3 S1 C4 C5 C6 S2 C7 C8 C9 S3 D1 D2 D3 D4 D5 D6 D7 D8 D9 K L M T1 Y T2 T3 X N
- 変数順序：x11 x12…x19 x21 x22…x29…x91 x92…x99

ただし，{D1, …, D9}，{L, M}，{T1, Y}，{T3, X, N}，および，{x11, …, x19}, …, {x91, …, x99} はそれぞれそれらの中で順序が入れ替わってもよい．また，***C*** のサブグループの順序およびサブグループ内の順序は入れ替わってもよいが，変数順序もそれに対応して変化する．

4.2 CCVO アルゴリズムの意味と拡張

探索型組合せ問題を解くシステムは一般に(1)組合せ生成器，と(2)制約フィルターという2つのサブシステムから構成される．択一型制約条件は組合せ生成器と考えることができる．本稿で提案する CCVO アルゴリズムでは，*Comp*, *Incomp* が組合せ生成器，*Other* が制約フィルターに相当する．制約フィルターの制約順序を決定するときに，それが持つ変数の個数を考慮するのは，変数の個数からその制約条件の BDD のノード数が推測でき，初期段階で大きな BDD 同士に対して apply 演算を施した結果の BDD のノード数が爆発する可能性を少なくするためである．たとえば，4次（3次）の魔法陣では，後述する盤優先変数順序のもとで，択一型制約条件の BDD☆ は 30(16) ノードしか持たないのに対して，K の BDD は 447(322) ノード，L, M, N の BDD はそれぞれ 377(322) ノード，Si, Ti, X, Y の BDD はそれぞれ 7088(319) である．したがって，これらの制約条件は極力後回しにした方がよい．

ここで，制約条件の BDD のノード数を考慮した CCVO の変形を考えることができる．準備として，すべての制約条件の BDD のノード数を求めておく．次に，CCVO アルゴリズムの(3)-(a)-i．の次に以下のステップを挿入する．

Other のどの制約条件の BDD のノード数が *Comp*, *Incomp* の制約条件のそれよりも所与の係数倍以上に大きいときには，以下のことを行う．
Incomp のうち，今，制約順序を定めた制約条件 (*C*) と非零相関度を持つ制約条件のうち最大のものを一つ取り出し，*Order* に入れ，制約条件プールから抜く．

魔法陣の場合にはその係数を1桁とすると，この変形アルゴリズムによって，完全相関の択一型制約条件を実行すると次にそれに関連した不完全相関の択一型制約条件を一つ実行するという交互型制約順序が得られる．ただし，ノード数の比の係数をどのように定めるかは問題ごとに見つけなければならない．

さらに，提案した CCVO アルゴリズムの別の変形として，どれかの制約条件グループに対する相関度の和が1になった制約条件をすぐに実行するように *Order* に入れる**即実行型アルゴリズム**が考えられる．即実行型アルゴリズムでは制約フィルターが優先される傾向にあるので，多くの問題ではあまりよい制約順序とは思われない．

5. 分割統治法による大規模問題への挑戦

5.1 分割統治法

前節で提案した CCVO アルゴリズムだけでは中間ノードの爆発が抑え切れず解が求まらない場合がどうしても残る．本章では，分割統治法 (Divide-and-Conquer) によって，部分問題を解き，それらの解を統合する方法を検討する．本稿で提案する手法は，**オンライン型分割統治法** (On-line Divide-and-Conquer) と

☆ n 個から一つを選択する択一型制約条件の BDD のノード数は $2n-2$ である．

呼ぶ．すなわち，中間結果はBDDとして内部に保持しておき，すべての部分問題の解が求まった時点でそれらの論理和を取る．この場合，すでに構築されたBDDを利用して部分問題を解くことができるので，部分問題を個別に解くよりも効率がよくなると期待される．もちろん，部分問題の解を保持しておくと空間の使用効率が悪くなるので，システムの自由領域が少なくなってガーベッジコレクションの回数が増えて，効率が逆に低下することもあろう．

オンライン型分割統治法に対して，オフライン型分割統治法も考えられよう．すなわち，部分問題の解を一たんファイルに書き込み，すべての部分問題の解が求まった時点で，ファイルから読み込んで最終的な解のBDDを構築する．オフライン型では，すでに解いた部分問題の情報は使用できない．しかし，部分問題を解くだけでシステムの領域が目一杯になるときには，有効な手法である．

5.2 部分問題の分割法

与えられた問題を部分問題に分割する方法には，(1)論理和で表現される制約条件を分割する方法と，(2)特定の変数に具体値を代入する方法，の2種類がある．

また，分割された部分問題の大きさを分割の**粒度**(granularity)と呼ぶ．組合せ問題での最も細かい粒度は場合分けを完全に行ったときに得られる部分問題である．たとえば，魔法陣での最も細かい粒度はすべてのマス目にどれかの数を置いた場合の部分問題であり，N-Queens 問題では各行に一つずつ Queen を置いた場合の部分問題である．

オフライン型では，粒度を細かくすると，システムのリスタート回数が増え，また，出力ファイル数の増加によってファイルのオープン／クローズ回数が増え，それによるオーバヘッドが大きくなるので，できるだけ粒度を粗くする方がよいと思われる．一方，オンライン型では，一たん計算した結果はその後の計算で利用できるので，粒度によるオーバヘッドはそれほど大きくないと予想される．

分割の粒度をプログラム実行前に決定する静的分割と実行時に決定する動的分割が考えられる．後者は一種の動的プログラミングである．本稿では，静的分割方式を取り上げ，分割すべき候補を決定するヒューリスティックを提案する．

(1) 論理和で表現される制約条件があれば，それを分割の対象とする．
(2) 完全相関である択一型制約条件があれば，変数順序の最も上位のものを含むもののうち，ノード数の少ないものを分割の対象とする．
(3) 不完全相関択一型制約条件があれば，変数順序の最も上位のものを含むもののうち，ノード数の少ないものを分割の対象とする．
(4) 論理和か算術和で定義された出力関数があれば，それを分割の対象とする（たとえば，図2の P_{ij})．
(5) 入力変数のうち，最上位のものを分割の対象とする（たとえば，図2の x11)．

変数順位が上位のものから分割の候補として選択しているのは，BDD に対する apply 演算で展開をできるだけ少なくするためである．

候補が定まったら，その候補の形によって，以下のように分割方法を決める．

- 論理和 $f_1 \vee f_2 \vee \cdots \vee f_n$ のとき：(例，図2の(P1)～(P9))．

  制約順序・変数順序決定アルゴリズム CCVO で求めたプログラムを P とする．$\boldsymbol{P}$ を途中まではそれほどノードが大きくならない部分に分け，$\boldsymbol{P}_1, \boldsymbol{P}_2$ とする（これは試行錯誤による)．分割統治法により，

  $C=\boldsymbol{P}_1$
  $S_1=C \wedge f_1$; $S_1=S_1 \wedge \boldsymbol{P}_2$
  $S_2=C \wedge f_2$; $S_2=S_2 \wedge \boldsymbol{P}_2$
  …
  $S_n=C \wedge f_n$; $S_n=S_n \wedge \boldsymbol{P}_2$
  $G=S_1 \vee S_2 \vee \cdots \vee S_n$

  で計算する．粒度が大きくできるときには，f_i は論理和の形であってもよい．

- 択一型制約条件 $f_1+f_2+\cdots+f_n==1$ のとき：(例，図2の(C1)～(C9)，(D1)～(D9))

  同様に，

  $C=\boldsymbol{P}_1$
  $S_1=C \wedge (f_1==1)$; $S_1=S_1 \wedge \boldsymbol{P}_2$
  $S_2=C \wedge (f_2==1)$; $S_2=S_2 \wedge \boldsymbol{P}_2$
  …
  $S_n=C \wedge (f_n==1)$; $S_n=S_n \wedge \boldsymbol{P}_2$
  $G=S_1 \vee S_2 \vee \cdots \vee S_n$

  で計算する．粒度が大きくできるときには，f_i は算術和の形であってもよい．

- 出力関数のときは，上記の2つのどちらかになろう．
- 入力変数 x を分割の対象とするときには，

  $C=\boldsymbol{P}_1$

$S_1 = C \wedge x$; $S_1 = S_1 \wedge \boldsymbol{P}_2$

$S_2 = C \wedge \neg x$; $S_2 = S_2 \wedge \boldsymbol{P}_2$

$G = x \wedge S_1 \vee \neg x \wedge S_2$

で計算する（これはシャノン展開そのもの）.

具体的な制約条件の分割は以下の通りである．4次の魔法陣では，図2での（P 1）～（P 9）に相当する制約条件を $\boldsymbol{P}_1$ とし，残りの制約条件を $\boldsymbol{P}_2$ とした．論理和型の出力関数（（P 11）に相当する制約条件）を二つの論理和に分けることによって問題分割を行った．N-Queens 問題では，行と列に女王を一つだけ置くという制約条件を $\boldsymbol{P}_1$ とし，対角線上にはたかだか一つしか女王が置けないという制約条件を $\boldsymbol{P}_2$ とした．問題分割は，行の択一型制約条件をもとに行った．

もし，1段の分割でも計算できないときには，さらに別の候補を選択し，多段の分割を行う．主記憶が 128 Mbyte の場合には，4次の魔法陣では1段の分割で十分であり，13-Queens では2段の分割が必要である．

6. 実験結果と評価

本稿での実験環境は，SPARCStation 10（主記憶 128 Mbyte）であり，tcsh のもとで time を用いて測定を行った．時間はユーザとシステムの CPU 時間の合計を秒単位で示す．

6.1 制約順序・変数順序の評価

3次の魔法陣プログラム（図2）を取り上げて，提案する CCVO アルゴリズムで決定された制約順序・変数順序を評価する．さまざまな制約順序と変数順序によって，最終ノード数と計算途中で必要となる最大ノード数，および，実行時間がどう変わるかを実験した．

比較の対象とした制約順序は以下の5種類である．

（1）**CCVO**：3個のマス目の制約の次に和の制約（S_i）を繰り返し与え，その後で数の制約等を与える．

（2）**盤数**：マス目の制約（C_i）を与え，その後で数の制約（D_i）を，最後に和の制約（S_i, T_i）等を与える．

（3）**数盤**：数の制約，次にマス目の制約，最後に和等の制約を与える．

（4）**交互型**：マス目の制約，数の制約を3組与え和の制約（S_i）を与えることを繰り返す．これは，CCVO の最初の変形アルゴリズムで決まる制約順序である．

（5）**即実行型**：CCVO の2番目の変形アルゴリズムである即実行型アルゴリズムで決まる制約順序である．これはまた3次の魔法陣に対して有望そうなものをシラミ潰しで発見した制約順序でもあり，ほぼ最適な制約順序と考えられる．

C 1 C 2 C 3 S 1 K C 4 C 5 C 6 S 2 C 7 T 1 Y L M C 8 T 2 C 9 S 3 D 1 D 2 D 3 D 4 D 5 D 6 D 7 D 8 D 9 T 3 X N

また，変数順序は以下の4種類を取り上げた．

（1）**盤優先**：マス目を優先．一つのマス目では小さい数を優先．

x 11 x 12 … x 19 x 21 … x 29 … の順に優先度が下がる（下位になる）.

（2）**逆盤優先**：マス目を優先．一つのマス目では大きい数を優先．

x 91 x 92 … x 99 x 81 … x 89 … の順に優先度が下がる．

（3）**数優先**：数を優先．一つの数では小さいマス目を優先．

x 11 x 21 … x 91 x 12 … x 92 … の順に優先度が下がる．

（4）**逆数優先**：数を優先．一つの数では大きいマス目を優先．

x 19 x 29 … x 99 x 18 … x 98 … の順に優先度が下がる．

（5）**盤逆優先**：マス目を優先．一つのマス目では大きい数を優先．

x 19 x 18 … x 11 x 29 … x 21 … の順に優先度が下がる（下位になる）.

盤逆優先変数順序を試した理由は，CCVO アルゴリズムでは変数順序が完全には定まらず，ユーザの書いたプログラムに依存する部分が残っており，たとえば，盤優先や盤逆優先などの可能性があるからである．

3次と4次の魔法陣の実行結果を**表 4** と**表 5** にまとめる．

これらの実験から，本論文で提案する CCVO アルゴリズムによって得られる制約順序と変数順序はほぼ最適であることがわかった．4次の魔法陣では，次節で説明する分割統治法を用いて1段の部分問題分割で解が求まったのは，CCVO アルゴリズムで得られた順序，交互型，および盤数（7つの部分問題に分割）の3つの制約順序だけである．3次の魔法陣で即実行型が一番速いのは，変数の個数が少ないので，制約条

表 4 3次の魔法陣プログラムにおける変数順序と制約順序の影響
Table 4 Variable ordering and constraint ordering in 3-order magic square program.

最終ノード(システム全体)	変数順序									
	盤優先 81 (237)		逆盤優先 81 (236)		数優先 81 (237)		逆数優先 81 (264)		盤逆優先 81 (264)	
制約 順序	時間(秒)	最大ノード数	時間(秒)	最大ノード数	時間(秒)	最大ノード数	時間(秒)	最大ノード数	時間(秒)	最大ノード数
CCVO	7.31	2341	7.47	2341	61.75	221078	52.82	221105	6.78	2369
盤 数	9.41	10504	10.17	10743	19.63	33944	16.42	33971	8.44	10770
数 盤	13.16	33947	16.52	30329	15.12	7838	9.17	7865	12.12	33971
交互型	7.35	7.53	7.35	2791	15.89	11542	10.17	12685	6.78	2818
即実行型	7.01	7.06	7.06	1958	15.37	12805	9.88	12159	6.49	1985

表 5 4次の魔法陣プログラムにおける変数順序と制約順序の影響
Table 5 Variable ordering and constraint ordering in 4-order magic square program.

最終ノード数(システム全体)		変数順序				
		盤優先 109999 (110529)	逆盤優先 112443 (112973)	数優先 102494 —	逆数優先 103004 —	盤逆優先 112468 (113103)
制約順序	分割数	時間(秒)	時間(秒)	時間(秒)	時間(秒)	時間(秒)
CCVO	2	1299.29	1820.16	×	×	1259.76
盤 数	7	7038.56	×	×	×	6375.31
数 盤	7	×	×	×	×	×
交互型	2	1601.13	4169.77	×	×	2049.21
即実行型	7	×	×	×	×	×

×は1段のいかなる分割でもノード数の爆発で計算ができないことを示す.

件間の BDD のノード数にあまり差が出ず, CCVO の優位さが顕在化しないからである.

魔法陣の解法から以下のような知見が得られた：

(1) 同じ変数順序でも,制約順序によって最大ノードや実行時間が異なる. 特に, 最大ノード数の差は大きい.

(2) 規則的な変数順序を用いる場合には,最終ノード数の差はわずかである.

3次の魔法陣では, 解が1個であるので, いかなる変数順序を用いても解を表す BDD のノード数は同じである. 4次の魔法陣では, 表5で取り上げた変数順序では最大ノード数の差はあまりないが, ランダムな変数順序では, 最終ノード数は 160,500 程度と表5の 1.5 倍程度になるものが多い. また, N-Queens 問題でもランダムな変数順序の場合には最終ノード数は CCVO アルゴリズムで得られた変数順序の場合よりも多くなっている.

(3) 制約順序にはそれにふさわしい変数順序がある. 制約条件が解くべき組合せ問題を表現しているので, CCVO アルゴリズムが与える変数順序は問題の構造を反映したものである. 変数順序に問題の構造を反映させる方針は, 回路図にしたがって変数順序を決めるとほぼよい結果を得るという報告[4]とも一致している.

(4) 最大ノード数が大きければ実行時間も遅い.

(5) 4次の魔法陣の最適変数順序(最終ノード数が最小)では, 1段の分割統治法ではどのような制約順序でも解が求まらない.

(6) すべての制約条件が解を求めるのに貢献するわけではない. また, 制約順序によって冗長な制約条件は変化する. たとえば, 即時型制約順序では, D1 を実行すれば, その時点で解が得られ, 残りの D2〜D9 は冗長な条件である.

(7) 4次の魔法陣の場合, 組合せ的爆発が 100 秒程度ですぐ生じる場合もあれば, 1万秒以上も走ってから生じる場合もある.

なお, 変数順序が同じであると, BDD の最終ノード数が同じであることに注意してほしい. これは, BDD がカノニカル表現であることを示している.

6.2 計算可能性の限界

本節では二つのベンチマークプログラムについて報告する.

魔法陣のプログラムの実行結果を**表6**に示す. 4次の魔法陣は, 変数を 256 個使用する. 変数の個数から, 4次の魔法陣は 16-Queens に相当する組合せ生

表 6 魔法陣プログラムの実行結果
Table 6 Result of execution of magic-square program.

	魔法陣	変数の数	最終ノード数	最大ノード数	時間(秒)
3	1	81	81	2785	7.35
4	880	256	109999	2231212	1299.29

成器を持つので，それを解くには制約順序が重要なだけでなく，オンライン型分割統治法が必要であった．魔法陣のプログラムは対称解を除くようになっているので，P_{11} には1から7までしか置けない．この事実を利用して，問題分割を行った（除外した9個についてのチェックには，339.39秒余分にかかる）．さらに，この事実から変数の個数を9個減らすと，実行時間は1% の改善，最終ノード数は微減する．4次の魔法陣を，同じ制約条件で Lisp を用いてバックトラック型プログラムで解くと，同じ計算環境の下で 11 時間42分要した☆.

N-Queens 問題についての従来の報告では，主記憶 128 Mbyte のシステム（最大 4 M ノード）のもとで 11-Queens までしか解かれていなかった[8),15)]．同じ規模のシステム上で，提案する CCVO アルゴリズムから得られた制約順序と変数順序を使用することにより 12-Queens まで解くことができた．京都大学の越智裕之氏はわれわれの報告を受け，同じ手法を用いて主記憶 1 Gbyte のシステム（最大 50 M ノード）で，13-Queens と 14-Queens を解くことに成功している．これらの結果を**表 7** にまとめる．さらに，本稿で提案した分割統治法により，主記憶 128 Mbyte のシステムで 13-Queens を解くことができた．13-Queens の場合，最終的なノードが 2 M 個程度であるので，システムの全ノードを目一杯使用した問題が解けたといえよう．Solbonne の 13-Queens のデータと 12-Queens の両者のデータから，主記憶が十分に大きい SPARC Station で 13-Queens を解いたときの速度は，分割統治法による解法よりも1桁程度速く，1400秒程度と予測できる．

Lisp で 11-Queens と 12-Queens をバックトラック型で解くと，それぞれ 16.53 秒と 98.33 秒を要した．また，全解探索型の真偽維持システム ATMS[10)] を用いると，11-Queens は11時間48分で解けたが，12-Queens はメモリ不足で解けなかった．

☆ プログラムを工夫すれば，FACOM 270/20 の FORTRAN で880個の解が数分程度で求まるという報告がある[14)].

6.3 オンライン型分割統治法の評価

オンライン型分割統治法は，オフライン型分割統治法と比較すると，すでに計算した結果を使用することができること，および，中間結果を保持するための出力ファイルが不要なこと，といった利点がある．本節では，二つのベンチマーク問題について，解法のための時間と出力ファイルの大きさについて調べた結果を示す．

魔法陣の問題をオンライン型とオフライン型の2つの分割統治法で解いた結果を**表 8** に示す．問題は2つ

表 7 N-Queens 問題解法の結果
Table 7 Result of N-Queen problem.

SPARCStation 10 主記憶 128 Mbyte BEM-II（最大 4M ノード）

女王	変数の数	解の数	最終ノード数	時間(秒)
4	16	2	29	0.42
5	25	10	166	0.70
6	36	4	129	1.06
7	49	40	1098	2.07
8	64	92	2450	4.72
9	81	352	9556	9.97
10	100	724	25944	30.82
11	121	2680	94821	120.97
12	144	14200	435169	392.73
13	169	73712	2044393	13394.58

Solbonne (Sun4 相当) 主記憶 1 Gbyte 京都大学のパッケージ（最大 50M ノード）

女王	変数の数	解の数	最終ノード数	時間(秒)
12	144	14200	425217	1129.3
13	169	73712	2000566	4068.7
14	296	365596	9381383	29503.3

Solbonne 上のデータは京都大学越智裕之氏による．
時間にはノード数，解の個数の出力時間を含む．いずれのシステムも湊 真一氏 (NTT) が開発．

表 8 4次の魔法陣のオンライン型とオフライン型分割統治法の比較
Table 8 Comparison of online and offline divide-and-conquer for 4-order magic square problem.

統合方法	時間	(出力時間)	最終ノード数	解の個数	ファイル
オンライン型	1299.29	—	109999	880	—
オフライン型					
部分問題1	1135.34	(514.46)	72037	574	1097KB
部分問題2	901.59	(273.87)	41335	306	585KB
統合	1430.44	—	109999	880	—
合計	3467.37	(788.33)	109999	880	1682KB

の部分問題に分割した．オンライン型はオフライン型より2.7倍速い．また，オフライン型では，途中結果の出力として 1.7 Mbyte のファイルが必要である．

N-Queens 問題をオンライン型とオフライン型の2つの分割統治法で解いた結果を**表 9** に示す．問題は29の部分問題に分割をした．オンライン型はオフライン型より7.2倍速い．また，オフライン型では，途中結果の出力として 89 Mbyte のファイルが必要である．

6.4 分割における粒度

主記憶が 128 Mbyte の場合には，4次の魔法陣では1段の分割で十分であるが，13-Queens では2段の分割が必要であった．問題を分割するときの粒度と実行時間との関係を調べるために，3次の魔法陣で，分割数を2から7まで変えたときの実行時間の変化，および，解の数と BDD のノード数の増え方を**表 10** に示す．分割数を2から7に増やし粒度を細かくしても，実行時間は1.4倍しか増えないことがわかり，オンライン型分割統治法の有効さを示すことができた．

6.5 考　察

前節で述べた実験から以下のような結果が得られた．

- CCVO アルゴリズムによって，計算途中で必要な最大ノード数をほぼ最小にする変数順序と制約順序を求めることができた．これによって，同じ空間容量では従来計算不可能であった問題を解くことができた．
- オンライン型分割統治法によって，同じ空間容量では従来計算不可能であった問題を解くことができた．

組合せ問題を論理式で表現し，BDD でコンパクトに表現すれば，従来にはないようないくつかの新しい機能を提供することが可能となり，さまざまな応用分野が広がると考えられる．今，BDD が表現する論理式を F としよう．いくつかの具体例によって BDD の有効性を示す．

（1）人間が求めた解が正しいかのチェック：

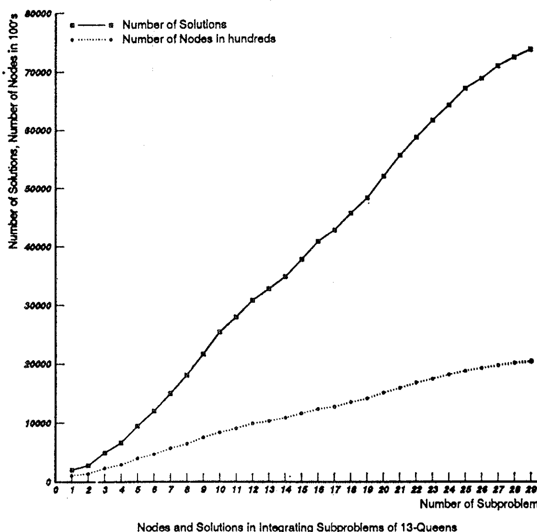
解を x とすると，$\neg x \vee F$ が恒真かを調べる．$\neg x \vee F$ の BDD が1に帰着できるかを

表 9 13-Queens のオンライン型/オフライン型分割統治法の比較
Table 9 Comparison of online and offline divide-and-conquer for 13-queen problem.

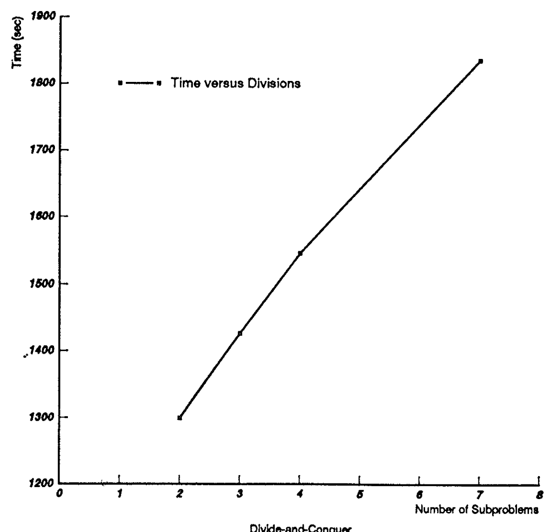
統合方法	時間（秒）（内出力）	ファイルサイズ（KB）
オンライン型	13394.58	—
オフライン型		
部分問題 1～29	30758.24 (16486.34)	89384
統　合	65323.38	—
合　計	96081.62	89384

表 10 分割統治法の粒度
Table 10 Granularity of divide-and-conquer.

分割数	分割法	時間（秒）
2	123/4567	1299.29
3	12/34/567	1426.25
4	12/34/56/7	1545.67
7	1/2/3/4/5/6/7	1834.97



Nodes and Solutions in Integrating Subproblems of 13-Queens



Divide-and-Conquer

見ればよい．

（2）人間がすべての解を求めたかのチェック：
解の論理和を X とする．X がすべての解を表現するかは，X の BDD と F の BDD とが一致するかを調べればよい．これは，ポインタの比較で行える．
さらに，一致しない場合の反例を求めるには $F \oplus X$ を求めればよい（ただし，$\oplus$ は排他的論理和を示す）．なお，$F \oplus X$ の BDD が0に帰着できる場合は，X がすべての解を表現していることを示している．

（3）人間が問題を正しくコーディングしたかのチェック：
問題の仕様を Y とすると，$F \oplus Y$ が恒偽かを調べる．$F \oplus Y$ の BDD が0に帰着できるかを見ればよい．

（4）得られた解にさらに制約条件を付加して別の問題を解く：
制約条件を C とすると，$F \wedge C$ を計算する．例で説明しよう．両端をつなげ（トーラス面に張りつけ），どの斜めも同じ数になる魔法陣は悪魔法陣（diabolic square）と呼ばれる[13]．求まった BDD にさらなる制約条件を与えれば，悪魔法陣の個数が 48 であることがただちにわかる．このように求まった解にさらに制限を加えた解を求めることができるのは，BDD による全解を同時に求める方法の強みである．

（5）冗長な制約条件のチェック，定理の証明：
制約条件を C とすると，$\neg F \vee C$ が恒真かを調べる．
たとえば，4次の魔法陣で内側の正方形に含まれる数の和が 34 である定理はその論理式 C とすれば，上記の方法によって簡単に証明できる．

（6）解の内包的表現と外延的表現の比較：
問題の仕様を論理式で与えることは，解を内包的（intensional）に表現したことである．この論理式を I としよう．問題の解をすべて（重複を許して）論理式の論理和で与えることは，解を外延的（extensional）に表現したことである．この論理式を E としよう．両者が一致するかどうかは，それぞれの BDD のポインタを比較すればよい．一致しない場合には，
$I \oplus E$ で反例が求まる．

これらの計算はたかだかノード数に比例する時間で行うことができる．

本稿で提案した BDD 使用法は組合せ問題に適用できるだけではない．疑似ブール代数解法による多面体や曲面物体の自動生成[6]にも適用することができる．三面図における平面と稜線を変数で表現し，平面と稜線の関係を BEM-II で記述し，BDD を構成する．次に，その BDD において根からの1へのパスを列挙することで，候補物体を得られる．もし，候補物体が複数個ある場合には，さらなる制約規則を適用することで，候補物体を絞ることができる．文献6）の例ではいずれも変数の数が 100 個程度と小さいが，BDD を使用し，本稿で提案した変数順序・制約順序決定アルゴリズムである CCVO あるいはオンライン型分割統治法を適用すれば，極めて大きな問題も取り扱うことができよう．また，上述したように，組合せ問題に対して BDD を応用することによってさまざまな機能が得られるのと同様に，この場合にも BDD を用いれば，三面図間の等価性のチェック，三面図と実物体との一貫性のチェックなどにも応用できよう．

7．おわりに

本稿では，探索型組合せ問題に BDD を適用するときの問題点として，制約順序の重要性および変数順序の制約順序への依存性を指摘し，最適な制約順序と変数順序を求めるアルゴリズム CCVO を提案した．また，組合せ的爆発によってどうしても計算できない問題に対しては中間結果を再利用するオンライン型分割統治法を応用することを提案し，問題分割法を提案した．ここで提案した手法の有効性を，魔法陣の問題と N-Queens 問題に適用し，確認した．これらの結果，組合せ問題解法の高機能化と高速化を同時に達成する可能性が開けたと考えられる．

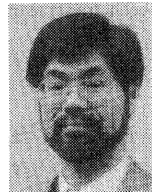
謝辞 NTT LSI 研究所　湊真一氏からは，SBDD パッケージ BEM-II の提供と，その使用法についてご教示いただき，さらに，BDD の組合せ問題への応用についてコメントをいただいた．茨城大学仙波一郎助教授には再帰関係型組合せ問題への BDD の応用について，UCB の久木元裕治氏には BDD について教えていただき，東京大学工学部の下國治氏には Common Lisp で SBDD を実装していただいた．BDD メーリングリスト参加者の多くの方々には本研究についてのコメントを，また，NTT 基礎研究所竹内郁雄

グループリーダーには本稿についてのコメントをいただいた．以上の方々に感謝します．

参考文献

1) Akers, S. B.: Binary Decision Diagrams, *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 509-516 (1978).
2) Brace, K. S., Bryant, R. E. and Rudell, R. L.: Efficient Implementation of a BDD Package, *Proc. of 27th Design Automation Conf.*, ACM/IEEE, pp. 40-50 (1990).
3) Bryant, R. E.: Graph-based Algorithm for Boolean Function Manipulation, *IEEE Trans. Comput.*, Vol. C-35, No. 5, pp. 677-691 (1986).
4) 藤田昌宏，藤沢久典，松永裕介，角田多苗子：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，情報処理学会論文誌，Vol. 31, No. 4, pp. 532-541 (1990).
5) 藤田昌宏，佐藤政生：特集「BDD（二分決定グラフ）一幅広い応用範囲をもつ論理関数の処理技術一」の編集にあたって，情報処理，Vol. 34, No. 5, p. 584 (1993).
6) 伊藤　潔：三面図を用いたソリッドモデルの構成一主に多面体を対象として一，情報処理，Vol. 31, No. 8, pp. 1095-1106 (1990).
7) 湊　真一，石浦菜岐佐，矢島脩三：論理関数の共有二分決定グラフによる表現とその効率的処理手法，情報処理学会論文誌，Vol. 32, No. 1, pp. 77-85 (1991).
8) 湊　真一：BEM-II：二分決定グラフを用いた算術論理式計算プログラム，信学技報，COMP 92-75, 電子情報通信学会 (1993).
9) 湊　真一：計算機上での BDD の処理技法，情報処理，Vol. 34, No. 5, pp. 593-599 (1993).
10) 奥乃　博：網：ATMS の新しい処理系とその並列処理，人工知能学会誌，Vol. 5, No. 3, pp. 333-342 (1990).
11) Senba, I. and Yajima, S.: Combinatorial Algorithms by Boolean Processing I, 情報処理学会アルゴリズム研究会報告，32-4 (1993).
12) Senba, I. and Yajima, S.: Combinatorial Algorithms by Boolean Processing II, 情報処理学会アルゴリズム研究会報告, 32-5 (1993).
13) 嶋田君枝：4×4 魔法陣の話，数学セミナー，Vol. 12, No. 2, pp. 18-22 (1973).
14) 安村，佐々，古森，川合，野下：4×4 魔法陣数え上げのプログラム競争，*bit*, Vol. 5, No. 9, pp. 86-89 (1973).
15) 柳谷雅之：組合せ最適化問題の BDD による解法，情報処理，Vol. 34, No. 5, pp. 617-623 (1993).
16) 渡部悦穂，久木元裕治：BDD の応用，情報処理，Vol. 34, No. 5, pp. 600-608 (1993).

（平成5年8月30日受付）
（平成6年1月13日採録）



奥乃　博（正会員）

1950 年生．1972 年東京大学教養学部基礎科学科卒業．同年電電公社（現 NTT）武蔵野電気通信研究所入所．1986～1988 年スタンフォード大学コンピュータ科学科知識システム研究所客員研究員．1992～1993 年東京大学工学部電子工学科知能工学（富士通）寄付講座客員助教授．現在，NTT 基礎研究所情報科学部勤務．主幹研究員．Lisp, 演繹データベース，論理型プログラミング，プログラミング環境の研究を経て，現在は推論機構，創発的計算モデル，音環境理解の研究に従事．1990 年度人工知能学会論文賞受賞．人工知能学会，日本認知科学会，日本ソフトウェア科学会，ACM, AAAI 各会員．本学会英文図書委員，人工知能学会並列人工知能研究会幹事．「知的プログラミング」（共著，オーム社，1993)．