HOKKAIDO UNIVERSITY

Title	二分決定グラフによる制約充足問題の解法
Author(s)	奥乃, 博; 湊, 真一
Citation	情報処理学会論文誌, 36(8), 1789-1799
Issue Date	1995-08-15
Doc URL	https://hdl.handle.net/2115/47399
Rights	ここに掲載した著作物の利用に関する注意 本著作物の著作権は情報処理学会に帰属します。本著作物は著作権者である情報処理学会の許可のもとに掲載するものです。ご利用に当たっては「著作権法」ならびに「情報処理学会倫理綱領」に従うことをお願いいたします。
Type	journal article
File Information	55_johoshori.pdf

Hokkaido University Collection of Scholarly and Academic Papers : HUSCAP

二分決定グラフによる制約充足問題の解法

奥乃 博[†] 湊 真一[††]

BDD(二分決定グラフ)はブール関数のコンパクトな表現方法である．我々は，BDDを使用して組合せ問題の複数の解を同時に表現したり，ATMSといった多重文脈型真偽維持システムの機能拡張をする方法を検討してきた．与えられた問題記述あるいは制約条件からBDDを構築する過程は制約充足問題の解法とみなすことができる．本稿では，2種類のBDD，算術論理式が使用できる通常のBDDと組合せ集合が使用できるZBDD (Zero-Suppressed BDD) を取り上げ，それらを用いた制約充足問題の解法を検討する．制約充足問題のデータと制約条件のコーディング方法を提案し，N-Queens問題や魔方陣の問題などの具体的な問題を取り上げ，2種類のBDDによる解法を評価する．さらに，BDDによる解法を，制約充足問題での一貫性アルゴリズムやATMSと比較し，評価を行う．BDDでは，一旦適用された制約条件が以降ずっと成立するという単調一貫性維持が成立する．一方，ZBDDでは，組合せ集合演算の性質から，制約条件が適用する対象によって制限される．しかし，この結果ZBDDでは段階的解法が容易となる．

Solving Constraint Satisfaction Problems by Binary Decision Diagram

Hiroshi G. Okuno [†] and Shin-ichi Minato [††]

BDD (Binary Decision Diagram) is a compact representation of boolean functions. We have applied BDD to represent all solutions of combinatorial problems or to extend capabilities of multiple-context truth maintenance systems such as ATMS. The process of building BDDs from a set of problem specifications or constraints is considered as solving Constraint Satisfaction Problems (CSPs). In this paper, we focus on two types of BDD; a normal BDD representing arithmetic logical formula, and ZBDD (Zero-Suppressed BDD) representing sets. We present encoding methods of data and constraints and new operations for ZBDD. We evaluate these methods by solving N-Queens and Magic Square problems. Then, we discuss the relationship of these methods to constraint propagation methods and ATMS. In BDD, the monotonic consistency maintenance that constraints applied so far hold for ever. In ZBDD, when a constraint is applied to a set, its elements unrelated to the set are removed from it. However, this property allows incremental computation in ZBDD.

1. はじめに

組合せ問題は，問題空間における制約条件を満足する点の探索であり，制約充足問題としてとらえることができる．制約充足問題では解を1つだけ求めるか，あるいは，その延長としてすべての解を数え上げることが中心であった．具体的な解法としては，後ろ戻り型探索や制約伝播によって問題の簡約化，さらには，後ろ戻り型探索と制約伝播とを組み合わせた方法など数多く提案されている[9]．

一般に制約充足問題を内包する問題では，制約充足問題が解ければ終了というわけではない．制約充足問題で得られた結果を使用してさらに処理を行うためには，すべての解が同時に表現できることが必要である．すべての解を表現できれば，たとえば，異なった解の間での比較，解全体に対する論証などを容易に行うことができる[5),19)]．

これまで，二分決定グラフ (BDD) を使用して複数の解を同時に表現したり[19]，ATMS[5),17)]といった多重文脈型真偽維持システムの機能拡張をする方法[11),18)]が検討されてきた．与えられた問題記述あるいは制約条件からBDDを構築する過程は制約充足問題の解法とみなすことができる．本稿では，2種類のBDD—通常のBDDとZBDD—を取り上げ，それらによる制約充足問題の解法を提案し，さらに，従来の制約充足問題の解法である制約伝播およびATMSとの関連について議論する．

† NTT 基礎研究所
NTT Basic Research Laboratories, Nippon Telegraph and Telephone Corporation
†† NTT LSI 研究所
NTT LSI Laboratories, Nippon Telegraph and Telephone Corporation

以下，第2章で，BDDによる算術論理式表現とZBDDによる組合せ集合表現について説明し，第3章で，制約充足問題の算術論理式と組合せ集合へのコーディングを提案し，第4章で，BDDによる解法の問題点を指摘し，制約充足問題の解法やATMSとの関係を議論し，第5章で，BDDとZBDDによるプログラミングを具体例で比較し，第6章で，まとめをする．

2. 二分決定グラフ（BDD）

2.1 BDD（Binary Decision Diagram）

BDDは，Akersが提案し[1]，Bryantが効率のよいBDD処理アルゴリズムを考案したコンパクトな論理関数（論理式）の表現法である[2),12)]．BDDの主な特徴は次の3点である：

（1）変数順序を固定すると論理関数の標準形（カノニカル形）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がノード数に比例する時間で行える．

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分野で使用されている．

BDDが表現するのはブール関数であるが，その外部表現としては算術論理式を使用することができる[13]．たとえば，変数x_1, x_2, x_3のどれか1つだけが成立することを表現する択一型論理式$(x_1 \wedge \overline{x_2} \wedge \overline{x_3}) \vee (\overline{x_1} \wedge x_2 \wedge \overline{x_3}) \vee (\overline{x_1} \wedge \overline{x_2} \wedge x_3)$は，算術論理式を用いると$x_1+x_2+x_3==1$と書くことができる．ここで，“==”は算術論理式の値が等しいという述語である．

今，変数集合から取り出した要素の組合せを集合の要素とする組合せ集合をBDDで表現することを考えよう．その準備として変数集合を“〈”と“〉”の対でくくって表現し，変数集合の組合せを要素とする組合せ集合を“{”と“}”の対でくくって表現することにする．本稿では，組合せ集合を単に集合と呼ぶこともある．

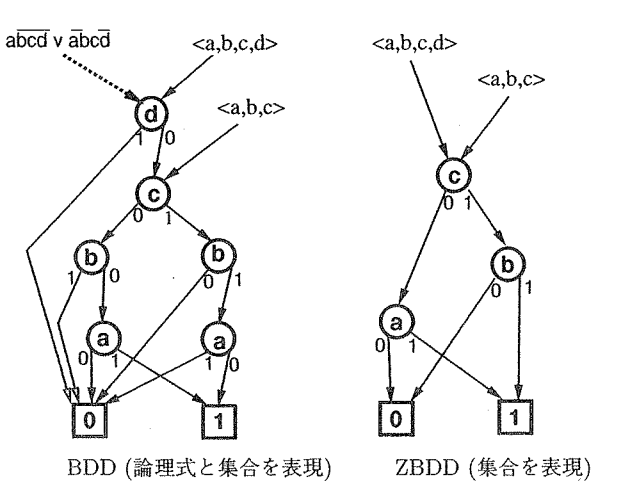


図1 BDDとZBDDによる組合せ集合$\{a, b\times c\}$の表現
Fig. 1 Representation of a combinatorial set $\{a, b\times c\}$ by BDD and ZBDD.

図1の左側のBDDの1番左上のノードは$(a \wedge \bar{b} \wedge \bar{c} \wedge \bar{d}) \vee (\bar{a} \wedge b \wedge c \wedge \bar{d})$という論理式を表現する．この論理式は，変数集合$\langle a, b, c, d\rangle$からの組合せ集合$\{a, b\times c\}$を表現していると解釈することもできる．変数集合が$\langle a, b, c\rangle$であるとすると，同じ組合せ集合の表現は異なったものになる（図1左側のノードc）．このように，従来のBDDでは，組合せ集合の表現が一意に定まらなかった．

2.2 ZBDD（Zero-Suppressed BDD）

湊は組合せを構成する変数集合が未定であっても組合せ集合が効率よく表現できるBDDとしてZBDD（Zero-Suppressed BDD）を提案した[14]．ZBDDでは，ノードの表現方法およびノードの簡約方法が通常のBDDとは異なっているが，グラフはカノニカル形である．図1の右側には，変数集合が$\langle a, b, c, d\rangle$と$\langle a, b, c\rangle$の場合の組合せ集合$\{a, b\times c\}$を示した．このように，ZBDDでは，組合せ集合がユニークに表現できる．ノード[0]は空集合（{}，以下ϕ）を表し，ノード[1]は集合$\{\varepsilon\}$を表す．後者の唯一の要素εは空の組合せを表し，すべての変数の値が暗黙値として0を取ることを意味している．ZBDDを集合表現に使用すると，BDDから主項（prime implicant）が効率よく求められることが知られている[4]．一般に疎な組合せの集合を扱う場合にはZBDDの方がBDDよりも効率よく表現できる[14]．

2.3 組合せ集合演算

組合せ集合演算の基本演算には以下のようなものがある．組合せ集合の要素はその表現における変数の順序には無関係である．たとえば，$a\times b\times c, b\times c\times a, a\times b\times b\times c, a\times b\times c\times c\times b\times a$はすべて同じ要素である．組合せ集合に対する演算は，単なる集合の演算とは少し異なるので，理解を助けるために例とともに説明をする．今，$F=\{a\times b, a\times b\times c, b\times c\times d\}$，$C=\{a\times b\times c, b\times c\}$としよう．

（1）組合せ集合要素どうしの積“×”，商“÷”，余り“%”：$\mathrm{var}(\alpha)$を組合せ要素αに含まれる変数の集合（ただし，$\mathrm{var}(\varepsilon)=\phi$）とすると次のように定義される．

$$\alpha\times\beta=\begin{cases}0 & \text{if} \quad \alpha=0\vee\beta=0\\ \prod x & \text{where } x\in(\mathrm{var}(\alpha)\cup\mathrm{var}(\beta))\\ & \text{Otherwise.}\end{cases}$$

$$\alpha \div \beta \equiv \begin{cases} \varepsilon & \text{if} \quad \mathrm{var}(\alpha)=\mathrm{var}(\beta) \\ 0 & \text{if} \quad \mathrm{var}(\alpha)\subset\mathrm{var}(\beta) \\ \prod x & \text{where } x\in(\mathrm{var}(\alpha)-\mathrm{var}(\beta)) \\ & \text{Otherwise.} \end{cases}$$

$\alpha\%\beta \equiv \alpha-((\alpha\div\beta)\times\beta)$.

例：$(a\times b\times c)\times(a\times d\times f)=a\times b\times c\times d\times f$,
$(a\times b\times c)\div(a\times b)=c$,
$(a\times b\times c)\div(a\times d\times f)=0$,
$(a\times b\times c)\%(a\times b)=0$,
$(a\times b\times c)\%(a\times d\times f)=a\times b\times c$

（2）変数と組合せ集合の積“×”：F の各組合せに x を付加する．組合せの中で，重複した変数は削除される．また，重複する組合せも集合から削除される．定義を次に示す．$F\times x\equiv\{\alpha\times x|\alpha\in F\}$
例：$F\times d=\{a\times b\times d, a\times b\times c\times d, b\times c\times d\}$

（3）組合せ集合の和“+”，差“−”：次のように定義される．$P+Q\equiv\{\alpha|\alpha\in P\vee\alpha\in Q\}$,
$P-Q\equiv\{\alpha|\alpha\in P\wedge\alpha\notin Q\}$
例：$F+C=\{a\times b, a\times b\times c, b\times c, b\times c\times d\}$
$F-C=\{a\times b, b\times c\times d\}$

（4）組合せ集合の直積“⊗”：それぞれの組合せ集合から取り出した組合せどうしの積を集合の要素とする新たな組合せ集合を求める．すなわち，直積は
$P\otimes Q\equiv\{\alpha\times\beta|\alpha\in P, \beta\in Q\}$ で定義される．
例：$F\otimes C=\{a\times b\times c, a\times b\times c\times d, b\times c\times d\}$

（5）組合せ集合間の商“/”と余り“\”：次のように定義される．$P/Q\equiv\bigcap_{\alpha\in Q}(P\div\alpha)$,
$P\backslash Q\equiv P-(P\otimes(P/Q))$.
例：$F/C=\phi$, $F/F=\{\varepsilon\}$, $(F\otimes C)/C=\{d\}$,
$F\backslash C=\{a\times b, a\times b\times c, b\times c\times d\}$,
$(F\otimes C)\backslash C=\{a\times b\times c\}$

（6）組合せ集合の和集合 ∪：組合せ集合の和と同じ．

（7）組合せ集合の共通集合 ∩：次のように定義される．
$P\cap Q\equiv\{\alpha|\alpha\in P\wedge\alpha\in Q\}$.
例：$F\cap C=\{a\times b\times c\}$

（8）組合せ集合の等価性，包含関係：
定義は $\alpha\supseteq\beta$ iff $\exists\gamma$ s.t. $\gamma\times\beta=\alpha$.
例は $a\times b\times c\times d\supseteq a\times b$.

2.4 制限演算，除外演算，許容演算

本稿では，制約充足問題のために3種類の集合演算を新たにZBDDに導入する．

（1）**制限演算**（Restriction）（$F\Delta C$）：
制限演算は，C のどれかの（組合せ）要素のスーパセットとなっているような F の要素を求める演算であり，次のように定義される．

$$F\Delta C\equiv\{\alpha\in F|\exists\beta\in C\ \ \alpha\supseteq\beta\}.$$

例：$F\Delta C=\{a\times b\times c, b\times c\times d\}$
（ただし，$F=\{a\times b, a\times b\times c, b\times c\times d\}$,
$C=\{a\times b\times c, b\times c\}$）
集合 F から，制約条件 C を満足する集合を求める．ここでの制約条件は，許容された組合せを求めるために使用される．

（2）**除外演算**（Exclusion）（$F\nabla C$）：
除外演算は，C のどれかの（組合せ）要素のスーパセットとなっているような要素を F から除外する演算である．これは，制限演算を用いて，

$$F\nabla C\equiv F-F\Delta C$$

と定義する．ただし，C が単一の要素しか持たない時には，除外演算 $F\nabla C$（$C=\{\alpha\}$ の時）は $F\%\alpha$ と同じになる．
例：$F\nabla C=\{a\times b\}$
集合 F から，制約条件 C を満足しないような集合を求める．ここでの制約条件は，禁止された組合せを排除するために使用される．

（3）**許容演算**（Permission）（$F\oslash C$）：
許容演算は，C のどれかの（組合せ）要素のサブセットとなっているような F の要素を求める演算であり，次のように定義される．

$$F\oslash C\equiv\{\alpha\in F|\exists\beta\in C\ \ \alpha\subseteq\beta\}.$$

例：$F\oslash C\equiv\{a\times b, a\times b\times c\}$

制限演算と許容演算は，それぞれCoudertらが提案したSupSetとSubSetと同じ演算である[3]．本稿では，制限演算と除外演算だけを使用し，許容演算は使用してはいない．

制限演算は，$F\Delta C\equiv F\cap(F\otimes C)$ と定義することもできるが，この定義に忠実に実行すると，直積でむだなノードを作成してしまい効率が悪い．制限演算の実現アルゴリズムは，繰り返し型と再帰型の2種類が考えられる．繰り返し型アルゴリズムは，

$$F\Delta C=\bigcup_{\alpha\in C}((F\div\alpha)\times\alpha) \qquad (1)$$

で計算をする．

次に再帰型アルゴリズムを説明する．今，集合 P を

$$P=(x_k\times(P\div x_k))\cup(P\%x_k)$$

と表現する．ただし，$(P\%x_k)$ は P の要素のうち x_k を含まないものを集めた集合であり，$(P\div x_k)$ は残りの各要素から x_k を除いたものの集合である．この時，制限演算 $F\Delta C$ は次の再帰型アルゴリズムで計算でき

る．

$$F\Delta C=\begin{cases} F & \text{if} \quad \varepsilon\in C \text{ or } F=C \\ \phi & \text{else if} \quad F=\phi \text{ or } C=\phi \text{ or } F=\{\varepsilon\} \\ (\{x_k\}\times(((F\div x_k)\Delta(C\div x_k)) & \\ \quad \cup((F\div x_k)\Delta(C\%x_k)))) & \\ \quad \cup((F\%x_k)\Delta(C\%x_k)) & \\ \quad \text{Otherwise.} & \end{cases} \tag{2}$$

繰り返し型では要素を1個ずつ取り出して計算するため要素数（式の文字数）に比例する計算時間がかかる．一方，再帰型では変数順序に従って部分計算に分解して実行するので，類似した部分計算をすることが多くなり，もしその結果が演算キャッシュにあれば，それを使用することによってその部分計算を省略することができる．ただし，いずれのアルゴリズムも，与えられた組合せ集合を自明な要素まで分解して，個々に処理をしている点は共通している．したがって再帰型アルゴリズムでキャッシュを使わない場合は，分解の順序が繰り返し型と異なるだけなので，両者の計算量はほぼ同じになろう．

制限演算，除外演算に対して以下のような定理が成立する．

定理1

$$F\Delta(\bigcup_i C_i)=\bigcup_i(F\Delta C_i) \tag{3}$$

$$(\bigcup_i X_i)\Delta C=\bigcup_i(X_i\Delta C) \tag{4}$$

$$F\Delta(\bigotimes_i C_i)=\bigcap_i(F\Delta C_i) \tag{5}$$

$$=((F\Delta C_1)\cdots)\Delta C_n \tag{6}$$

定理2

$$F\nabla(\bigcup_i C_i)=\bigcap_i(F\nabla C_i) \tag{7}$$

$$=((F\nabla C_1)\cdots)\nabla C_n \tag{8}$$

$$(\bigcup_i X_i)\nabla C=\bigcup_i(X_i\nabla C) \tag{9}$$

$$(\bigotimes_i X_i)\nabla C)=(\bigotimes_i(X_i\nabla C))\nabla C \tag{10}$$

$$F\nabla(\bigotimes_i C_i)=\bigcup_i(F\nabla C_i) \tag{11}$$

制限演算や許容演算と和集合，集合の直積などの演算との間で交換則が成立する．これらの定理の証明は図を使用すると簡単なので省略する．定理2(10)から，「問題を分割し，除外演算を行い，その結果を統合する」というアプローチでは正しい答が見つけられないことが分かる．このような不完全性は，節形式のブール制約充足（Boolean Constraint Propagation, BCP）の不完全性と同じ性質のものである．論理式を節形式に変形し，節の集合に対してBCPを適用すると，処理効率はよいが正しい答が求まらない場合がある．それは，論理式を節形式に変換する際に情報が落ちたからであり，その落ちた情報を回復するために主項が必要となる[8]．

3. 制約充足問題のコーディング

制約充足問題（Constraint Satisfaction Problem, CSP）とは，変数の集合$\{x_1,\cdots,x_n\}$と各変数の値域D_i，および，変数間の制約条件の集合が与えられた時に，制約条件を満足する変数の具体的な値の組を求める問題である．制約条件には，変数間の許される組合せ，あるいは，逆に禁止される組合せ，線形計画法のようなコスト関数など，さまざまなものがある．今，3個の変数，x_1，x_2，x_3を考えよう．各変数に対する値域は$D_1=\{a,b\}$, $D_2=\{c,d,e\}$, $D_3=\{f,g\}$である．制約条件は，2変数x_i，x_jの許容組合せC_{ij}として与えられ，各々を$C_{12}=\{b\times c, b\times d, b\times e\}$, $C_{13}=\{b\times f, b\times g\}$, $C_{23}=\{d\times f, e\times g\}$としよう（**図2**では“×”を省略）．

3.1 算術論理式へのコーディング

変数の値域は，$X1=a\vee b$, $X2=c\vee d\vee e$, $X3=f\vee g$とコーディングする（大文字で始まる変数はBDD中間変数である）．変数が値を1つしか持たないという択一型制約条件は，$C1=(a+b==1)$, $C2=(c+d+e==1)$, $C3=(f+g==1)$とコーディングする．制約条件は，$C12=(b\wedge c)\vee(b\wedge d)\vee(b\wedge e)$, $C13=(b\wedge f)\vee(b\wedge g)$, $C23=(d\wedge f)\vee(e\wedge g)$とコーディングする．所与の問題は，

$$F=X1\wedge X2\wedge X3\wedge C1\wedge C2\wedge C3\wedge C12 \wedge C13\wedge C23 \tag{12}$$

とコーディングできる．Fに対するBDDの根から[1]へのパスをリストアップすることによって，$(b\wedge d\wedge f)\vee(b\wedge e\wedge g)$という解が得られる．もちろん，値域のコーディングや制約条件をそれらの共通変数という相関度の高い順に適用するようにした

$$F=X1\wedge C1\wedge X2\wedge C2\wedge C12\wedge X3\wedge C3 \wedge C13\wedge C23 \tag{13}$$

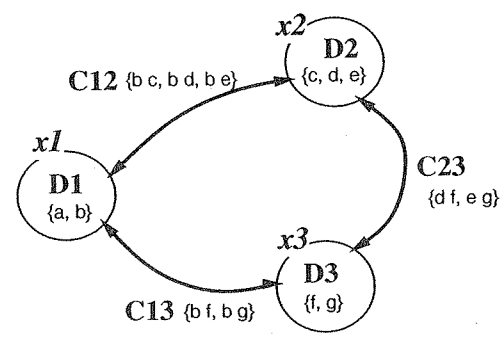


図2 制約充足問題の例
Fig. 2 A simple constraint satisfaction problem.

というコーディングでも結果は同じである．

3.2 集合へのコーディング

変数の値域は，$X1=\{a, b\}$, $X2=\{c, d, e\}$, $X3=\{f, g\}$ とコーディングする．変数が値を1つしか持たないという択一型制約は不要である．制約条件は，$C12=\{b\times c, b\times d, b\times e\}$, $C13=\{b\times f, b\times g\}$, $C23=\{d\times f, e\times g\}$ とコーディングする．所与の問題は，

$$F=(X1\otimes X2\otimes X3)\Delta(C12\otimes C13\otimes C23) \quad (14)$$

とコーディングできる．もちろん，定理1 (6) により

$$F=(((X1\otimes X2\otimes X3)\Delta C12)\Delta C13)\Delta C23 \quad (15)$$

とコーディングをしてもよい．F に対するZBDDの根から[1]へのパスをリストアップすることによって，$\{b\times d\times f, b\times e\times g\}$ という解が得られる．

変数間の禁止組合せという制約条件からも解くことができる．禁止組合せは $NG12=(X1\otimes X2)-C12$, $NG13=(X1\otimes X3)-C13$, $NG23=(X2\otimes X3)-C23$ で計算できるので，所与の問題は，$F=(X1\otimes X2\otimes X3)\nabla(NG12\otimes NG13\otimes NG23)$ とコーディングでき，同じ解が得られる．この場合も，定理2 (11) および定理2 (8) により $F=(((X1\otimes X2\otimes X3)\nabla NG12)\nabla NG13)\nabla NG23$ と展開しても，同じ結果が得られる．制約条件のコーディングとして許容組合せを使用するのか，禁止組合せを使うのかは，それらの集合を記述した時の要素数の少ない方を使用するのがよい．というのは，要素数が少なければ，その集合を表現するZBDDのノード数が減り，かつ，多くの演算の実行時間はノード数にほぼ比例するからである．

4. BDD構築上の問題点

制約充足問題は，BDDの解法では論理積としてコーディングされ，ZBDDの解法では直積と制限と除外演算によってコーディングされる．これらのコーディングは，組合せ生成器 (generator) と制約判定器 (constraint-tester) という2種類の部分から構成されているとみなすこともできる．図2の具体的なコーディングは以下のように分類できる．

	BDD	ZBDD
組合せ生成器	$G=X1\wedge X2\wedge X3$	$G=X1\otimes X2\otimes X3$
制約判定器	$G\wedge C1\wedge C2\wedge C3\wedge C12\wedge C23\wedge C31$	$G\Delta(C12\otimes C23\otimes C13)$

コーディングされた制約充足問題からBDDを構築する方法は，論理積を取る順序によって何種類もある．また，同様にZBDDでも定理1によってどのように展開するかに応じて，さまざまなZBDDを構築する方法がある．ここで，図2に示した制約充足問題に対するBDDを構築する時の制約順序と変数順序の及ぼす影響を調べてみる．ただし，中間ノード数の違いを明らかにするために，各変数の値域には無関係な値を5個ずつ加えておいた．式 (12) の左側から順に論理積を取っていく制約順序と式 (13) の左側から順に論理積を取っていく制約順序との比較を**表1**に示した．変数順序は，最終ノード数が最小になる変数順序（最適変数順序）および，制約順序を取っていく順から決まる変数順序を使用した．なお，式 (13) で示した制約順序はCCVO[19]というヒューリスティクスによって決めたものである．

表1 制約順序・変順序のBDDノード数への影響
Table 1 Number of nodes in BDD affected by constraint and variable order.

制約順序	式(12)による		式(13)による
変数順序	最適変数順序	制約順序から決まる変数順序	
最大中間ノード数	101	41	31
最終ノード数	24	30	30

表2 制約順序のZBDDノード数への影響
Table 2 Number of nodes in ZBDD affected by constraint order.

制約順序	式(14)による	式(15)による
最大中間ノード数	33〜63	22
最終ノード数	5	5

ZBDDによる集合を用いたコーディングでは，解 $\{b\times d\times f, b\times e\times f\}$ を表現するZBDDのノード数は5か6のいずれかであり，ノード数が5となる変数順序は複数ある．その各々に対する最大中間ノード数は一定ではないので，**表2**では範囲を示した．中間ノード数の最大数は，BDD同様に制限演算の適用の順序によって異なることが分かる．

以上示したように，BDDやZBDDを構築する時には，中間に作成されるノードをどのように抑えるかということがポイントとなる．これは，BDDを回路の設計や検証に用いる場合にも遭遇する問題であり，また，制約充足問題や組合せ問題解法での組合せ爆発をどう避けるかと同じ問題でもある．

4.1 BDDと制約充足問題解法との関連

一般に，BDD, ZBDDに限らず，さまざまな方法による制約充足問題の解法は，組合せ生成器と制約判定器から構成されるととらえることができる．組合せ生成器と制約判定器をどのような順序で適用するかについては，後ろ戻り型探索 (backtracking) やさまざまな前向きチェック (forward checking), などさまざまな方法が提案されている．また，一般的な手法だけでなく，問題領域に応じた方法も提案されており，また，

これらの定性的な解析も行われている．

定性的な解析に用いられる特徴として一貫性（consistency）が知られている．一貫性は大域的な一貫性と局所的な一貫性に分類できる．

4.2 大域的な一貫性

効率化手法を評価するために制約ネットワークを使用する．制約充足問題の変数はノードに対応させ，2変数の制約条件があればそれらのノードを接続するアーク（枝）を作成することによって，制約充足問題は制約ネットワークに変換できる．ノードに与える値が制約条件をどの程度満足するかという性質は一貫性（consistency）と呼ばれる．次のような一貫性が知られている[9]．

- **ノード一貫性（1・一貫性）**：ノードに与えられる値はそのノードに関する1変数の制約条件を満足する．
- **アーク一貫性（AC，2・一貫性）**：任意のアークに対して，それらのノード値が2変数の制約条件を満足する．
- **パス一貫性（3・一貫性）**：2つの接続したアークの両端の2つのノードのいかなる値のペアに対しても，必ず中間のノードで2変数の制約条件を満足する値がある．以上の一貫性はたかだか2変数の制約条件しかないものと仮定している．これを一般化すると，次の一貫性になる．
- **k・一貫性**：$k-1$個のノード間で$k-1$変数の制約条件が満足されれば，k変数の制約条件を満足する新たなノードの値が存在する．
- **強k・一貫性**：k項までのすべてのi・一貫性が成立する．

BDDによる問題のコーディングは論理積だけで行われる．今，ある時点までで論理積を取った制約条件について，各制約条件をコーディングする論理式に現れる変数の個数の最大値をkとしよう．各時点で，強k・一貫性が成立している．言い換えると，BDDでは適用された制約条件は以降ずっと達成されていることになる．このような一貫性維持を**単調一貫性維持**（monotonic consistency maintenance）と呼ぶ．制約順序は，論理積を取る順番を指定し，制約順序が異なると，各時点でいつも成立している強k・一貫性のkが異なるだけであった．しかも，kの値は制約条件を適用するごとに単調に増えていく．CCVOというヒューリスティクスは制約条件間の共通変数という相関度を使用して制約順序を定めていた．これは，kの値ができるだけ増えないようにして論理積を取るようにするヒューリスティクスであるといえる．

ただ，どのような制約順序をとっても中間ノード数が爆発する場合には，その対策の1つとして部分問題に分割して解く分割統治法を使うことが考えられる[19]．たとえば，論理積と論理和との交換則によって，$(X_1 \vee \cdots \vee X_n) \wedge Y$を$(X_1 \wedge Y) \vee \cdots \vee (X_n \wedge Y)$と展開する．次に得られた論理積形の部分論理式を部分問題とし，これらの部分問題を解き，部分解の論理和を取れば，解が得られる．うまく分割ができれば，その効果は大きい．

ZBDDでは，制限演算や除外演算を1回だけ適用する場合にはそれで一気に強n・一貫性が達成される．（ただし，nは問題に出現する変数の個数である．）しかし，定理1や2を用いて展開をし，複数の制限演算や除外演算を適用する時には単調一貫性維持が成り立たなくなるので，注意が必要である（例，定理2（10））．この理由は，制限演算（$F\Delta C$）や除外演算（$F\nabla C$）では，Fの要素とは無関係な要素がCに含まれていた場合にはそれらの要素は無視されるからである．$\mathrm{var}(F)$をFに含まれる変数の集合とする*と，$C\Delta\mathrm{var}(F)$はCから$\mathrm{var}(F)$のいずれも含まないような要素を除いた集合となる．すると，

定理3

$$F\Delta C \equiv F\Delta(C\Delta\mathrm{var}(F)) \quad (16)$$

$$F\nabla C \equiv F\nabla(C\Delta\mathrm{var}(F)) \quad (17)$$

が成立し，Cとしていくら一般的な制約条件を与えても，制限演算では，無関係な要素は無視されることになる．これが，BDDの論理式によるコーディングと異なる点となっている．さらに，CがFと関連するものだけでよい，という事実から，集合によるコーディングでは，段階的にプログラムができることになる．

4.3 局所的な一貫性

一般に一貫性のレベルが高くなっても，解を求める効率が改善されるわけではない．問題に応じて，効率が最もよくなる一貫性のレベルというトレードオフが存在する．多くの問題では，アーク一貫性が追求されることが多い．アーク一貫性よりも弱い特徴として，さまざまな局所的なアーク一貫性が提案されている[16]．

- **AC1/5**：変数に新しい値を与える時に，それ以前の状態と矛盾するかをチェックする．例は，後ろ戻り法．
- **AC1/4**：変数に新しい値を与える時に，AC 1/5に加えて，それが次に値を与える変数と矛盾しないかをチェックする．例は，前向きチェック．

* ただし，$\mathrm{var}(\{\epsilon\}) = \phi$

- **AC1/3**：変数に新しい値を与える時に，AC 1/4に加えて，それがアークで隣接する変数のうちまだ値の定まっていないいずれの変数とも矛盾しないかをチェックする．例は，部分的先読み．
- **AC1/2**：変数に新しい値を与える時に，AC 1/3に加えて，それが間接的にでも関係のあるまだ値の定まっていない変数と矛盾しないかをチェックする．例は，完全先読み（Full Lookahead, FL）．
- **AC**（アーク一貫性）：変数に新しい値を与える時に，AC 1/2に加えて，それがまだ値の定まっていないすべての変数と矛盾しないかをチェックする．

前述したように，BDDを構築する時には組合せ爆発の防止が重要な課題である．最終ノード数は変数順序に依存するので，それを最小にする変数順序を求めるヒューリスティクスが数多く提案されてきた（最適順序を求めるのはNP完全）．奥乃は組合せ爆発が，制約順序（制約条件を組み合わせる順序）に依存し，また，制約順序から最適な変数順序が決まることを指摘し，さらに，制約順序と変数順序とを求めるヒューリスティクスCCVO（Correlation-based Constraint and Variable Ordering）法を提案している[19]．CCVO法では，制約条件間に共通する変数の個数をもとに定義した制約条件間の相関関数を使用する．これは，関連する変数すべてに対して制約条件を適用するので，局所アーク一貫性AC 1/2に相当する．

ZBDDでは，組合せ的爆発を避けるために定理1や2に示した展開規則を用いて部分問題に分割してから制限演算や許容演算を適用することが多い．この時に，制約条件は不要な要素を抜いた$C\Delta\mathrm{var}(F)$に強制的に変換されるので，先読みの一切ないAC 1/5に相当する一貫性アルゴリズムとなる．

4.4 ATMSとの比較

ATMS（Assumption-based Truth Maintenance System）は，命題ホーン節を制約条件（正当化と呼ばれる）として，複数の解を保持する真偽維持システムである．ATMSでは，ホーン節しか扱えないので，一般の論理式を扱うために論理和や論理否定をホーン節でコーディングし，完全性を保証するためにさまざまなメタルールを導入している[5),6)]．de Kleerは制約充足問題をホーン節の集合にコーディングし，ATMSを使用する解法を提案している[7)]．

3章の例を取り上げ，ATMSでのコーディングを説明しよう．まず，制約充足問題の変数と値の対に対してATMSでの命題シンボル$x_{1:a}$を定義する．変数の値域は，論理和を用い$x_{1:a}\vee x_{1:b}$という正当化でコーディングし，各変数が1つしか値が持てないという制約条件はATMSでは2つの値の禁止組合せとして$\overline{x_{1:a}}\vee\overline{x_{1:b}}$という正当化でコーディングする．制約条件は，ATMSの正当化$\overline{x_{1:a}}\vee\overline{x_{2:e}}$としてコーディングする．これらの条件を満足する充足解は，ラベルの更新と（一般節を扱うための）メタルールによって求める．

さらに，さまざまな大域的な一貫性がATMSでのメタルールに相当することを主張している．しかし，この対比が成立するのは，制約充足問題がATMSに翻訳できる場合だけであり，すべての制約充足問題がATMSに翻訳できるわけではない．

BDDは一般の節がそのまま表現でき，かつ，複数の解が同時に保持できるのでATMSの概念的な拡張となっている．ATMSを一般節が扱えるように拡張したCMS（Clause Management System）が提案されているが，よい処理技法がなかったため，従来あまり注目を浴びてこなかった．しかし，BDDを使用すればCMSの実用的で効率的な処理系が構築できると期待される[18)]．

5. 2種類のBDDによるプログラミングの比較

本章では，BDD（算術論理式）とZBDD（集合）という2種類のBDDによってさまざまな組合せ問題を解くことを通じて，両者のプログラミングを表現法，実行時間の観点から比較する．

5.1 N-Queens問題

盤面(i,j)を$x_{i,j}$で表現し，そこに女王が置かれている時1，それ以外は0を取ることにする．BDDプログラミングでは，組合せ生成器部は3.2節と同じようにコーディングする．各行，列に女王を1つだけ置けるという制約条件は$X_{i,1}+\cdots+x_{i,n}==1$のように表現する．また，対角線上にはたかだか1つの女王しか置けないという制約条件は，$x_{3,1}+x_{2,2}+x_{1,3}<2$のように表現する．N-Queens問題は，組合せ生成器部とこれら2種類の制約判定器部との論理積としてコーディングできる．制約条件の論理積を取る順序はCCVO[19)]というヒューリスティクスで決定した．

ZBDDプログラミングでは，組合せ生成器部はすべての変数からn個取った組合せの集合ではなく，各列ごとの候補集合$X_i=\{x_{1,i},\cdots,x_{n,i}\}$の直積としてコーディングする．このコーディングには，同じ列に女王が1つしか置けないという制約条件がすでに含まれている．制約判定器部は，女王の取り合う位置の対の集合を禁止組合せ集合Cとして除外演算を使用してコーディングする．すなわち，$G=(\otimes_i X_i)\nabla C$でN-Queens問題はコーディングできる．この式を忠実に

実行すると，組合せ爆発が容易に生じるので，組合せ生成器と制約判定器の実行順序を定理1,2を用いて変更する．

この制約条件を展開し，列の順に女王を置き，その制約条件を適用していく．G_{i-1} を第1列から第 $i-1$ 列まで制約条件を満足するように配置した女王の集合とする．i 列目の各盤面に対して，

$$G_1=X_1$$
$$G_i=(G_{i-1}\otimes X_i)\nabla C$$

G_i の式は定理1, 2, 3を使用してさらに変形できる．

$$G_i=(G_{i-1}\otimes X_i)\nabla(C\Delta(\mathrm{var}(G_{i-1}\otimes X_i)))$$

$\mathrm{var}(X_i)=X_i$ より

$$G_i=(G_{i-1}\otimes X_i)\nabla(C\Delta(\mathrm{var}(G_{i-1})\cup X_i))$$
$$=(\bigcup_j((G_{i-1}\times x_{i,j})\nabla(C\Delta(\mathrm{var}(G_{i-1})\cup X_i))))$$
$$\nabla(C\Delta(\mathrm{var}(G_{i-1})\cup X_i))$$

C の要素は2変数の組合せ集合だけだから

$$G_i=\bigcup_j((G_{i-1}\times x_{i,j})\nabla((C\div x_{i,j})\cap\mathrm{var}(G_{i-1}))$$
$$=\bigcup_j(x_{i,j}\times(G_{i-1}\nabla((C\div x_{i,j})\cap\mathrm{var}(G_{i-1}))))$$

すなわち，

$$F_j=x_{i,j}\times(G_{i-1}\nabla\{\overbrace{x_{1,j},\cdots,x_{i-1,j}}^{\text{同じ列の位置}},$$
$$\underbrace{x_{i-1,j-1},x_{i-2,j-2},\cdots}_{\text{右下がり対角線上の位置}}\ \overbrace{x_{i-1,j+1},x_{i-2,j+2},\cdots}^{\text{右上がり対角線上の位置}}\})$$

を計算し，最後にその列で可能な配置の集合を $G_i=F_1\cup\cdots\cup F_n$ で計算すれは，G_n で解が求まる．

2つの解法の結果を**表3**に示す．表中のノード数は解に対するBDD，ZBDDの最終ノード数である．ZBDDでは，集合の直積（⊗）を使用しないので，むだなノードはほとんど作成されない．しかし，大きなNでは，集合演算でのオーバヘッドのためにZBDDのBDDとの相対的な実行速度は段々低下し，1に近付く．

表3 N-Queens実行時間
Table 3 Execution times of N-queens problems.

女王	解の個数	BDD ノード数	BDD 時間	ZBDD ノード	ZBDD 時間	ノード比
4	2	29	0.12	8	0.024	3.6
5	10	166	0.17	40	0.070	4.2
6	4	129	0.36	24	0.133	5.4
7	40	1098	0.95	186	0.33	5.9
8	92	2450	3.50	373	1.03	6.6
9	352	9556	6.85	1309	3.72	7.3
10	724	25944	23.8	3120	14.8	8.3
11	2680	94821	97.2	10503	6.1	9.0
12	14200	435169	301	45833	285	9.5
13	73712	2044393	11052†	204781	1295	10.0

（すべての時間の単位は秒）．

BDDのプログラムは文献[19]と同じものを使用した．実行環境は，SUN SS10（hyperSPARC 66MHz Dual-CPU）であり，主記憶は128 MAyteである．†主記憶不足のため分割統治法で解いたために遅くなっている．主記憶が十分にある場合は1300秒程度と予想される[19]．

N-Queens問題の効率のよい解法が知られている．たとえば，Dijkstraは行列を使用して女王がお互いに取り合う位置にあるのかを調べる方法を考案している．この方法を使用し，本稿と同じ計算機環境でLucid Common Lispを用いて1つの解をリストで表し，すべての解を求めた．この解法の実行時間は，12-Queens問題で101秒，13-Queens問題で650秒であった．これと比べると，BDD，ZBDDでの解法の時間は，12-Queens問題で約3倍，13-Queens問題で約2倍要する．このオーバヘッドが妥当なものかの判断は，全解の表現を使用する応用プログラムに依存するであろう．

5.2 魔方陣の問題

n 次の魔方陣とは，$n\times n$ の盤面に1から n^2 の数を並べた時に，縦，横，斜めの数の和がすべて等しいものである．4次の魔方陣では，探索空間は16！（約 2.1×10^{13}）対称解を排除してもその1/8にしか縮小されない．

算術論理式（BDD）を使用すると，簡潔にコーディングできる．4次の魔方陣の盤面 (i, j) に置く数を $P_{i,j}$ とすると，$P_{i,j}=x_{i,j,1}\times1\vee\cdots\vee x_{i,j,16}\times16$ とコーディングできる．変数 $x_{i,j,k}$ は数 k が盤面 (i, j) に置かれる時に1となる．盤面に数を1つだけ置くという制約条件は $x_{i,j,1}+\cdots+x_{i,j,16}==1$ でコーディングし，1から16までの数をどこかの盤面に置くという制約条件も同様に与える．各行，列，対角線上の数の和が等しいという制約条件は，$P_{1,1}+P_{2,1}+P_{3,1}+P_{4,1}==34$ とコーディングする．制約条件の論理積を取る順序はCCVO法で決定する．

組合せ集合（ZBDD）を用いると，各盤面に置かれる数 $P_{i,j}$ は $P_{i,j}=\{x_{i,j,1},\cdots,x_{i,j,16}\}$ で表現される．BDDと同様に盤面に置かれる数についての制約条件は禁止組合せとして表現する．これらの直積をNGとする．各

表4 魔方陣プログラムの実行時間
Table 4 Execution time of magic square problems.

魔方陣	解の個数	BDD[19] ノード数	BDD[19] 時間(秒)	ZBDD ノード数	ZBDD 時間
3	1	81	6.58	9	0.45
4	880	109999	1299	7646	345

行，列，対角線上の数の和が等しいという条件のコーディングは，算術論理式ほど簡単ではない．まず，和が等しくなる変数の組合せをすべて求める．それをCとする．求める解は$(((P_{1,1}\otimes\cdots\otimes P_{4,4})\nabla NG)\Delta C)$で得られる．直積，△，▽を取る計算順序は，CCVO法によって求める．

2つの解法の実行時間と解に対する最終ノード数を**表4**に示す．表にはないが，BDDでは計算途上での最大ノード数が最終ノード数の20倍の約$2.3M$個と極めて大きく，組合せ爆発への対策が重要である．4次の魔方陣のBDDによる解法では，論理積と論理和との交換則による分割統治法により問題分割を行って求めた[19]．

BDDの方が解に対するノード数が多いのは，ZBDDでは集合しか表現されていないのに対して，BDDでは制約条件も非明示的に解のBDDの中に表現されているからである．この結果，以下のような機能がたかだかBDD中のノード数に比例する計算量で実現することができる．

（1）他の方法で求めた解のチェック，
（2）問題の別のコーディングのチェック，
（3）さらに制約条件を付加した問題を解く(例：すべてのトーラスで考えた疑似対角線上の数の和が等しい魔方陣)，
（4）定理の証明(例：4次の魔方陣の内側の正方形の数の和が34)，Gを解のBDDとすると，$G\supset(P_{2,2}+P_{2,3}+P_{3,2}+P_{3,3}==34)$が恒真（[1]）かを調べる．
（5）解の内包表現と外延表現の比較．

5.3 その他の問題

算術論理式の記述能力は高く，覆面算，虫食い算，最小木問題などの組合せ問題は容易にコーディングできる．たとえば，$SEND+MORE=MONEY$という覆面算は，変数を$S=s_1\times 1\vee\cdots\vee s_9\times 9$と表現し，$(S\times 1000+E\times 100+N\times 10+D)+(M\times 1000+O\times 100+R\times 10+E)==(M\times 10000+O\times 1000+N\times 100+E\times 10+Y)$とコーディングする．これからBDDを構築すれば，すべての解が見つかる．同じ問題を通常の論理式，あるいは，集合でコーディングするのは算術論理式の場合ほど単純ではない．

また，算術論理式は三面図から立体の復元問題などにも威力を発揮する．正木らは，三面図を算術論理式で表現し，それをもとに立体を復元し，さらに，立体の認識用の表現を生成するのにも応用している[10]．

5.4 制限演算，除外演算の効用

前述したように制限演算の実装法には，(1)式で定義される繰り返し型と(2)式で定義される再帰型の2つがある．本節では，両者の実装法を評価するために，制限演算と除外演算を組み込み関数で実行した場合と，繰り返し型で実行した場合の比較を行う．まず，N-Queens問題における組み込み制限演算の効果を評価する．**図3**に示したデータは，すべての制限演算を組み込み関数で実行した時と繰り返し型で実行した時の速度比である．再帰型の方が繰り返し型よりもNが増えるに従ってより速くなり，13-Queensでの速度比は4.5倍である．魔方陣の場合には，許容演算を実現する制限演算を組み込み関数で実行した場合の結果を**表5**に示す．再帰型の方が繰り返し型よりも2.1倍早い．表4に示したデータも再帰型による実行結果である．これらのベンチマークに関する限り，ハッシュ表がうまく機能しており，その結果再帰型の方が繰り返し型よりも効率のよいことが分かった．

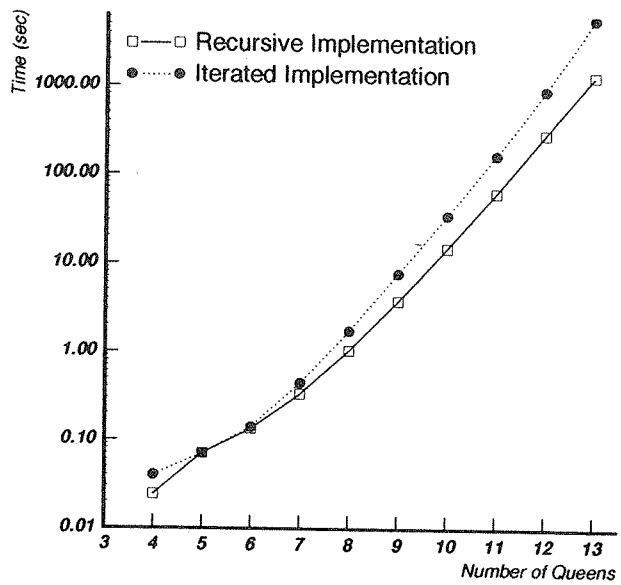


図3 N-Queensにおける制限演算の効果
Fig. 3 Effects of restrict operations for N-Queens

表5 制限演算による魔方陣プログラムの実行時間(ZBDD)
Table 5 Exection time of magic square problems with restrict operations (ZBDD).

魔方陣	組込み(再帰型)		繰り返し型	
	最大ノード数	時間(秒)	最大ノード数	時間
3(全解)	209	0.76	1104	4.18
3(非対称解)	155	0.45	622	0.88
4	83852	345	489098	702

なお，表2のBDDでの最大ノード数は約2.3M個である．

6. おわりに

本稿では，制約充足問題の解法という立場からBDDとZBDDをとらえ，それらによる解法を提案した．ZBDDでは，組合せ集合演算として制限演算と除外演

算という新たな演算を導入し，それらが制約充足問題を解く時に重要な役割を果たすことを示した．BDDでは，一貫性が計算とともに段々増えていく単調一貫性維持という性質が成り立つことを示した．ZBDDでは，制限演算や除外演算を分解することにより，制約条件もより小さくすることができることを示した．この性質からBDDでは苦手な段階的解法がZBDDでは容易であることが分かった．さらに，具体的な例として，N-Queens問題，魔方陣を取り上げ，2種類のBDDで解き，その時に時間，空間使用量について評価を行った．また，制限演算と排除演算の再帰的な処理と繰り返し的な処理との比較を行い，再帰的な処理の方が効率がよいことを示した．

また，BDDやZBDDと多重文脈型真偽維持システムATMSとの比較も行った．ATMSでは扱える論理式が命題ホーン節に限定されたためにコーディング技法とメタルールが必要であったが，BDDではそのようなものが一切不要である．BDDがATMSを拡張したCMSに使用できることが分かったので，今後，さらに検討を進める．本稿では，BEM-II（BDD＋算術論理式）[13]とUCC（ZBDD）[15]，および，Common Lispを用いた．システムレベルでの今後の課題としてはより柔軟なインタフェースの設計と実装が挙げられる．

謝辞 最後に，ご討論いただいた第3回BDDセミナーの参加者に感謝します．とくに，富士通研究所の松永裕介氏からは，制限演算とCoudertたちが提案するSupSet演算との類似性を指摘していただいた．また，ご討論いただいた東京大学工学部石塚満教授，正木寛人氏，下國治氏，本稿および記号の用法についてコメントをいただいたNTT基礎研究所磯崎秀樹主任研究員に感謝します．

参考文献

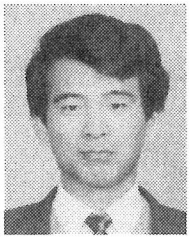
1) Akers, S. B.: Binary Decision Diagrams, *IEEE Trans. Comput.*, Vol. C-27, No. 6, pp. 509-516 (1978).
2) Bryant, R. E.: Graph-based Algorithm for Boolean Function Manipulation, *IEEE Trans. Comp.*, Vol. C-35, No. 5, (1986).
3) Coudert, O., Madre, J. C. and Fraisse, H.: A New Viewpoint on Two-Level Logic Minimization, *Proc. of 30th ACM/IEEE Design Automation Conference*, pp. 625-630 (1993).
4) Coudert, O.: Doing Two-Level Logic Minimization 100 Times Faster, *Proc. of ACM-SIAM SODA*, pp. 112-121 (1995).
5) de Kleer, J.: An Assumption-based TMS, *Artificial Intelligence*, Vol. 28, pp. 127-162 (1986).
6) de Kleer, J.: Extending the ATMS, *Artificial Intelligence*, pp. 163-196 (1986).
7) de Kleer, J.: A Comparison of ATMS and CSP Techniques, *Proc. of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pp. 290-296 (1989).
8) Forbus, K. and de Kleer, J.: *Building Problem Solvers*, The MIT Press (1993).
9) Kumar, V.: Algorithms for Constraint-Satisfaction Problems: A Survey, *AI MAGAZINE*, Vol. 13, No. 1, pp. 32-44 (1992).
10) 正木，奥乃，石塚：二分決定グラフによる三面図からの3Dモデルの解釈，人工知能学会第8回全国大会（1993）．
11) Madre, J. C. and Coudert, O.: A Logically Complete Reasoning Maintenance System, *Proc. of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pp. 294-299 (1991).
12) 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).
13) Minato, S.: BEM-II: An Arithmetic Boolean Expression Manipulator Using BDDs, 電子情報通信学会，英文誌，Vol. E76-A, No. 10, pp. 1721-1729 (1993).
14) Minato, S.: Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems, *Proc. of the 30th ACM/IEEE Design Automation Conference*, pp. 272-277 (June 1993).
15) Minato, S.: Calculation of Unate Cube Set Algebra Using Zero-Suppressed BDDs, *Proc. of the 31st ACM/IEEE Design Automation Conference*, pp. 420-424 (1994).
16) Nadel, B.: Tree Search and Arc Consistency in Constraint-Satisfaction Algorithms, in *Search in Artificial Intelligence*, eds. Kanal, L. and Kumar, V., Springer (1988).
17) 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).
18) 奥乃 博，下國 治，田中英彦：二分決定グラフ（BDD）による多重推論型真偽維持システムの実現．第2回「知識のリフォーメーション」シンポジウム，情報処理学会（Nov. 1993）．
19) 奥乃 博：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爆発抑制法．情報処理学会論文誌，Vol. 35, No. 5, pp. 739-753 (1994).

（平成7年1月10日受付）
（平成7年4月14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．同年電電公社入社．1986～88年スタンフォード大学客員研究員．1992～93年東京大学工学部客員助教授．現在，NTT基礎研究所勤務．主幹研究員．推論機構，音環境理解の研究に従事．本学会英文図書委員，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．著書：『知的プログラミング』（共著，オーム社，1993），『マルチエージェントと協調計算III』（編，近代科学社，1994）．



湊 真一（正会員）

1965年生．1988年京都大学工学部情報工学科卒業．1990年同大学院修士課程修了．同年日本電信電話(株)入社．以来，同社LSI研究所において，LSI論理設計システムの研究に従事．特に論理関数の処理手法に興味を持つ．現在同社研究主任．1994～95年京都大学大学院在学．工学博士．1992年情報処理学会全国大会奨励賞．IEEE，電子情報通信学会各会員．