論 文

ハッシングの技法を用いた数式処理アルゴリズムとその擬似ブール計画法への応用*

今井正治** 吉田雄二*** 福村晃夫**

Abstract

In this paper we propose some algorithms for formula manipulation of Boolean functions and pseudo-Boolean functions. They are based upon the hashing technique. It takes $O(m^2)$ computation time to simplify a (pseudo-) Boolean function using conventional algorithms, where m is the number of terms involved in the function. On the other hand, our algorithms can reduce the computation time to $O(m)$. The performance is attained by two facts.

The one is that the size of the hash table is adaptively changed according to the number of terms involved in the function processed, and the other that the test whether a term consists of a single variable or not is made quite easily during the process to hash the term into the table.

The performance of the algorithms is checked through some experiments, and also the algorithms are successfully applied to the Pseudo-Boolean Programming algorithm.

1. まえがき

ブール関数，あるいは擬似ブール関数****を用いて与えられる問題を，計算機で解くとき，これらの関数を簡単化するための数式処理がしばしば必要となる．従来の方法によれば，この簡単化は，たとえば積和形のブール関数では関数を構成する積項どうしの比較にもとづいて行われることになるが，この方法では比較回数が項数 (m) の2乗に比例するため処理に要する時間も $O(m^2)$ となって大規模な問題を扱うのは困難となる．

本論文では，この点を考慮して，ハッシングの技法を用いた数式処理アルゴリズムを提案し，それを擬似ブール計画法[3]に適用することを試みる．このアルゴリズムではハッシュ・テーブルの大きさが式の項数に応じて動的に変更されることで，処理時間が $O(m)$ となり，しかも，ハッシュ関数として特別な形のものが採用されることで積項が単一変数からなるか否かの判定が高速化される，などの理由により，本アルゴリズムは，特に大規模な数式処理に適していると考えられる．

以下では，初めに **2.** で，ここで扱われるブール関数および擬似ブール関数を定義し，それらの計算機記憶装置内部での表現（以下では内部表現と呼ぶ）の方法について述べる．**3.** では，ハッシングの技法を用いた数式処理アルゴリズムと，従来の数式処理アルゴリズムとについて述べ，**4.** では，これらのアルゴリズムによる数式の処理時間の理論的評価を行う．最後に，**5.** では，これらのアルゴリズムによる数式処理の時間を測定した結果と，これらのアルゴリズムを用いて実現された擬似ブール計画法のアルゴリズムの効率の比較結果とについて報告する．

2. ブール関数，擬似ブール関数およびそれらの内部表現

ここでは初めに，以下で扱われるブール関数および

---

* Algorithms for Formula Manipulation based upon Hashing Technique and their Applications to Pseud-Boolean Programming, by Masaharu IMAI, Teruo FUKUMURA (Faculty of Engineering, Nagoya University) and Yuuji YOSHIDA (Computation Center, Nagoya University)

** 名古屋大学工学部情報工学科

*** 名古屋大学大型計算機センター

**** 擬似ブール関数の定義は 2.1 に与えられる．

擬似ブール関数を定義し，それらが擬似ブール計画法の中で占める役割について述べる．次にこれらに対して，効率的な処理を目的とした内部表現を与える．

2.1 ブール関数と擬似ブール関数の処理

$L_2 \triangleq \{0,1\}$, $L_2^{L_2} \triangleq \{0,1\}^*$ として，$x_j, (j=1,2,\cdots,n)$ を L_2 上の変数，$\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ を $L_2^{L_2}$ 上の変数ベクトルとする．

〔**定義 1**〕 ここで扱われるブール関数 $F(\boldsymbol{x})$ は次のように定義される．

$$\begin{cases} F(\boldsymbol{x}) \triangleq \bigvee_{i=1}^{s} F_i(\boldsymbol{x}), \\ F_i(\boldsymbol{x}) \triangleq \prod_{j=1}^{n} x_j^{\varepsilon_{ij}}, \ (i=1,2,\cdots,s). \end{cases} \tag{1}$$

ここに $\prod$ は論理積を表わす．また，$x_j^{\varepsilon_{ij}}, (i=1,2,\cdots,s;\ j=1,2,\cdots,n)$ は，$\varepsilon_{ij} \in \{0,1,t,f\}$ として，次式で定義される．

$$x_j^{\varepsilon_{ij}} \triangleq \begin{cases} \bar{x}_j & \text{if} \quad \varepsilon_{ij}=0, \\ x_j & \text{if} \quad \varepsilon_{ij}=1, \\ 1 & \text{if} \quad \varepsilon_{ij}=t, \\ 0 & \text{if} \quad \varepsilon_{ij}=f. \end{cases} \tag{2}$$ ☒

ここに $\bar{}$ は否定を表わす．

〔**定義 2**〕 ここで扱われる擬似ブール関数 $y(\boldsymbol{x})$ は次のように定義される．

$$\begin{cases} y(\boldsymbol{x}) \triangleq \sum_{i=1}^{m} a_i Y_i(\boldsymbol{x}) \\ Y_i(\boldsymbol{x}) \triangleq \prod_{j=1}^{n} x_j^{\delta_{ij}}, \ (i=1,2,\cdots,m). \end{cases} \tag{3}$$

ここに，$a_i, (i=1,2,\cdots,m)$ は整数，$x_j^{\delta_{ij}}, (i=1,2,\cdots,m;\ j=1,2,\cdots,n)$ は，$\delta_{ij} \in \{0,1\}$ として，次式で定義される．

$$x_j^{\delta_{ij}} \triangleq \begin{cases} 1 & \text{if} \quad \delta_{ij}=0, \\ x_j & \text{if} \quad \delta_{ij}=1. \end{cases} \tag{4}$$ ☒

擬似ブール計画法のアルゴリズム[3]は，ブール制約条件 $F(\boldsymbol{x})=0$ のもとで，擬似ブール関数で与えられる目的関数 $y(\boldsymbol{x})$ を最小化するが，その過程で，これらの数式に対して，次の処理が必要である．

（1） ブール関数の処理

式(1)で定義されるブール関数 $F(\boldsymbol{x})$ に含まれる変数の一部に 0 または 1 を代入して得られる式を，

$$\tilde{F}(\boldsymbol{x}) \triangleq \bigvee_{i=1}^{s} \tilde{F}_i(\boldsymbol{x}), \tag{5}$$

として，次の(ⅰ)～(ⅲ)をブール関数の処理と呼ぶ．

(ⅰ) $\tilde{F}(\boldsymbol{x})$ を求めること，

(ⅱ) $\tilde{F}_i(\boldsymbol{x}), (i=1,2,\cdots,s)$ が単一の変数からなるか否かの判定*，

(ⅲ) $\tilde{F}(\boldsymbol{x})$ の簡単化，すなわち，

$$\tilde{F}_i(\boldsymbol{x}) \not\equiv \tilde{F}_j(\boldsymbol{x}) \quad \text{if} \quad i \neq j,\ i,j=1,2,\cdots,s$$

となるようにすること**．

（2） 擬似ブール関数の処理

式(3)で定義される擬似ブール関数 $y(\boldsymbol{x})$ に含まれる変数の一部に 0 または 1 を代入して得られる式を，

$$\tilde{y}(\boldsymbol{x}) \triangleq \sum_{i=1}^{m} a_i \tilde{Y}_i(\boldsymbol{x}), \tag{6}$$

として，次の(ⅰ)，(ⅱ)を擬似ブール関数の処理と呼ぶ．

(ⅰ) $\tilde{y}(\boldsymbol{x})$ を求めること，

(ⅱ) $\tilde{y}(\boldsymbol{x})$ の簡単化，すなわち，

$$\tilde{Y}_i(\boldsymbol{x}) \not\equiv \tilde{Y}_j(\boldsymbol{x}) \quad \text{if} \quad i \neq j,\ i,j=1,2,\cdots,m,$$

となるようにすること．

2.2 式の内部表現

定義1，定義2によって定義されたブール関数および擬似ブール関数を計算機で処理するためには，それらを記憶装置内部で表現する必要がある．以下では，ここで用いた表現方法について述べる．

式(1)で与えられるブール関数 $F(\boldsymbol{x})$ は，これを構成する積項 $F_i(\boldsymbol{x}), (i=1,2,\cdots,s)$ の各々に対して，式(7)で定義される2つの整数 σ_i, τ_i の順序対（以下では簡単に対という），(σ_i,τ_i) を対応させることにより，これらの並びとして式(9)のように表わされる．

$$\begin{cases} \sigma_i \triangleq \sum_{j \in I_i} 2^{j-1}, \\ \tau_i \triangleq \sum_{j \in J_i} 2^{j-1}. \end{cases} \tag{7}$$

ここに，$I_i, J_i, (i=1,2,\cdots,s)$ は，$I \triangleq \{1,2,\cdots,n\}$ として，次式で定義される．

$$\begin{cases} I_i \triangleq \{j \mid j \in I, (\varepsilon_{ij}=1) \vee (\varepsilon_{ij}=0)\}, \\ J_i \triangleq \{j \mid j \in I, (\varepsilon_{ij}=1) \vee (\varepsilon_{ij}=f)\}. \end{cases} \tag{8}$$

$$\{(\sigma_1,\tau_1),(\sigma_2,\tau_2),\cdots,(\sigma_s,\tau_s)\}. \tag{9}$$

式(7)に示される整数の対の並びは，記憶装置内部では，整数型2次元配列などで表現される．以下では σ_i, τ_i の2進数表現をそれぞれ，$\boldsymbol{\sigma}_i, \boldsymbol{\tau}_i$ と表わすことにしよう．たとえば，積項 $F_i(\boldsymbol{x})=x_1x_2\bar{x}_3x_4x_5$ に対して，$x_2=x=1$ と値が与えられているとき，計算機内部では，この積項は，$\boldsymbol{\sigma}_i=10101$, $\boldsymbol{\tau}_i=10001$ によって表わされる．

一方，式(3)で与えられる擬似ブール関数 $y(\boldsymbol{x})$ は，

---

* 積項が単一変数になれば，その変数の値が自動的に定まるという理由による．

** $\tilde{F}_i(\boldsymbol{x}) \equiv \tilde{F}_j(\boldsymbol{x})$ で，式 $F_i(\boldsymbol{x})$ と $F_j(\boldsymbol{x})$ の内部表現が等しいことを表わす．この命題が成り立たないことを $\tilde{F}_i(\boldsymbol{x}) \not\equiv \tilde{F}_j(\boldsymbol{x})$ で表わす．なお，ここでいう式の簡単化には，相補律，吸収律などを適用するものは含まれない．

$y(\boldsymbol{x})$ を構成する各積項 $Y_i(\boldsymbol{x})$ に対して，式(10)で定義される整数 η_i と a_i との対の並びとして，式(12)のように表わされる．

$$\eta_i \triangleq \sum_{j\in K_i} 2^{j-1}, \tag{10}$$

ここに，K_i は，次式で定義される．

$$K_i \triangleq \{j \mid j\in I, \delta_{ij}=1\}. \tag{11}$$

$$\{(\eta_1, a_1), (\eta_2, a_2), \cdots, (\eta_m, a_m)\} \tag{12}$$

以下では，η_i の2進数表現を $\boldsymbol{\eta}_i$ で表わす．

また，**3.** に述べるアルゴリズムの途中で現われる，未定要素を含む n 次元ベクトル $\boldsymbol{x}$ に対し，その内部表現として，式(13)で定義される2つの整数 $\boldsymbol{\xi}, \boldsymbol{\theta}$ の対 (ξ, θ) を対応させる．

$$\begin{cases} \xi \triangleq \sum_{j\in M} 2^{j-1}, \\ \theta \triangleq \sum_{j\in N} 2^{j-1}. \end{cases} \tag{13}$$

ここに，M, N はそれぞれ次式で定義される．

$$\begin{cases} M \triangleq \{j \mid j\in I, x_j=u\}, \\ N \triangleq \{j \mid j\in I, x_j=1\}. \end{cases} \tag{14}$$

ただし，$x_j=u$ は，変数 x_j の値が未定であることを示す．

以下，ξ, θ の2進数表現をそれぞれ $\boldsymbol{\xi}, \boldsymbol{\theta}$ で表わすことにしよう．たとえば，$\boldsymbol{x}=(1, 0, u, 1, 0)$ は，$x_1=x_4=1$, $x_2=x_5=0$, $x_3=u$ であることを示すが，この $\boldsymbol{x}$ は計算機内部では，$\boldsymbol{\xi}=00100$, $\boldsymbol{\theta}=10010$ によって表わされる．

2.3 式の処理

$F(\boldsymbol{x}), y(\boldsymbol{x})$ を構成する各項を **2.2** で述べた形で内部表現することによって，積項どうしの比較，積項に含まれる変数の一部への値の代入，積項が単一の変数からなるか否かの判定などは，計算機のビット処理機能を利用することで，以下のように，容易にかつ高速で実行される．

(1) 積項どうしの比較

$F_i(\boldsymbol{x})\equiv F_j(\boldsymbol{x})$ であるか否か，および $Y_i(\boldsymbol{x})\equiv Y_j(\boldsymbol{x})$ であるか否かは，次の関係によって判定される．

(i) $F_i(\boldsymbol{x})\equiv F_j(\boldsymbol{x}) \Leftrightarrow [\sigma_i=\sigma_j]\wedge[\tau_i=\tau_j]$, (15)

(ii) $Y_i(\boldsymbol{x})\equiv Y_j(\boldsymbol{x}) \Leftrightarrow \eta_i=\eta_j$. (16)

ここに，$\Leftrightarrow$ は対等を，$\wedge$ は論理積を表わす．

(2) $F_i(\boldsymbol{x})$ に含まれる変数の一部への値の代入

積項 $F_i(\boldsymbol{x})$ に含まれる変数の一部に値を代入して得られる積項を，

$$\tilde{F}_i(\boldsymbol{x}) \triangleq \prod_{j=1}^{n} x_j^{\varepsilon_{ij}}, \tag{17}$$

で表わす．任意の $j(j=1, 2, \cdots, n)$ について変数 x_j に与える値を c_j とし，x_j に値が与えられないことを $c_j=u$ で表わすことにする．ε_{ij} と c_j との値の組み合わせに対する $\tilde{\varepsilon}_{ij}$ の値は **Table 1** に与えられる．

Table 1

ε_{ij} \ c_j	0	1	u
0 ($\bar{x}_j$)	t	f	0
1 (x_j)	f	t	1
t (1)	t	t	t
f (0)	f	f	f

Table 1 によれば，任意の積項 $F_i(\boldsymbol{x})$ の内部表現から $\tilde{F}_i(\boldsymbol{x})$ の内部表現を導くことができる．たとえば，積項 $x_1\cdot 1\cdot \bar{x}_3\cdot 1\cdot x_5$ に対して，値 $(c_1, \cdots, c_5)=(1, 0, u, 1, 0)$ が与えられたとき，得られる積項は $1\cdot 1\cdot \bar{x}_3\cdot 1\cdot 0$, その内部表現は，$\tilde{\boldsymbol{\sigma}}_i=00100$, $\tilde{\boldsymbol{\tau}}_i=00001$ となる．一般に，$\tilde{F}_i(\boldsymbol{x})$ の内部表現 $\tilde{\boldsymbol{\sigma}}_i, \tilde{\boldsymbol{\tau}}_i$ は，$\tilde{F}_i(\boldsymbol{x})$ の内部表現 $\boldsymbol{\sigma}_i, \boldsymbol{\tau}_i$ および $\boldsymbol{x}$ の内部表現 $\boldsymbol{\xi}, \boldsymbol{\theta}$ から，式(18)によって導びかれる．(付録(1)参照.)

$$\begin{cases} \tilde{\boldsymbol{\sigma}}_i = \boldsymbol{\xi}\wedge\boldsymbol{\sigma}_i, \\ \tilde{\boldsymbol{\tau}}_i = (\boldsymbol{\theta}\wedge\boldsymbol{\sigma}_i)\oplus\boldsymbol{\tau}_i. \end{cases} \tag{18}$$

ここに，$\oplus$ は排他的論理和を表わし，各演算は要素(けた)ごとに行われるものとする．

更に，$\tilde{F}_i(\boldsymbol{x})\equiv 0$ あるいは $\tilde{F}_i(\boldsymbol{x})\equiv 1$ となるか否かは，式(19)によって知られる．(付録(2)参照.)

$$\begin{cases} \text{(i)} \quad \tilde{F}_i(\boldsymbol{x})\equiv 0 \Leftrightarrow (\neg\tilde{\boldsymbol{\sigma}}_i)\wedge\tilde{\boldsymbol{\tau}}_i \neq \boldsymbol{0}, \\ \text{(ii)} \quad \tilde{F}_i(\boldsymbol{x})\equiv 1 \Leftrightarrow \tilde{\boldsymbol{\sigma}}_i=\tilde{\boldsymbol{\tau}}_i=\boldsymbol{0}. \end{cases} \tag{19}$$

ここに，$\neg$ は，各要素ごとの否定を表わし，$\boldsymbol{0}$ は整数0の2進数表現である．

(3) $Y_i(\boldsymbol{x})$ に含まれる変数の一部への値の代入

積項 $Y_i(\boldsymbol{x})$ に含まれる変数の一部に値を代入して得られる積項を，

$$\tilde{Y}_i(\boldsymbol{x}) \triangleq \prod_{j=1}^{n} x_j^{\tilde{\delta}_{ij}}, \tag{20}$$

で表わす．δ_{ij} と変数 x_j に与えられる値 c_j との組み合わせに対する $\tilde{\delta}_{ij}$ の値は **Table 2** に与えられる．積項 $\tilde{Y}_i(\boldsymbol{x})$ の内部表現 $\tilde{\boldsymbol{\eta}}_i$ は，$Y_i(\boldsymbol{x})$ の内部表現 $\boldsymbol{\eta}_i$ および $\boldsymbol{x}$ の内部表現から得られる $\boldsymbol{\xi}$ を用いて，式(21)から

Table 2

δ_{ij} \ c_j	0	1	u
0 (1)	0	0	0
1 (x_j)	*	0	1

$\tilde{\delta}_{ij}$ = * denotes that $\tilde{Y}_{ij}(\boldsymbol{x})\equiv 0$

導びかれる*.

$$\tilde{\eta}_i=\xi\wedge\eta_i. \tag{21}$$

また，$\tilde{Y}_i(x)\equiv 0$ あるいは $\tilde{Y}_i(x)\equiv 1$ となるか否かは，それぞれ，式(22)によって知られる．(付録(3)参照.)

$$\begin{cases}(\text{i}) & \tilde{Y}_i(x)\equiv 0\Leftrightarrow\neg(\xi\vee\theta)\wedge\eta_i\neq 0,\\ (\text{ii}) & \tilde{Y}_i(x)\equiv 1\Leftrightarrow[\neg(\xi\vee\theta)\wedge\eta_i=0]\\ & \qquad\wedge[\tilde{\eta}_i=0]\end{cases} \tag{22}$$

(4) 積項が単一の変数からなるか否かの判定

与えられた積項 $F_i(x)$ の内部表現から得られる σ_i が次の性質をみたせば，$F_i(x)$ は単一の変数からなる．

$$\sigma_i=2^{j-1}\ \text{for some}\ j,\ j=1,2,\cdots,n. \tag{23}$$

この性質は，**3.** に述べるハッシングの技法を用いる数式処理アルゴリズムできわめて有効に用いられる．

3. 数式処理アルゴリズム

ここでは，**2.2** で述べた形式で内部表現された擬似ブール関数およびブール関数に対して，**2.1** で述べた処理を，ハッシングの技法を用いて効率良く行うアルゴリズムを示す．さらに，このアルゴリズムを従来のアルゴリズムと比較することで，その特徴を明らかにする．

3.1 擬似ブール関数の処理アルゴリズム

擬似ブール関数 $y(x)$ の内部表現 $\{(\eta_1,a_1),(\eta_2,a_2),\cdots,(\eta_m,a_m)\}$ を，η_i をキーとするハッシュ・テーブルとして実現する．ハッシュ関数としては，式(24)で定義される h_p を用いる．同じ h_p の値を持つ異なる η_i に対しては，リニア・リストを構成する．

$$h_p(\eta)=\mathrm{mod}(\eta,p)+1 \tag{24}$$

ここに，p は2を原始根とする素数である．

式(6)で表わされる擬似ブール関数 $\tilde{y}(x)$ の簡単化は以下に示される**アルゴリズム H**（以下 **Al. H** と略す）によって行われる．このアルゴリズムに対する $y(x)$ のデータ構造は，**Fig. 1** のようにして与えられる．

〔アルゴリズム H〕

Step 1:〔初期設定〕
$i\leftarrow 1$; $l\leftarrow 0$; $c'\leftarrow 0$;
HEAD$(j)\leftarrow 0$ for $j=1,2,\cdots,p$; Step 2 へ;

Step 2:〔項が定数か否かの判定〕
(i) $\tilde{Y}_i(x)\equiv 0$ ならば，Step 5へ;
(ii) $\tilde{Y}_i(x)\equiv 1$ ならば，$c'\leftarrow c'+a_i$ として Step 5 へ;

* 付録(1)と同様にして導びかれる.

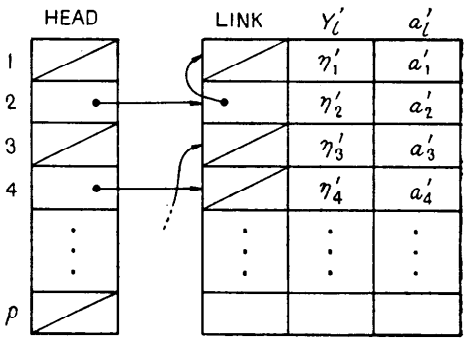

Fig. 1 Date Structure used in the Algorithm H

(iii) それ以外のとき，Step 3 へ;

Step 3:〔同一項の除去〕
Step 3.1: $j\leftarrow h_p(\tilde{\eta}_i)$; $k\leftarrow$HEAD(j);
Step 3.2: (i) $k=0$ ならば，Step 4 へ;
(ii) $k\neq 0$ ならば，Step 3.3 へ;
Step 3.3: (i) $Y'_k(x)\equiv\tilde{Y}_i(x)$ ならば，
$a'_k\leftarrow a'_k+a_i$ として Step 5 へ;
(ii) $Y'_k(x)\neq\tilde{Y}_i(x)$ ならば，
$k\leftarrow$LINK(k) として Step 3.2 へ;

Step 4:〔項の格納〕
$l\leftarrow l+1$; $Y'_l(x)\leftarrow\tilde{Y}_i(x)$; $a'_l\leftarrow a_i$;
LINK$(l)\leftarrow$HEAD(j); HEAD$(j)\leftarrow l$; Step 5 へ;

Step 5:〔アルゴリズムの終了の判定〕
$i\leftarrow i+1$ とする.
(i) $i\leq m$ ならば，Step 2 へ;
(ii) $i>m$ ならば，アルゴリズム終了;

$$y'(x)\triangleq c'+\sum_{i=1}^{l}a'_iY'_i(x) \tag{25}$$

が求める式である．□

このアルゴリズムに対して，$y(x)$ を構成する積項どうしの比較にもとづく従来の処理アルゴリズム（以下では**アルゴリズム A**，または略して **Al. A** と呼ぶ）は **Al. H** の Step 1, Step 3, Step 4 をそれぞれ以下のように変更して得られる．

Step 1:〔初期設定〕
$i\leftarrow 1$; $l\leftarrow 0$; $c'\leftarrow 0$; Step 2 へ; □

Step 3:〔同一項の除去〕
Step 3.1: (i) $l=0$ ならば Step 4 へ;
(ii) $l\neq 0$ ならば，$k\leftarrow 1$ として Step 3.2 へ;
Step 3.2: (i) $Y'_k(x)\equiv\tilde{Y}_i(x)$ ならば，
$a'_k\leftarrow a'_k+a_i$ として Step 5 へ;

（ii） $Y'_k(\boldsymbol{x}) \neq \hat{Y}_i(\boldsymbol{x})$ ならば，Step 3.3 へ；

Step 3.3: $k \leftarrow k+1$ とする．

（i） $k \leq l$ ならば Step 3.2 へ；

（ii） $k > l$ ならば Step 4 へ；□

Step 4:〔項の格納〕

$l \leftarrow l+1$；$Y'_l(\boldsymbol{x}) \leftarrow \hat{Y}_i(\boldsymbol{x})$；$a'_l \leftarrow a_i$；Step 5 へ；□

3.2 ブール関数の処理アルゴリズム

ブール関数 $F(\boldsymbol{x})$ の数式処理は，擬似ブール関数の数式処理と同様に，$F(\boldsymbol{x})$ の内部表現 $\{(\sigma_1, \tau_1), (\sigma_2, \tau_2), \cdots, (\sigma_s, \tau_s)\}$ を，σ_i をキーとするハッシュ・テーブルとして実現することによって効率良く行われる*．ただし，ブール関数の処理では，積項が単一の変数からなるか否かの判定が行われるが，これは，式(23)と整数の剰余類に関する次の性質を利用して容易に実現される．

〔性質〕** p を 2 を原始根とする素数とすると，

$$2^i \not\equiv 2^j \bmod p \text{ for } i \neq j,\ i, j = 1, 2, \cdots, p. \quad (26)$$

この性質と h_p の定義とにより，異なる単一の変数からなる積項どうしは，異なるハッシュ・コードを持つことになる．よって，式(27)で与えられるテーブル BASE(・) を用意すれば，上記の判定は σ_i のハッシングに際して簡単に行われる．

$$\mathrm{BASE}(i) \triangleq \begin{cases} 2^{i-1} & \text{if } i = h_p(2^{i-1}) \\ 0 & \text{otherwise} \end{cases} \quad (27)$$

ここに，$i = 1, 2, \cdots, p$；$j = 1, 2, \cdots, n$；p は n 以上の 2 を原始根とする素数である．

4. アルゴリズムの計算時間の理論的評価

この章では，Al. H および Al. A の計算時間の理論的評価式を示し，更に，ある仮定のもとで，それぞれのアルゴリズムの計算時間の上限を求める．

4.1 計算時間の理論的評価式

式(3)の形式に従う目的関数*** $y(\boldsymbol{x})$ に含まれる変数の一部に 0 または 1 の値を与え，さらに式の簡単化を行って，

$$y'(\boldsymbol{x}) \triangleq \sum_{i=1}^{l} a'_i Y'_i(\boldsymbol{x}) \quad (28)$$

が得られたとする．ただし，以下の解析では，初めに与えられる $\{Y_i(\boldsymbol{x}) \mid i = 1, 2, \cdots, m\}$ の中には，$\hat{Y}_i(\boldsymbol{x}) \equiv 0$ および $\hat{Y}_i(\boldsymbol{x}) \equiv 1$ となる積項は含まれないとする．このとき，両アルゴリズムの計算時間の一般的評価式は，次のように与えられる．

$$T = t_1 + m \cdot t_2 + T_3 + l \cdot t_4 + m \cdot t_5. \quad (29)$$

ここに，t_1, t_2, t_4, t_5 は，それぞれアルゴリズムの Step 1, Step 2, Step 4, Step 5 を一回実行するのに要する時間（l, m に依存しない定数）とする．T_3 は，アルゴリズムが終了するまでに要する Step 3 の計算時間の総和を表わすが，その値は，扱われる目的関数 $y(\boldsymbol{x})$ に強く依存するので，一般的評価は困難である．t_1 は，Al. A については定数，Al. H については，ハッシュ・テーブルの大きさ p の一次式として，

$$t_1 = a \cdot p + b \quad (30)$$

で表わされる．(a, b は定数)

4.2 計算時間の上限

両アルゴリズムの計算時間が最大となるのは，$\hat{y}(\boldsymbol{x})$ が次の条件Cを満たす場合である．

〔条件C〕 $\hat{y}(\boldsymbol{x}) = \sum_{i=1}^{m} a_i \hat{Y}_i(\boldsymbol{x})$ として，

$$\hat{Y}_i(\boldsymbol{x}) \not\equiv 0 \wedge \hat{Y}_i(\boldsymbol{x}) \not\equiv 1 \text{ for all } i,\ 1 \leq i \leq m,$$

かつ，

$$\hat{Y}_i(\boldsymbol{x}) \not\equiv Y_j(\boldsymbol{x}) \text{ for all } i, j,\ 1 \leq i \leq j \leq m \quad \square$$

条件Cがなり立つ場合の両アルゴリズムの計算時間の上限と，その m（項数）に対する依存性が，以下のように求められる．

（1） アルゴリズムHのばあい

HEAD (j), $(j = 1, 2, \cdots, p)$ を始点とするリニア・リストの長さの期待値が，数式処理の各時点で j について一様であるとすると，条件Cのもとでの Step 3 の i 回目の計算時間の期待値は，積項間の比較回数の期待値 $(i-1)/p$ を用いて，

$$c \cdot (i-1)/p + d \quad (c, d \text{ は定数}) \quad (31)$$

で与えられる．よって，Step 3 の計算時間の総和の期待値は，

$$T_3 = \sum_{i=1}^{m} [c \cdot (i-1)/p + d]$$

$$= c \cdot (m-1) \cdot (m-2)/(2p) + d \cdot m \quad (32)$$

式(29), (30), (32)から，計算時間の上限の期待値 T_H が次式のように与えられる．

$$T_H = a \cdot p + b + m \cdot t_2 + c \cdot (m-1) \cdot (m-2)/(2p) + d \cdot m + m \cdot t_4 + m \cdot t_5$$
$$= c \cdot (m-1) \cdot (m-2)/(2p) + (t_2 + t_4 + t_5 + d) \cdot m + a \cdot p + b. \quad (33)$$

T_H を最小にする p の値 p^* は，

---

* 式(8)によると，σ_i には x_j と $\bar{x}_j$ とを区別する情報は含まれていないが，$F(\boldsymbol{x})$ が積和形であることにより，相補律が陰に適用されるという理由で，σ_i がキーになりうる．

** たとえば文献 5) 参照．

*** 数式処理の内容については，$y(\boldsymbol{x})$, $F(\boldsymbol{x})$ とも本質的には同じであるから，以下の考察は主として $y(\boldsymbol{x})$ について行う．

$$\partial T_H/\partial p = -c\cdot(m-1)\cdot(m-2)/(2p^2)+a=0$$

から，

$$p^*=\sqrt{c\cdot(m-1)\cdot(m-2)/(2a)} \qquad (34)$$

として求められる．

実際には，pは2を原始根とする素数に限られるので，$m\gg1$の場合には，$\lambda=\sqrt{c/(2a)}$として，

$$p^*\approx\lambda\cdot m \qquad (35)$$

となるp^*をもとめ*，$p=p^*$とすれば，T_Hを最小にすることができる．このばあい，T_Hの最小値$T_H^{\min}$は，

$$T_H^{\min}\approx(\sqrt{2ac}+t_2+t_4+t_5+d)\cdot m+b. \qquad (36)$$

すなわち，Al. Hの計算時間の上限は$O(m)$となることが知られる．

このことにより，与えられた$y(x)$の項数mに応じてpを動的にp^*設定するAl. Hを，**特にアルゴリズムH*（Al. H*と略す）**と呼ぶ．

（2） アルゴリズムAのばあい

Step 3のi回目の実行時間は，条件Cのもとでは，積項間の比較回数が$i-1$となることから，

$$e\cdot i+g \quad (e,g\text{は定数}), \qquad (37)$$

で与えられる．よって，Step 3の計算時間の総和は，

$$T_3=\sum_{i=1}^{m}(e\cdot i+g)$$
$$=e\cdot m\cdot(m-1)/2+g\cdot m \qquad (38)$$

計算時間の上限T_Aは，次式で与えられる．

$$T_A=t_1+m\cdot t_2+e\cdot m\cdot(m-1)/2+g\cdot m+m\cdot t_4+m\cdot t_5$$
$$=e\cdot m\cdot(m-1)/2+(t_2+t_4+t_5+g)\cdot m+t_1. \qquad (39)$$

よって，$m\gg1$の時には，Al. Aの計算時間の上限は$O(m^2)$であることがわかる．

5. 実験及びその結果の評価

5.1 実験の目的

実験は，実験1と実験2とからなり，実験1では，Al. H*とAl. Aとに対して，**4.**で評価された計算時間の上限の妥当性の検証と，両アルゴリズムの計算時間の比較を行う．実験2では，これらのアルゴリズムを用いて擬似ブール計画問題の求解ソフトウェア・システムを2つ実現し，両システムによる計算時間の比較を行う．

5.2 実 験 1

一様乱数を用いて，条件Cをみたす30変数の擬似ブール関数$y(x)$を発生させ，これに対して，**2.1**で述べた処理をAl. H，Al. Aによってほどこし，それに要する計算時間を測定した．ただし，$y(x)$の項数mは，m=100, 200, 300, 400, 500とし，Al. Hについては，mの各値について20通りの$y(x)$について実験を行い**，その平均値を求めた．

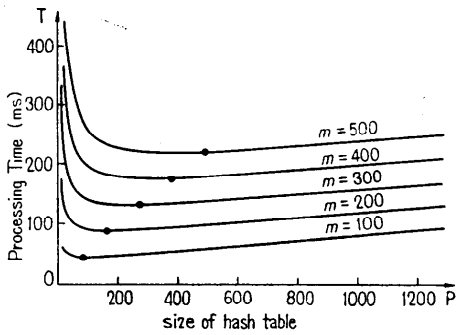

Fig. 2 Procesing Time for $y(x)$ by the Al. H
• in this Fig. denotes the minimum of T.

Fig. 2に，pを変化させた場合のAl. Hの平均計算時間を示す．同図より，Al. Hの計算時間の上限の評価式が妥当であることがわかる．すなわち，式(33)で，

$$\begin{cases} k_0=c\cdot(m-1)\cdot(m-2)/2, \\ k_1=a, \\ k_2=(t_2+t_4+t_5+d)\cdot m+b, \end{cases} \qquad (40)$$

とおくと，

$$T_H=k_0/p+k_1\cdot p+k_2 \qquad (41)$$

の関数形を得る．これは，**Fig. 2**の各曲線を適切に表現していると考えられ，従って，式(33)にもとづいて行ったAl. H*の存在の予測は正しかったことが結論される．

そこで，**Fig. 2**において最小の平均計算時間を与えるpの値にもっとも近い2を原始根とする素数を，各mについて求め，これをAl. H*におけるp^*とした．Al. AとAl. H*の平均計算時間が**Table 3**（次頁参照）に，それをグラフ化したものが**Fig. 3**（次頁参照）に示されている．同図よりAl. AとAl. H*の平均計算時間はそれぞれ$O(m^2)$, $O(m)$であることがわかり，このことから計算時間の上限の評価式が妥当であることが知られる．また，Al. H*を用いることによって，とくにmの値が大きい場合，数式処理（**2.1**による）の時間が大幅に改善される（m=500で約4%に）ことがわかる．

---

* $\lambda\cdot m$が2を原始根とする素数でない時には，$\lambda\cdot m$にもっとも近い2を原始根とする素数をp^*とする．

** 条件Cが満たされていれば，Al. Aの計算時間は問題によらず一定である．

Table 3 Processing Time for $y(x)$ by the **Al. H*** and the **Al. A**

m	p^*	Processing Time (ms.)		Ratio (%)
		Al. H*	Al. A	
100	83	44	236	18.6
200	163	88	898	9.8
300	269	132	1,984	6.7
400	373	175	3,501	5.0
500	491	219	5,442	4.0

m: number of terms involved in $y(x)$;
p^*: optimum size of Hash Table;

5.3 実 験 2

擬似ブール計画法を実行するために Al. H* と Al. Aとを用いて実現されたソフトウェア・システムをそれぞれ Sys. H, Sys. A と呼ぶことにする．一様乱数を用いて発生された30変数の擬似ブール計画問題を両システムによって解いた場合の，全ての最適解を得るまでに要した計算時間，及びそのうちの数式処理にかかわる部分の計算時間を **Table 4** に示す．**Table 4** において s, m は，それぞれ制約条件式の項数，目的関数の項数を表わし，表中の数値は，s, m の値の各組み合わせについて，それぞれ20問題を解いたときの平均計算時間である*．

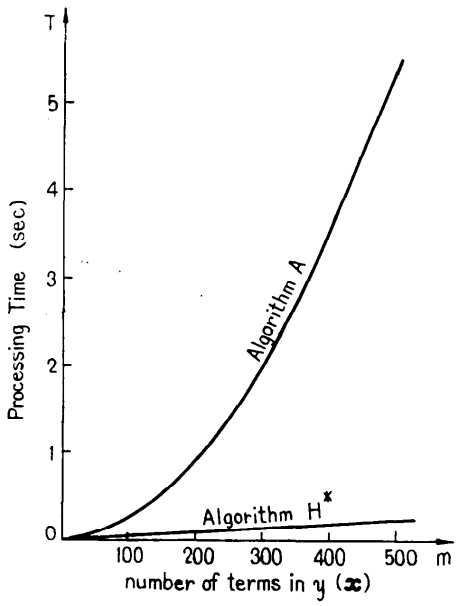

Fig. 3 Comparison of Processing Time for $y(x)$ by the Algorithm H* and Algorithm A

Table 4 Inprovement Rate of System H (Average for 20 problems) unit: sec

s	m / システム / 時間	20			60			100		
		A	H	改善度	A	H	改善度	A	H	改善度
0	全体	3.41	3.40	0.3	95.5	88.8	7.2	414.0	360.0	12.6
	数式処理	0.386	0.379	4.2	12.7	6.05	53.2	90.9	37.1	61.1
	比率† (%)	11.2	10.9		13.4	6.76		20.7	9.41	
60	全体	4.42	4.09	7.7	44.2	40.2	9.0	110.0	99.0	10.3
	数式処理	2.03	1.70	18.7	13.6	9.69	29.0	32.1	21.2	34.3
	比率† (%)	45.7	40.4		31.1	24.3		30.1	22.1	
120	全体	7.23	4.41	39.3	16.2	12.1	27.3	20.4	15.4	26.1
	数式処理	5.33	2.51	53.3	9.07	4.80	48.4	9.98	5.00	53.0
	比率† (%)	73.9	56.7		56.4	39.9		49.4	31.2	

s: $F(x)$ の項数; m: $y(x)$ の項数; 比率†: 数式処理の時間の全体の時間に対する比率;

以下，システムの改善度として，次式で定義される I を用いる．

$$I \triangleq (T_A - T_H)/T_A \times 100(\%). \qquad (42)$$

ここに，T_H, T_A はそれぞれ，Sys. H, Sys. A の要した計算時間を表わす．

Fig. 4 (次頁参照) にシステムの改善度を，**Fig. 5** (次頁参照) に数式処理に要した時間のシステム全体の計算時間に対する割合を示す．**Fig. 4** より，とくに数式処理に関しては，目的関数の項数 m の増大とともに改善度が増す傾向が認められ，**Fig. 2** にもとづく予測結果と一致する．

擬似ブール計画法のアルゴリズムは Branch and Bound 法にもとづいており，アルゴリズムの進行途次においては，原問題に含まれる変数の一部に値を与えることによって，いくつかの部分問題が生成される．これらの部分問題の大きさは，あきらかに原問題よりは小さい．従って，m の値の減少によって Al. H* の効果が減少する **Fig. 2** の事実を考慮すれば，**Fig. 4** に示された処理時間の改善度は良好であるといえよう．

なお，現在のシステムでは，部分問題を生成すると

* **Table 2** における m の範囲が100～500にとられているのに対し，**Table 4** ではそれが20～100になっているのは，使用した計算機システムの主記憶容量から来る制限による．

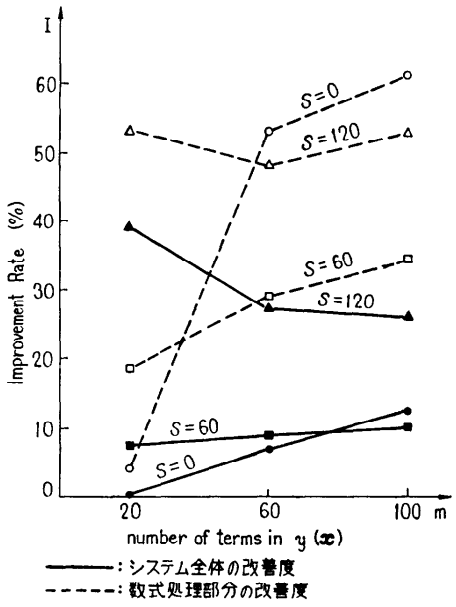

Fig. 4 Inprovemet Rate of System

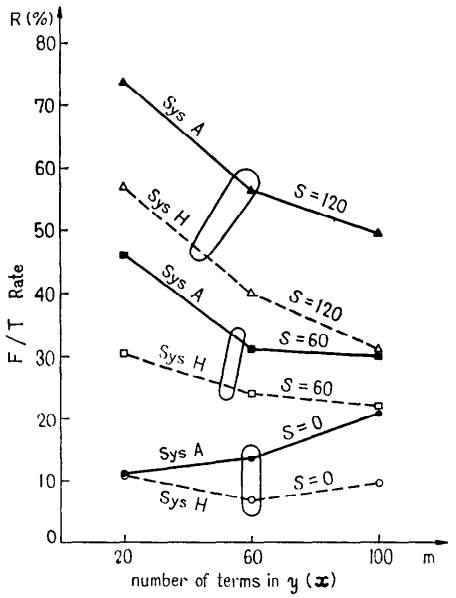

Fig. 5 *F/T* Rate

きには，かならず原問題に立ちかえって変数に値を与えるようにしているが，一つ前の部分問題から新しい部分問題をつくるようにシステムを改良すれば，Al. H* の効果はさらに有効になると考えられる．

システム全体の改善度は，制約条件式の項数 s が増大するにつれて増す傾向にあることが認められる．これは，数式処理部分に要する計算時間のシステム全体の計算時間に対する割合（***F/T* 比**と呼ぶ）が s の増加にともなって増すためであることが **Fig. 5** から知られる．また，$s=120$ のばあいに，システム全体の改良度が m の増大にともなって減少しているのは，m の増大にともなう F/T 比の減少がこの場合とくに著しいためであることが同じく **Fig. 5** より知られる．

6. あ と が き

ハッシングの技法を用いて数式処理アルゴリズムを構成することにより，従来の方法では時間的な制約により取り扱うことができなかった大規模な問題を効率良く解くことが可能になると期待される．

また，アルゴリズム H* は，変数のべき乗を含む多項式の数式処理の範囲まで容易に拡張できるので，この種の数式処理に関して，かなりの有効さを持つことが予想される．

本研究には，名古屋大学工学部情報工学科の FACOM 230/38 FORTRAN-S が用いられた．

謝辞：末筆ながら，日頃御指導賜る本学の本多波雄教授ならびに，熱心に討論していただき，貴重な御意見をいただく福村研究室の諸氏に深謝します．

参 考 文 献

1) D. E. Knuth: The Art of Computer Programming, Vol. 3, Sorting and Searching, Chap. 6. 4, Addison-Wesley, (1973).
2) 吉田，稲垣，福村: Branch and Bound 法にもとづく擬似ブール計画法のアルゴリズム，電子通信学会雑誌，第 50 巻 10 号，pp. 231～238，(昭和 42 年 10 月)．
3) 今井，吉田，福村: 目的関数と制約条件式の特徴を利用した擬似ブール計画法のアルゴリズムとその評価，昭和 50 年度電子通信学会 全国大会講演論文集 (分冊 6) 講演番号 1123，(昭和 50 年 3 月)．
4) 今井，吉田，福村: Hashing の技法を用いた数式処理アルゴリズム，昭和 50 年度情報処理学会第 16 回大会講演論文集，講演番号 247，(昭和 50 年 11 月)．
5) 高木: 初等整数論講義，共立，(昭和 6 年)．

付 録

(1) 式(18)の導出について

Table 1 に定義される演算は，ε_{ij} と x_j に与える値 C_j との組に対する ε_{ij} の値を与える．x の内部表現

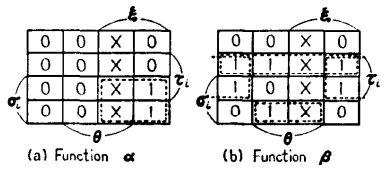

Fig. A1 Veitch-Karnnugh map for Function α and Function β

(ξ, θ) および F_i の内部表現 (σ_i, τ_i) より $\tilde{F}_i$ の内部表現 $(\tilde{\sigma}_i, \tilde{\tau}_i)$ を得る関数を α および β とする．すなわち，

$$\begin{cases}\tilde{\sigma}_i=\alpha(\xi, \theta, \sigma_i, \tau_i), \\ \tilde{\tau}_i=\beta(\xi, \theta, \sigma_i, \tau).\end{cases} \tag{A1}$$

$\tilde{\sigma}_i, \tilde{\tau}_i, \xi, \theta, \sigma_i, \tau_i$ の同一けたを考えると，α および β はともに4変数のブール関数と考えることができる．関数 α, β を決定するため Veitch-Karnaugh 図表を **Table 1** をもとにして作り，**Fig. A1** に示す．

Fig. A1 より関数 α, β はそれぞれ次式で与えられる．

$$\begin{cases}\alpha(\xi, \theta, \sigma_i, \tau_i)=\xi \wedge \sigma_i \\ \beta(\xi, \theta, \sigma_i, \tau_i)\end{cases} \tag{A2}$$

$$\begin{aligned}&=(\neg\theta)\wedge\tau_i \vee (\neg\sigma_i)\wedge\tau_i \vee \theta\wedge\sigma_i\wedge(\neg\tau_i) \\ &=\neg(\theta\wedge\sigma_i)\wedge\tau_i \vee \theta\wedge\sigma_i\wedge(\neg\tau_i) \\ &=(\theta\wedge\sigma_i)\oplus\tau_i.\end{aligned}$$

（2） 式(19)の導出

$F_i(\boldsymbol{x})=\prod_{j=1}^{n} x_j^{\varepsilon_{ij}}$ とすると，次の命題が成り立つ*.

$$\begin{aligned}[F_i(\boldsymbol{x})\equiv 0] &\Leftrightarrow (\exists j)[1\le j\le n \wedge x_j^{\varepsilon_{ij}}=0] \\ &\Leftrightarrow (\exists j)[1\le j\le n \wedge \varepsilon_{ij}=f] \\ &\Leftrightarrow (\exists j)[1\le j\le n \wedge (\sigma_i)j=0 \wedge (\tau_i)j=1] \\ &\Leftrightarrow [(\neg\sigma_i)\wedge\tau_i \neq \boldsymbol{0}].\end{aligned} \tag{A3}$$

$$\begin{aligned}[F_i(\boldsymbol{x})\equiv 1] &\Leftrightarrow (\forall j)[1\le j\le n \Rightarrow x_j^{\varepsilon_{ij}}=1]^{**} \\ &\Leftrightarrow (\forall j)[1\le j\le n \Rightarrow \varepsilon_{ij}=t] \\ &\Leftrightarrow [\sigma_i=\boldsymbol{0}\wedge\tau_i=\boldsymbol{0}].\end{aligned} \tag{A4}$$

（3） 式(22)の導出

$Y_i(\boldsymbol{x})=\prod_{j=1}^{n} x_j^{\delta_{ij}}$, $\bar{Y}_i(\boldsymbol{x})=\prod_{j=1}^{n} x_j^{\bar{\delta}_{ij}}$ とすると，次の命題が成り立つ．

$$\begin{aligned}&[\bar{Y}_i(\boldsymbol{x})\equiv 0]\Leftrightarrow(\exists j)[1\le j\le n\wedge(\delta_{ij}=1)\wedge(c_j=0)] \\ &\Leftrightarrow(\exists j)[1\le j\le n\wedge((\eta_i)_j=1)\wedge((\xi)_j=0)\wedge((\theta)_j=0)] \\ &\Leftrightarrow[(\neg\xi)\wedge(\neg\theta)\wedge\eta_i\neq\boldsymbol{0}] \\ &\Leftrightarrow[\neg(\xi\vee\theta)\wedge\eta_i\neq\boldsymbol{0}].\end{aligned} \tag{A5}$$

（昭和51年6月30日受付）

---

* $(\sigma_i)_j$ は σ_i の下位より j 番目のビットを表わす．$(\tau_i)_j$ 等についても同様である．

** ⇒は含意を表わす．