社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

不完全定義インデックス生成関数の変数最小化について

中村　高明† 　笹尾　勤† 　松浦　宗寛††

† 九州工業大学大学院 情報工学府 情報創成工学専攻 〒820–8502 福岡県飯塚市大字川津 680–4
†† 九州工業大学 情報工学部 〒820–8502 福岡県飯塚市大字川津 680–4

あらまし　不完全定義インデックス生成関数の入力変数を削減する方法を示す. もとの入力変数 (原始変数) を EXOR して得られる変数を多重化変数という. 不完全定義インデックス生成関数の場合, もとの変数と多重化変数を併用すると, 関数表現のために必要な変数を大幅に削減できる. 本論文では, 変数の選択のヒューリスティックな方法として, 情報利得法を提案する. 原始変数のみを用いる場合と, 多重化変数を用いる場合の実験結果を示し, 多重化変数を用いることにより, より多くの入力変数を削減可能なことを実験的に示す.

キーワード　不完全定義関数, インデックス生成関数, 最小被覆, 論理関数, パターンマッチング

On the Minimization of Input Variables for Incompletely Specified Index Generation Functions

Takaaki NAKAMURA†, Tsutomu SASAO†, and Munehiro MATSUURA††

† Program of Creation Informatics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan
†† Department of Computer Science and Electronics, Kyushu Institute of Technology
680–4, Kawazu, Iizuka, Fukuoka, 820–8502 Japan

Abstract　This paper shows a method to reduce the numbers of input variables to represent incompletely specified index generation functions. A compound variable is generated by EXORing a few original input variables. Incompletely specified index generation functions can be represented by a combination of original and compound variables. As a means to select variables, we propose a heuristic method using information gains. We represented randomly generated incompletely specified index generation functions using only original variables, and using both original and compound variables. Experimental results show that the use of compound variables is effective for reducing the number of input variables.

Key words　Incompletely specified function, Index generation function, Minimum covering, Logic function, Pattern matching

1. はじめに

二値のベクトルが入力されたとき, そのベクトルに対応する固有のインデックスを出力する回路をインデックス生成回路という. インデックス生成回路は, IP アドレスリスト等に利用されている [7] ほか, メモリの欠陥マップへの応用も考えられる [5]. インデックス生成回路の構成方法としては, CAM (Content Addressable Memory) や PLA (Programmable Logic Array) を用いる方法が一般的であるが, 消費電力が大きいという問題がある [6]. そのため, メモリを用いる構成方法が考えられるが, 単一メモリでは効率が悪いため, ハッシュメモリと補助メモリを用いる方法が提案されている [5]. メモリを用いる場合, CAM と比べて多くのビット数を必要とする. ハッシュメモリと補助メモリに分けて構成する場合, ハッシュメモリの入力数削減により, 総メモリ量の削減が可能となる.

本論文では, 不完全定義インデックス生成関数の入力変数削減手法について述べる. まず, 第 2 章では, 不完全定義インデックス生成関数の定義と性質について述べる. 第 3 章では, 変数の多重化による不完全定義インデックス生成関数の入力変数削減について述べる. 第 4 章では, 不完全定義インデックス生成関数の多重化変数を選択するためのヒューリスティック法である情報利得法について述べる. 第 5 章では, 不完全定義インデッ

クス生成関数の変数削減効果を, 多重化変数の多重度や, 登録ベクトル分布を変化させた場合の結果について述べる. 最後に, 第 6 章で本論文のまとめと今後の課題について述べる.

2. 不完全定義インデックス生成関数

［定義 2.1］ k 個の異なる登録ベクトル $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_k$ に対して, 対応するインデックス $1, 2, \ldots, k$ の値を表にしたものを**インデックス表**という.

［定義 2.2］ $B = \{0, 1\}$ とし, B^n 中の k 個の異なるベクトルを $\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_k$ とする.

$D = \{\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_k\}$, $i \in \{1, 2, \ldots, k\}$ とするとき,

$\vec{a}_i \in D$ のとき $f(\vec{a}_i) = i$,

$\vec{b} \in B^n - D$ のとき $f(\vec{b}) = 0$,

となる関数 $f : B^n \to \{0, 1, 2, \ldots, k\}$ を**重み k のインデックス生成関数**という. また,

$\vec{a}_i \in D$ のとき $\hat{f}(\vec{a}_i) = i$,

$\vec{b} \in B^n - D$ のとき $\hat{f}(\vec{b}) = d$（ドント・ケア）,

となる関数 $\hat{f} : B^n \to \{1, 2, \ldots, k, d\}$ を**重み k の不完全定義インデックス生成関数**という.

不完全定義インデックス生成関数では, 関数を表現するための変数を削減可能である [1].

［例 2.1］ 図 1(a) にインデックス表を示す. また, 図 1(b) に, 対応する不完全定義インデックス生成関数の分解表を示す. ここでの空白セルはドント・ケアを表す.

この関数では, $\vec{a}_1 = (0,0,0,1)$, $\vec{a}_2 = (1,0,1,1)$, $\vec{a}_3 = (1,1,0,0)$, $\vec{a}_4 = (0,1,1,1)$ の各入力に対し, $\hat{f}_1(\vec{a}_1) = 1$, $\hat{f}_1(\vec{a}_2) = 2$, $\hat{f}_1(\vec{a}_3) = 3$, $\hat{f}_1(\vec{a}_4) = 4$ である. その他の入力に対しては, $\hat{f}_1$ の値は d（ドント・ケア）である. 分解表の各列に非零要素が 1 個ずつある場合, 各列のドント・ケアの値を, その列にある非零要素の値に等しくすることにより, この関数を x_1 と x_2 の値のみに依存するように変換できる. 図 1(a) のインデックス表では, (x_1, x_2) の値がすべて異なっており, この 2 つの変数のみでインデックスを判別できる. このように, 分解表において, 各列に非零要素が高々1 個しか存在しない場合, 関数を表現するための変数の個数を削減できる. （例終り）

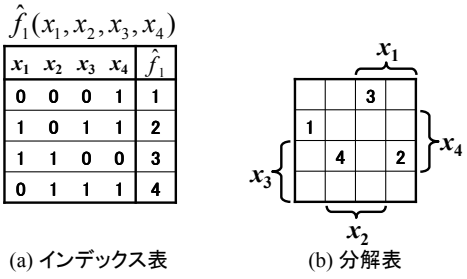

図 1　4 入力不完全定義インデックス生成関数 $\hat{f}_1$

［例 2.2］ 図 2 に不完全定義インデックス生成関数 $\hat{f}_2$ のインデックス表と分解表を示す. この関数を表現するために必要な変数の個数を考える. 図 2(a) の分解表では, $(x_1, x_2) = (1, 1)$ の列に 2 個の非零要素をもつ. 従って, 関数 $\hat{f}_2$ を $\{x_1, x_2\}$ のみで表現することは不可能である. 同様に, $(x_3, x_4) = (1, 1)$ の行にも 2 個の非零要素をもつため, 関数 $\hat{f}_2$ を $\{x_3, x_4\}$ のみで表現することは不可能である.

次に, 図 2(b) に示すように, 分解表の変数の分割を (x_1, x_4), (x_2, x_3) とすると, 各列に非零要素を高々1 個しかもたない. 図 2(b) のインデックス表では, (x_1, x_4) の値がすべて異なっており, この 2 つの変数のみでインデックスを判別できる. 従って, 関数 $\hat{f}_2$ は, $\{x_1, x_4\}$ のみで表現できる. （例終り）

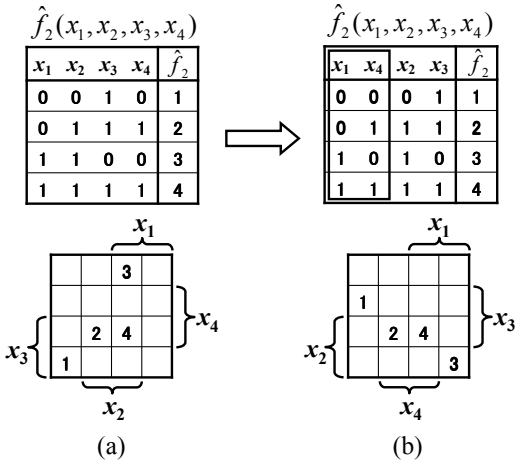

図 2　4 入力不完全定義インデックス生成関数 $\hat{f}_2$

以上のように, 不完全定義インデックス生成関数では, 変数の選択を工夫することにより, 関数を表現するための入力変数を削減できる. 不完全定義インデックス生成関数をメモリで実現する場合, 入力変数を 1 個削減すると, 必要なメモリ量は半分となる. つまり, 入力変数の削減により, メモリ量を大幅に削減できる.

単一出力不完全定義関数の入力変数の最小化法に関しては, 参考文献 [1], [2] で考案されている.

［補題 2.1］ 重み k の不完全定義インデックス生成関数を表現するための変数の個数を p とするとき, 関係

$$\lceil \log_2(k+1) \rceil \leqq p$$

を満たす.

例えば, $k + 1 = 2^p$ の場合, 高さ p の完全二分木を考えると, p 個の変数で k 個の登録ベクトルを区別できる. 不完全定義インデックス生成関数の変数削減問題は, 登録ベクトルを区別するための二分木の高さを削減する問題と考えることもできる. 高さを削減するためには, なるべく部分木の高さが等しくなるように変数を選択すればよい. つまり, なるべくバランスした部分木を構成するように変数を選択すればよい. この考え方は, 後述する情報利得法の変数選択のアルゴリズムで用いる.

重み k の不完全定義インデックス生成関数は, 多くの場合, 高々 $2\lceil \log_2(k+1) \rceil - 1$ 個の変数で表現可能である [5]. すなわち, $\lceil \log_2(k+1) \rceil \leqq p \leqq 2\lceil \log_2(k+1) \rceil - 1$ が成立する.

3. 多重化変数による入力変数削減

ここでは, 入力変数の線形関数を用いることにより, 不完全定義関数を表現するための入力変数をさらに削減する手法を示す.

［定義 3.3］ $B=\{0,1\}$, $i\in\{1,2,\ldots,n\}$ とする. 不完全定義インデックス生成関数の n 個の入力変数 $x_1,x_2,\ldots,x_n$ に対して, 多重化変数 y を次のように定義する.

$$y=c_1x_1\oplus c_2x_2\oplus\cdots\oplus c_nx_n \qquad (c_i\in B)$$

多重化変数 y の定数 c_i において, $c_i=1$ となる定数の個数を y の**多重度**という.

多重度 1 の変数は, もとの入力変数（**原始変数**）を表す. また, 多重度 2 の多重化変数を **2 重化変数**, 多重度 3 の多重化変数を **3 重化変数**という.

［例 3.3］ 図 3 に不完全定義インデックス生成関数 $\hat{f}_3$ を示す. この関数を表現するために必要な変数の個数を考える. 図 3(a) の分解表では, $(x_1,x_2)=(1,1)$ の列に 2 個の非零要素をもつため, 関数 $\hat{f}_3$ を $\{x_1,x_2\}$ のみで表現することは不可能である. 同様に, $(x_3,x_4)=(1,1)$ の行に 2 個の非零要素をもつため, $\{x_3,x_4\}$ のみで表現することも不可能である. また, 関数 $\hat{f}_3$ は対称関数であるため, 他の組合せの分解表でも同様の結果となる. 結局, $\hat{f}_3$ を表現するためには 3 個の変数が必要であることがわかる.

次に, 2 重化変数 $y_1=x_1\oplus x_2$, $y_2=x_2\oplus x_3$ を定義すると, 図 3(b) に示す関数 $\hat{g}_3(y_1,y_2,x_3,x_4)$ が得られる. このとき, 図 3(b) の分解表では, 各列に非零要素は高々1 個しか存在しない. 従って, 関数 $\hat{g}_3$ は, $\{y_1,y_2\}$ のみで表現できる. （例終り）

このように, 不完全定義インデックス生成関数の入力変数の多重化を行うことにより, 入力変数を削減できる. 以後, 複数の入力変数を多重化して得られる多重化変数 y についても, 1 個の入力変数として扱う.

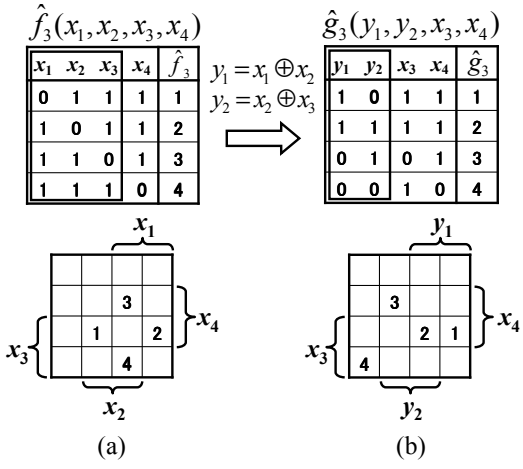

図 3　4 入力不完全定義インデックス生成関数 $\hat{f}_3$ と変数の多重化を行った関数 $\hat{g}_3$

変数の多重化を論理回路で実現する場合, 図 4 のような変数多重化回路を用いる. 図 4 のマルチプレクサの制御入力の値はレジスタに蓄える. これより, n 個の入力変数から 1 個の変数を選択可能である. 図 4(a) では, 2 重化変数を生成しているが, 変数 y に対する 2 個のマルチプレクサのうち, 下部のマルチプレクサの入力に定数 0 を入れて $y=x_i\oplus 0$ とすることにより, $y=x_i$ の実現も可能である. 図 4(a) では, 2 重化回路を示したが, マルチプレクサの個数を増やすことにより, 多重度を増やすことも可能である. 変数の多重化を全く行わない場合には図 4(b) の回路を用いる.

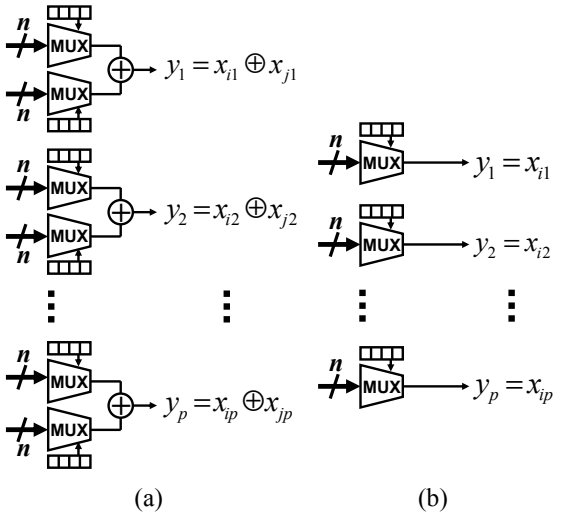

図 4　変数多重化回路

4. 多重化変数選択のヒューリスティック法

原始変数のみ用いる場合には, 一種の最小被覆問題を解くことにより, 不完全定義インデックス生成関数を表現するための変数を最小化できる [4] [5].

［アルゴリズム 4.1］ **不完全定義インデックス生成関数の変数最小化**

（1） 各登録ベクトルを区別するための条件を論理式で表現する.

（2） 論理式の値を 1 とするような, 重み最小の割り当てを求める.

これは, 論理式を BDD (Binary Decision Diagram) で表現し, その重み最小パスを求めることにより実行できる.

［例 4.4］ 図 2(a) に示す不完全定義インデックス生成関数 $\hat{f}_2$ の変数最小化を行う.

（1） 4 つの登録ベクトルをそれぞれ $\vec{a}_1=(0,0,1,0)$, $\vec{a}_2=(0,1,1,1)$, $\vec{a}_3=(1,1,0,0)$, $\vec{a}_4=(1,1,1,1)$ とおく.

（2） $\vec{a}_1$ と $\vec{a}_2$ を区別するためには, x_2 または x_4 が必要である. 従って, 条件式 $x_2\vee x_4$ が得られる. 同様に, $\vec{a}_1$ と $\vec{a}_3$ を区別するための条件式は $x_1\vee x_2\vee x_3$, $\vec{a}_1$ と $\vec{a}_4$ を区別するための条件式は $x_1\vee x_2\vee x_4$, $\vec{a}_2$ と $\vec{a}_3$ を区別するための条件式は $x_1\vee x_3\vee x_4$, $\vec{a}_2$ と $\vec{a}_4$ を区別するための条件式は x_1, $\vec{a}_3$ と $\vec{a}_4$ を区別するための条件式は $x_3\vee x_4$ となる.

（3） すべての登録ベクトルを区別する条件は, すべての条件式が同時に成立することである. 従って, 条件式 $R = x_1(x_2 \vee x_4)(x_3 \vee x_4)(x_1 \vee x_2 \vee x_3)(x_1 \vee x_2 \vee x_4)(x_1 \vee x_3 \vee x_4)$ を得る.

（4） 分配律および吸収律により, $R = x_1x_2x_3 \vee x_1x_4$ となる. 積項の最小リテラル数は 2 であり, 積項 x_1x_4 が最小解を表す. 従って, 関数 $\hat{f}_2$ は, 2 個の変数 x_1, x_4 で表現可能である.

（例終り）

変数の多重化を許す場合の入力変数最小化も, 理論的には通常の場合と同様に実行できる. つまり, 原始変数 $x_1, x_2, \ldots, x_n$ の他に, 多重化変数 $y_1, y_2, \ldots, y_m$ を独立な変数と考え, 変数の最小化を行えばよい. 2 重化変数までを許す場合には, 等価入力数は, $n + \binom{n}{2} = \binom{n+1}{2}$ となる. 3 重化変数までを許す場合には, 等価入力数は, $n + \binom{n}{2} + \binom{n}{3} = \frac{n(n^2+5)}{6}$ となる.

最小化に BDD を用いる場合, 入力変数の増加に伴い BDD の大きさは指数関数的に増加し, ある値を超えるとメモリオーバフロを起こす. 従って, 最小化が可能な関数の入力変数の個数には制限がある.

[定義 4.4] 不完全定義インデックス生成関数において, $|h_{i0} - h_{i1}|$ を変数 x_i における**分割差**という. ここで, h_{i0} は $x_i = 0$ となる登録ベクトルの個数, h_{i1} は $x_i = 1$ となる登録ベクトルの個数である.

分割差の小さい変数ほど, ベクトルの集合を均等に分割できる.

ベクトルの個数を k とする. 与えられた変数集合が, ベクトルの集合を均等に分割する場合, 関数を表現するための変数の個数は $\lceil \log_2(k+1) \rceil$ 個まで削減できる. このことから, 次のアルゴリズムを得る.

[アルゴリズム 4.2] **2 重化変数を用いた変数削減**

（1） 入力変数を $x_1, x_2, \ldots, x_n$ とする.

（2） 2 重化変数 $y_i = x_{j1} \oplus x_{j2}$ を m 個選ぶ. この際, 分割差がなるべく小さな変数を選ぶ.

（3） 入力数が $n + m$, ベクトル数が k のインデックス生成関数と考え, アルゴリズム 4.1 を用いて入力変数を削減する.

m の値を多くすると, 最適解に近い解を得ることができるが, 計算に多くの時間を要する.

次に, 多重化変数の選択を高速に行うヒューリスティック法である情報利得法について示す.

多重化変数の選択問題は, 決定木の最適化問題 [3] として考えることができる. 多重化変数を選ぶ際, 分割差が小さいほど, その変数を選ぶことによって得られる情報利得が大きい. 従って, 情報利得の大きな変数（つまり, 分割差の小さな変数）を選ぶことにより, 決定木の変数を削減可能である.

[アルゴリズム 4.3] **情報利得法**

（1） 原始変数を含む多重化変数の中から, 分割差が最小となる変数 y_1 を選び, 登録ベクトルを $y_1 = 0$ と $y_1 = 1$ の 2 つの集合に分割する.

（2） ベクトル数が 2 以上の集合に対して, 分割差の最大値が最小となる変数を選び, 各集合を分割する.

（3） すべての集合のベクトル数が 1 個になるまで手順 (2) を繰り返す.

（4） 得られた変数を $y_1, y_2, \ldots, y_p$ とする.

[例 4.5] 図 5 に不完全定義インデックス生成関数 $\hat{f}_4$ を示す. この関数の多重化変数を情報利得法によって選択する. 本来, 集合の要素は登録ベクトルであるが, 表記の簡略化のため, 以下では集合の要素をインデックスで表す.

まず, 集合 $S_{11} = \{1,2,3,4,5,6,7\}$ を分割するための変数 y_1 を選択する. この場合, $y_1 = x_1$ とすることにより, 集合 S_{11} を $S_{21} = \{3,4,5,6\}$ と $S_{22} = \{1,2,7\}$ の 2 つの集合に分割できる. 次に, 2 つの集合 S_{21}, S_{22} を 1 つの変数 y_2 によって分割する. この場合, $y_2 = x_2 \oplus x_3$ とすることにより, 集合 S_{21} を $S_{31} = \{4,5\}$ と $S_{32} = \{3,6\}$ に, 集合 S_{22} を $S_{33} = \{7\}$ と $S_{34} = \{1,2\}$ に分割できる. さらに, 要素数 2 以上の集合 S_{31}, S_{32}, S_{34} を 1 つの変数 y_3 によって分割する. この場合, $y_3 = x_4$ とすることにより, 各集合を分割できる. この時点で, すべての集合の要素数が 1 となるため, 手続きを終了する.

以上の集合分割により, 変数 $y_1 = x_1$, $y_2 = x_2 \oplus x_3$, $y_3 = x_4$ を得る.

（例終り）

情報利得法は, 入力変数や登録ベクトル数が多い場合でも高速に処理できる.

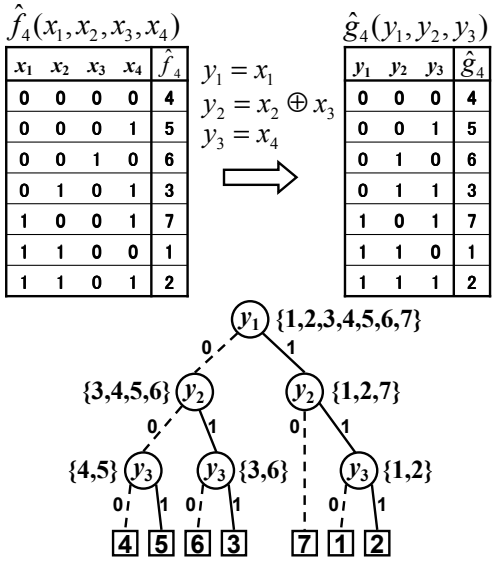

図 5 情報利得法による 4 入力関数 $\hat{f}_4$ の変数選択例

5. 実験結果

不完全定義インデックス生成関数の入力変数の削減を, **1)** 原始変数の選択のみで行う場合, **2)** 2 重化変数までを許す場合, および, **3)** 3 重化変数までを許す場合で行った. 原始変数のみ許す場合には, アルゴリズム 4.1 を用いて最適解を求めた. 多重化変数を許す場合には, アルゴリズム 4.3(情報利得法) を用いて近似解を求めた. ただし, 小規模問題に対しては, 2 重化変数を許す場合の最適解をアルゴリズム 4.1 によって求め, 情報

利得法による近似解との比較を行った．不完全定義インデックス生成関数は，(入力数 n, 登録ベクトル数 k, スキュー・ファクタ s) の各組ごとに 1000 個の関数を生成して実験を行った．ここで，s ($0 \leqq s \leqq 15$) は登録ベクトルの 0 と 1 の分布の偏りを示すパラメータであり，s の値が大きいほど，登録ベクトル中に 0 が発生する確率が高い．また，$s = 0$ のとき，0 と 1 の発生確率は等しい．登録ベクトル生成法をアルゴリズム 5.4 に示す．

［アルゴリズム 5.4］ **登録ベクトルの生成法**

（1） 閾値 Th を次のように定義する．

$Th_1 = 2^{30} - 1$,

$Th_2 = 2^{26} \times s$,

$Th = Th_1 + Th_2$

（2） 乱数 R を取得する．R の下位 31 ビットの値が Th 以上の場合は 1，Th 未満の場合は 0 のビットを生成する．

（3） 手順 (2) を n 回繰り返し，n ビットベクトルを生成する．

（4） 手順 (3) を k 回繰り返し，k 個の登録ベクトルを生成する．ただし，同一ベクトルが生じた場合，それは無視する．

実験環境は，CPU が Core 2 Duo 2.66GHz，メモリ 4GByte，OS は Windows XP Professional Service Pack 3 である．

変数削減アルゴリズムの評価には，**変数削減率**と**多重化削減率**を用いる．

［定義 5.5］ 登録ベクトル数 k，入力数 n の不完全定義インデックス生成関数を考える．削減後の入力数を $\hat{n}$ とする．変数削減数の上界 $(n - \lceil \log_2(k+1) \rceil)$ に対する，実際の削減数 $(n - \hat{n})$ の割合 r を**変数削減率**という．変数削減率 r は次のように定義できる．

$$r = \frac{n - \hat{n}}{n - \lceil \log_2(k+1) \rceil} \tag{1}$$

$r = 1$ のとき，$\hat{n} = \lceil \log_2(k+1) \rceil$ であり，最適解が求まったことを示す．$r < 1$ のときは，最適解か否かは不明であるが，r が 1 に近いときは，最適解に近い解が得られている．$r = 0$ のときは，変数を削減できなかったことを示す．

また，式 (1) 中の n を，原始変数のみ用いた場合の削減後入力数の最適解 $\hat{n}_1$ に置き換えたものを**多重化削減率**といい，変数の多重化を行った場合の変数削減効果を表す．多重化削減率 r_c は次のように定義できる．

$$r_c = \frac{\hat{n}_1 - \hat{n}}{\hat{n}_1 - \lceil \log_2(k+1) \rceil} \tag{2}$$

$r_c = 1$ のときは，最適解が得られていることを示す．$r_c < 1$ のときは，最適解か否かは不明である．r_c の値が小さいときは，原始変数のみを用いる場合の削減後入力数の最適解と，多重化を許した場合の削減後入力数の差が小さい．つまり，多重化の効果が小さいことを示す．

多重化削減率は，変数多重化アルゴリズムの性能評価に用いることができる．

5.1 入力変数の削減結果

不完全定義インデックス生成関数の各種入力変数削減方式における削減後の平均入力数，変数削減率，多重化削減率を表 1 に示す．

5.1.1 原始変数のみを用いる場合（アルゴリズム 4.1 を使用）

$s = 0$ の場合，高い変数削減率を実現できた．しかし，$s > 0$ の場合，同じ登録ベクトル数で s の値が小さい場合と比べ，変数削減率が低下している．s の値が大きいほど，各原始変数の分割差の値が大きくなり，登録ベクトルの判別に必要な変数の個数が増加するためと考えられる．したがって，原始変数のみを用いた入力変数の削減は，登録ベクトルの 0 と 1 の比率が等しい場合には多くの変数を削減できるが，登録ベクトルの 0 と 1 の比率が等しくない場合には，変数の削減の割合は減る．つまり，登録ベクトルの分布の影響を受けやすいと考えられる．

例えば，表 1 で，多重度 1，$k = 1023$ の場合，s の値を 0 から 10 に変化させると，変数削減率は，0.579 から 0 まで変化する．

5.1.2 多重化変数を許す場合（アルゴリズム 4.3 を使用）

$s = 0$ の場合，多重化変数を許す場合の変数削減率は，原始変数のみを用いる場合の変数削減率と比べて大差なかった．特に，2 重化変数までを許す場合は，k が小さい場合 ($k = 15$) を除き，多重化削減率が低くなり，多重化の効果が確認できなかった．多重化変数を許す場合の変数選択には情報利得法を用いているため，登録ベクトル数が多い場合には分割対象となる集合数が増加し，変数削減効果が低下すると考えられる．しかし，$s > 0$ の場合には，多重化変数を用いることにより，原始変数のみを用いる場合よりも多くの変数を削減できた．これは，変数の多重化により，分割差の小さい多重化変数を生成できたためと考えられる．つまり，多重化変数を用いれば，登録ベクトルの分布にかかわらず，多くの変数を削減できる．

5.1.3 計 算 時 間

$s = 0$ の場合の各種入力変数削減方式における平均実行時間を図 6 に示す．アルゴリズム 4.1 を用いた，原始変数のみを用いる場合の実行時間よりも，アルゴリズム 4.3 を用いた，多重化変数を許す場合の実行時間が短く，情報利得法が高速なアルゴリズムであることが確認できる．

5.2 最適解とヒューリスティック解の比較

アルゴリズム 4.3 で得られた解 (ヒューリスティック解) の最適性を確認するため，アルゴリズム 4.1 で得られた最適解と比較し，相対誤差を求めた．ただし，アルゴリズム 4.3 については，2 変数ずつの選択で集合分割を最適化した改良手法を用いた．$n = 24$, $k = 15$ の場合に関して，2 重化変数までを許す場合の最適解とヒューリスティック解の平均入力数，最適解に対するヒューリスティック解の相対誤差を表 2 に示す．2 重化変数までを許す場合の最適解と比較すると，ヒューリスティック解の相対誤差は最大で 0.118 であった．従って，情報利得法を改良すれば，さらに多くの変数を削減できる可能性はある．

2 重化変数までを許す場合の計算時間は，$s = 10$ のときで，最適解が最大 575.171[sec]，ヒューリスティック解が最大 0.047[sec] であった．k が大きい場合には，最適解を求めるのにさらに多くの計算時間が必要になると考えられる．したがって，実用上はヒューリスティック解を求める必要がある．

表 1　平均入力数, 変数削減率, 多重化削減率 ($n = 24$)

k	s	多重度	平均入力数	変数削減率	多重化削減率
15	0	1	4.882	0.956	-
		2 以下	4.209	0.990	0.763
		3 以下	4.000	1.000	1.000
	5	1	4.997	0.950	-
		2 以下	4.299	0.985	0.700
		3 以下	4.000	1.000	1.000
	10	1	6.432	0.878	-
		2 以下	4.905	0.955	0.628
		3 以下	4.059	0.997	0.976
63	0	1	7.996	0.889	-
		2 以下	7.968	0.891	0.014
		3 以下	7.334	0.926	0.332
	5	1	8.936	0.837	-
		2 以下	7.983	0.890	0.325
		3 以下	7.381	0.923	0.530
	10	1	13.480	0.584	-
		2 以下	9.448	0.808	0.539
		3 以下	7.965	0.891	0.737
255	0	1	11.852	0.759	-
		2 以下	11.819	0.761	0.009
		3 以下	11.000	0.813	0.221
	5	1	13.212	0.674	-
		2 以下	12.001	0.750	0.232
		3 以下	11.006	0.812	0.423
	10	1	21.952	0.128	-
		2 以下	15.543	0.529	0.459
		3 以下	12.225	0.736	0.697
1023	0	1	15.889	0.579	-
		2 以下	15.921	0.577	-0.005
		3 以下	15.012	0.642	0.149
	5	1	18.248	0.411	-
		2 以下	16.351	0.546	0.230
		3 以下	15.023	0.641	0.391
	10	1	24.000	0.000	-
		2 以下	20.395	0.258	0.258
		3 以下	16.024	0.570	0.570

(多重度 1 の場合：アルゴリズム 4.1, 多重度 2 以下・3 以下の場合：アルゴリズム 4.3)

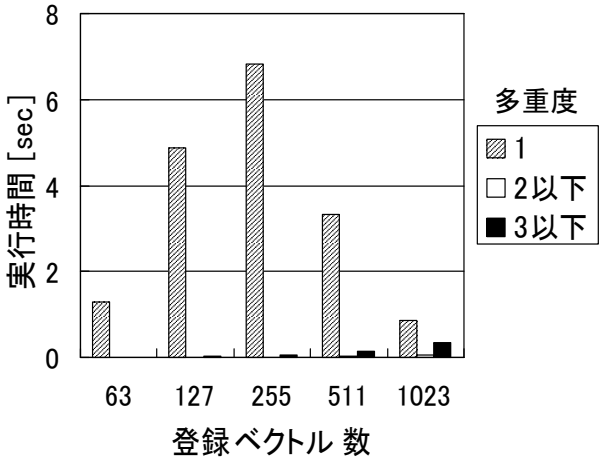

図 6　平均実行時間 ($s = 0, n = 24$)
(多重度 1 の場合：アルゴリズム 4.1, 多重度 2 以下・3 以下の場合：アルゴリズム 4.3)

6. ま と め

本論文では, 不完全定義インデックス生成関数の入力変数削減手法について述べた. 不完全定義インデックス生成関数では, 入力変数の選択を工夫することにより, 変数を削減できる. また, 変数の多重化により, 必要な変数の個数をさらに削減できる. しかし, 最適な多重化変数を求めるのは困難であるため, 本論文では, ヒューリスティック法として情報利得法を提案した. 情報利得法では, 登録ベクトル数が多い場合には変数削減効果が低下するが, 登録ベクトルの 0 と 1 の比率の偏りの影響は小さい. また, 高速に実行できる. 情報利得法によって選択した多重化変数を用いた場合, 不完全定義インデックス生成関数の入力変数の削減に十分な効果があることが実験的に確認できた. ただし, 最適解と比較した場合, 平均入力数の相対誤差が 0.1 程度になる場合がある.

表 2　2 重化変数までを許す場合の最適解とヒューリスティック解の平均入力数の比較 ($n = 24, k = 15$)

s	A：最適解	B：ヒューリスティック解	$(B - A)/A$：相対誤差
0	4.000	4.033	0.008
5	4.000	4.088	0.022
10	4.339	4.851	0.118

(最適解：アルゴリズム 4.1, ヒューリスティック解：アルゴリズム 4.3)

多重化変数の回路実現を行う場合には, ハードウェア量や遅延時間の問題がある. 従って, 2 重化変数までが実用的と考えられる.

今後の課題としては, 情報利得法よりも良い解を得ることができる多重化変数選択アルゴリズムの開発や, インデックス生成回路をハードウェアに実装した場合の性能評価などがある.

7. 謝　　辞

本研究は一部, 日本学術振興会科学研究費補助金, および文部科学省・知的クラスタ創成事業（第 2 期）の補助金による.

文　　献

[1] C. Halatsis and N. Gaitanis, “Irredundant normal forms and minimal dependence sets of a Boolean function,” *IEEE Transactions on Computers*, Vol. C-27, No. 11, pp. 1064-1068, November, 1978.

[2] Y. Kambayashi, “Logic design of programmable logic arrays,” *IEEE Transactions on Computers*, Vol. C-28, No. 9, pp. 609-617, September, 1979.

[3] B. M. E. Moret, “Decision trees and diagrams,” *ACM Computing Surveys*, Vol. 14, No. 4, pp. 593-623, December, 1982.

[4] T. Sasao, “On the number of dependent variables for incompletely specified multiple-valued functions,” *International Symposium on Multiple-Valued Logic(ISMVL)*, pp. 91-97, May, 2000.

[5] T. Sasao, “On the numbers of variables to represent sparse logic functions,” *International Conference on Computer Aided Design (ICCAD)*, pp. 45-51, November, 2008.

[6] D. E. Taylor, “Survey and taxonomy of packet classification techniques,” *ACM Computing Surveys*, Vol. 37, Issue 3, pp. 238-275, September, 2005.

[7] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, “Scalable high speed IP routing lookups,” *ACM SIGCOMM Computer Communication Review*, Vol. 27, No. 4, pp. 25-36, 1997.