最小被覆問題を解くアルゴリズム*

阿部 圭一** 福村 晃夫**

Abstract

A combinatorial algorithm to solve minimal cover problems is described here. These problems are encountered in minimization of Boolean functions and considered as the most simple but the most difficult ones among integer programming problems. Besides its own profit, the construction of the algorithm to solve them brings us many useful informations about solving the more general integer programming problems. The algorithm was written in the form of a FORTRAN program and several examples were solved by a digital computer. The solution of the problem of size 30×30 can be obtained in a minute or so.

1. 序

近年，整数型計画法あるいは整数線型計画法の組合せ論的解法 (combinatorial method) に興味が持たれ，種々のアルゴリズムが発表されているが，まだ，決定的なアルゴリズムは見い出されていないように思われる．その最大の理由は，この問題が，後述のように，いわゆる heuristics の問題の典型的なものであり，あるアルゴリズムの良否が，対象となる問題の性質に大きく依存するという点にある．しかも，この性質は，一般的なアルゴリズムであればあるほど著しい．したがって，対象となる問題の性質を限定して，いわば，対象に密着した手法というものが開発されても良いと考えられる．このような，対象を限定した手法は，その対象にしか適用できないという欠点をもつ反面，その特定の問題については，一般的手法よりもはるかに速く最適解を得ることができる．また，このように特定の問題に限定してアルゴリズムを考えることにより整数型計画法の組合せ論的解法に存在する問題点を，より明確にすることができるのではないかと考えられる．

ここでは，アルゴリズムの対象を，最小被覆問題 (minimal cover problem) に限定する．最小被覆問題は，一般の整数型計画法あるいは整数線型計画法からみればきわめて特殊な問題ではあるが，一般の整数型計画法の組合せ論的解法を考えるさいに生ずる問題点はすべて備えている．さらに，最小被覆問題の構造は最も単純なものであるため，解法の記述には枝葉末節を省略して本質的な点のみを述べることができる．したがって，この問題のアルゴリズムを考察することは一般的見地から十分意味があると思われる．一方，この問題は，組合せ論的解法では最も解きにくい型の問題でもあるため，この問題専用の高速アルゴリズムを考える実際的理由も存在する．

最小被覆問題は，最初，論理回路の簡単化の問題から発生し，現在まで主としてその面から取り組まれているが[1]，その他にも，割当問題，論理回路の診断などでこの問題に帰着されるものが，いくつか知られている[2)3)]．

筆者は，最小被覆問題を解くアルゴリズムを開発するだけでなく，それをディジタル計算機のプログラムとして作成し，実際に計算機にかけて問題をいくつか解くところまでを目標とした．これまで，各種の整数型計画問題の組合せ論的アルゴリズムが発表されているが，実際にプログラム化し，問題を解いてみてその有効性を評価したものは少ないように思われる．したがって，アルゴリズムをプログラム化し，問題を実際に解いてみることによって，組合せ論的解法の新たな問題点が見つけられる可能性がある．事実，すでに危惧されていたことではあるが，7. で述べるように，heuristics としての問題が非常に重大で，紙の上のアルゴリズムでは一見改良されたように見えても，実は改良になっているかどうかわからないという状況が起こりうることが実験的に確認された．

* An Algorithm to solve minimal cover problems, by Keiichi Abe and Teruo Fukumura (Faculty of Engnieering, Nagoya University)

** 名古屋大学工学部

2. 問題の定式化

最小被覆問題はつぎのように定式化される.

「条件

$$\sum_{j=1}^{n} c_{ij}x_j \geqq 1 \quad (i=1,2,\cdots\cdots,m) \qquad (1)$$

ここに, $c_{ij}=0$ または 1 $(i=1,\cdots\cdots,m;\ j=1,\cdots,n)$,

$$x_j=0 \text{ または } 1 \quad (j=1,2,\cdots\cdots,n) \qquad (2)$$

のもとで, 目的関数

$$y=\sum_{j=1}^{n} x_j \qquad (3)$$

を最小にする $(x_1, x_2, \cdots\cdots, x_n)$ を求めよ*.」

以下では, 式(2)の条件は暗黙の了解として省略し式(1)の形の条件式のみを「条件」とよぶ. また

$$X=(x_1, x_2, \cdots\cdots, x_n) \qquad (4)$$

を解ベクトルとよび, これが式(1)を満たすならば可能解, さらに可能解のうちで式(3)を最小にするものを最適解とよぶ. なお, 本論文では, 最適解が複数個存在する場合には, そのすべてを求める必要はないものと仮定する.

3. 例 題

本章では, つぎのような例題をとりあげて, その人手による解法を説明する.

「条件

$$\left.\begin{aligned} &x_1 \geqq 1,\\ &x_1+x_2 \geqq 1,\\ &x_2+x_3+x_4 \geqq 1,\\ &x_2+x_3 \geqq 1,\\ &x_3+x_6 \geqq 1,\\ &x_4+x_5 \geqq 1,\\ &x_5+x_6 \geqq 1,\\ &x_4+x_6 \geqq 1 \end{aligned}\right\} \qquad (5)$$

のもとに, 目的関数

$$y=x_1+x_2+x_3+x_4+x_5+x_6 \qquad (6)$$

を最小にせよ.」

この問題を解くには, 係数 c_{ij} を要素とする**第1表**(a)のような行列を用いるとよい. 上の問題は, この表を用いて, つぎのようにいいかえることができる.

「表のすべての列からなる集合の任意の部分集合を J とする. すべての行について J に属するいずれかの列との交点に1が存在するという条件を満たす集合 J のうち, それに属する列数が最小であるものを求めよ*」

このとき, $x_j=1$ は第 j 列が集合 J に属することを, $x_j=0$ は第 j 列が集合 J に属しないことを意味する.

第1表 例題の解法

(a)

行番号	列番号	1	2	3	4	5	6
	状 態						
1		1	0	0	0	0	0
2		1	1	0	0	0	0
3		0	1	1	1	0	0
4		0	1	1	0	0	0
5		0	0	1	0	0	1
6		0	0	0	1	1	0
7		0	0	0	0	1	1
8		0	0	0	1	0	1

(b)

行番号	列番号	1	2	3	4	5	6
	状 態	+1					
1	+1						
2	+1						
3	−1						
4			1	1	0	0	0
5			0	1	0	0	1
6			0	0	1	1	0
7			0	0	0	1	1
8			0	0	1	0	1

(c)

行番号	列番号	1	2	3	4	5	6
	状 態	−1	−1	+1			
1	+1						
2	+1						
3	−1						
4	+1						
5	+1						
6					1	1	0
7					0	1	1
8					1	0	1

最小被覆問題を解くには, 行状態ベクトル

$$U=(u_1, u_2, \cdots\cdots, u_m)$$

$$u_i=0 \text{ または } \pm 1 \quad (i=1,\cdots\cdots,m) \qquad (7)$$

および, 列状態ベクトル

$$V=(v_1, v_2, \cdots\cdots, v_n)$$

$$v_j=0 \text{ または } \pm 1 \quad (j=1,\cdots\cdots,n) \qquad (8)$$

* 本論文では, 最小被覆問題をこの形の問題に限定する. これに対し条件は同じで目的関数が $y=\sum_{j=1}^{n} w_j x_j$ の形の問題を「重みつきの最小被覆問題」という. これの解法については 8. で触れる.

* 論理回路の簡単化の場合には, ここでの行にあたるものを最小項, 列にあたるものを主項という. 通常は主項を縦に, 最小項を横に並べるため, ここでの記述とは行, 列が逆になる. 本論文の記述を慣習と逆にしたのは, 計算機プログラムにおける解の印刷の便宜のためである.

を用いる．最初，これら2つのベクトルは零ベクトルとし，表には0は記入しないで空白にしておく．

第1表(a)を見ると，第1行には1が1個（第1列との交点）しか存在しない．これは，第1列がかならず集合 J に含まれなければならないことを意味する．このような列を essential term（以後略して ET と書く），また，第1行に相当する行を以後 ET を生ずる行とよぶ．第1列を集合 J に入れるという意味で第1列の状態を +1 に変え，第1列を消す．第1列を集合 J に入れれば，第1行の条件だけでなく，第2行の条件も満たされる．したがって，第1，2行の状態を +1 に変え，これらの行を消す．

つぎに，表の第 3，4 行を見ると

$$c_{3j} \geqq c_{4j} \quad (\text{for all } j) \tag{9}$$

という関係が成立している．このような関係が成立するとき，第3行は第4行に優越するといい，この関係を row dominance（以後略して RD と書く）とよぶ．RD が存在するとき，優越する行に対応する条件は考える必要がない．なぜなら，それは，優越される行の条件を解が満たしさえすれば，自動的に満たされるからである．よって，第3行に，考える必要がないという意味で −1 をつけ*，この行を消す．以上の操作を行なった後の表を第1表(b)に示す．

第1表(b)を見ると，すでに消された行を除いては第2列と第3列のあいだに

$$c_{i3} \geqq c_{i2} \quad (\text{for all } i) \tag{10}$$

という関係が成立している．このとき，第3列は第2列に優越するといい，この関係を column dominance（以後略して CD と書く）とよぶ．なお，この CD は最初の第1表(a)では存在せず，ET の存在によって第 1，2 行を消したのちに初めて生じたことに注意する．CD が存在するとき，優越される列を選んで満たすことのできる条件は，すべて，優越する列を選んでも満たすことができるはずである．したがって，ここでは，すべての最適解を網羅する必要はないと仮定しているから，優越される列は集合 J に入れられることはないとしてよい．よって，第2列に −1 をつけ，第2列を消す．そうすると，第4行に ET が生ずるから，ET である第3列に +1 をつけて消し，それによって満たされる第 4，5 行に +1 をつけ，これらも消す．

このような操作の結果，第1表(c)が得られる．以上の操作を表の簡約とよぶ．第1表(c)には，もはや ET，RD，CD のいずれも存在せず，これ以上簡約できない．これを既約な表とよぶ．既約な表が残ったとき，真の意味での整数型計画法が始まる．以下，これを branch and bound 法[4]で解いてみよう．

残った行のうち，任意に1つの行を選んで +1 をつける．ここでは第6行としよう．第6行の条件を満たすためには，第 4，5 列のいずれか少なくとも一方を選ばなければならない．まず，第4列を選んでみる．第4列に +1 をつけた結果，第8行の条件も満たされるから，残るは第7行の条件のみとなる．これを満足させるためには，さらに第 5，6 列のいずれか一方を選べばよい．つぎに，第1表(c)にもどって，第6行の条件を満足させるために第5列を選んだ場合を考える．第5列の選択により，第7行の条件も満たされるから，残る第8行の条件を満足させるために，さらに第 4，6 列のいずれかを選ぶことになる．

上の解法の結果，つぎの3種類の列状態ベクトルが得られた（次式の最初のベクトルは，重複してえられる）．

$$\begin{aligned}&(+1, -1, +1, +1, +1, 0),\\ &(+1, -1, +1, +1, 0, \ \ +1),\\ &(+1, -1, +1, 0, \ \ +1, +1)\end{aligned} \tag{11}$$

これを解ベクトルに変換するには，次式の変換を行なえばよい．

$$\begin{aligned}&x_j = 1 \text{ if } v_j = +1\\ &x_j = 0 \text{ if } v_j = 0 \text{ or } -1\\ &\quad (j = 1, 2, \cdots\cdots, n)\end{aligned} \tag{12}$$

したがって，上の例題の可能解として

$$\begin{aligned}&(1\ 0\ 1\ 1\ 1\ 0),\ (1\ 0\ 1\ 1\ 0\ 1)\\ &(1\ 0\ 1\ 0\ 1\ 1)\end{aligned} \tag{13}$$

が得られた．これらにたいする目的関数の値はすべて4であるから，これらがすべて最適解であることは明らかである（もし，解の与える目的関数値が異なれば最小の値を与える解のみが最適解である）．

4. アルゴリズムの考察

本章では，前章の例題で述べた解法を厳密なアルゴリズムの形にすること，およびそのさいにできるだけ能率のよいアルゴリズムにすることを考察する．

4.1 表の簡約

上の例題で知られるように，与えられた表をそのまま整数型計画問題として解くのは得策でない．まず

(i) ET による表の簡約

---

* 実は，行に関しては，状態 +1 と −1 を区別する必要は全くない．

(ii) RD による表の簡約

(iii) CD による表の簡約

ができないかどうかを検査し，簡約できるならばできる限り簡約して，既約な形にしてから整数型計画問題として扱うべきである．

ここで注意すべきことは，例題で出会ったように，最初に与えられた表において CD が存在しないとしても，もし，ET が存在して表を簡約したとすれば，新たに CD が生ずる可能性のあることである．ET, RD についても同様のことがいえる．したがって，与えられた表を簡約するには，上述の(i)，(ii)，(iii)の操作を順ぐりに繰り返し，どの1つによる簡約もできなくなるところまで行なう必要がある．

しかし，ET は1つの行単独の性質であるが，RD (CD)は2つの行（列）相互間の性質であるから，ET については m 個の行の検査を，RD については $m(m-1)/2$ 回の行の対の比較を，CD については $n(n-1)/2$ 回の列の対の比較を，各操作へもどる度ごとに行なわなければならない．これは莫大な計算時間を費すことになるであろう．

この対策としては，つぎのようなことが考えられる．なるほど，最初の1回は，すべての行について ET を，すべての行の対について RD を，すべての列の対について CD を調べる必要があるが，その後生ずる，たとえば，RD は，CD に起因して生ずるもののみであるから，すぐ前に生じた CD によって RDがひきおこされる可能性のある行の対についてのみ RD の存在を調べればよい．ET, CD についても同様のことがいえる．

この方針にもとづいて，各操作が他の操作の可能性へ影響を及ぼす範囲を限定しよう．**第1図**の模式図からつぎの関係が知られる．

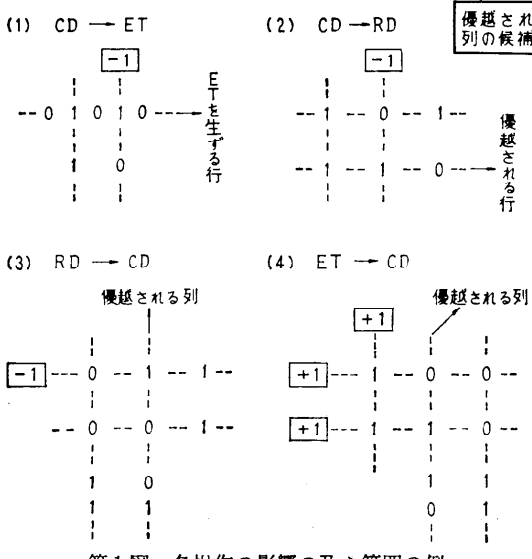
(1) CD による表の簡約操作で，優越される列に -1 をつけることにより，その列との交点に1をもつ行は ET を生ずる行になる可能性がある．

(2) CD による表の簡約操作で，優越される列に -1 をつけることにより，その列との交点に1をもつ行を優越される行とする RD が生ずる可能性がある．

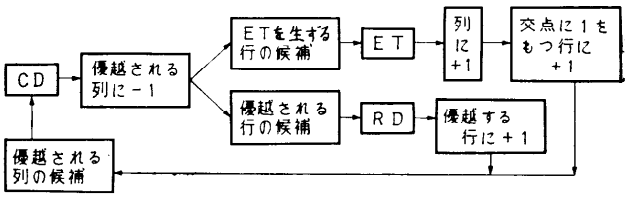
(3) RD による表の簡約操作で優越する行に -1 をつけることにより，その行との交点に1をもつ列を優越される列とする CD が生ずる可能性がある．

(4) ET による表の簡約操作では，ET である列およびその列との交点に1をもつすべての行に $+1$ がつけられる．この操作によって，これらの行との交点に1をもつ列を優越される列とする CD が生ずる可能性がある．

(5) ET による表の簡約によって RD が生ずることはない．また，RD による表の簡約によって ET が生ずることはない．



第1図　各操作の影響の及ぶ範囲の例



第2図　ERC ルーチン

これらの関係を図式的に表わすと，**第2図**のようになる．(1)，(2)より明らかなように，CD から得られる，ET を生ずる行の候補と優越される行の候補とは，全く同じ集合になるから，それらの記憶は共通にすることができる．この事実を考慮して，ここで述べるアルゴリズムでは，ET→RD→CD→ET→……の順に各操作を繰り返すことにした．これを ERC ルーチンとよぶ．

4.2 branch and bound 法

ERC ルーチンによって与えられた表を簡約し，既約な表が残ったならば，いよいよこれを整数型計画問題として解く．解法は，いわゆる branch and bound 法[4)5)]による．

ここで用いる branching の方法は，**3.** の例題の解

法で述べた，条件に注目して分岐する方法である．他の方法としては，文献[5]に述べられている，変数に注目して分岐する方法がある．両者は一長一短があるが，問題を最小被覆問題に限定してこれを能率的に解くには，条件に注目した分岐法のほうがすぐれていると思われる．

さて，例題の解法で述べた branching は一般的につぎのように述べることができる．

(1) まだ±1のついていないある行 i^* を選び，+1 をつける．

(2) 行 i^* との交点に1をもち，まだ±1のついていない列の中から，いずれか1つの列 j^* を選び，+1 をつける．

(3a) 列 j^* を選ぶことにより行状態ベクトルの要素がすべて±1になれば，可能解に到達したのであるから，解を記録し，前の段へもどって別の列へ分岐する．もし，分岐すべき列がもはや残っていないならば，さらに前の段へもどる．

(3b) 列 j^* を選んでも行状態ベクトルの要素に0が残っているならば，つぎの段へ進んでさらに分岐を行なう．

ここで，(2)において列 j^* を選び，+1 をつけた直後の状態を考えてみよう．それは，ちょうど第2図の ET 操作によって列に +1 がつけられた直後の状態と同じである．したがって，列 j^* に +1 をつけたことに起因して CD が起こり，それによって ET または RD が起こり，再び CD が起こり，……となる可能性がある．そうであるとすれば，最初に与えられた問題に直ちに branch and bound 法を適用すべきでないのと同様に，ここでもまた直ちにつぎの段へ進んで分岐を行なうのは得策でない．すなわち，(2)より直ちに(3a)または(3b)へ進むべきでなく，つぎの(2)′を経ていくべきである．

(2)′ ERC ルーチンを実行する．

いいかえれば，分岐した後，ERC ルーチンによってできるかぎり行および列の状態を決定して解の存在範囲を狭めてから，つぎの段の分岐を行なうべきである．

つぎに，1つの列を選んで分岐した後，再びその分岐点へもどってきたときを考える．そのさいには，まだ選択されていない別の列へ分岐するわけであるが，このとき，すでに選択された列は，その列をもはや集合 J には入れない状態 −1 にすべきである．なぜなら，これを怠ると，例題の解の (+1, −1, +1, +1, +1, 0) という状態のように，1つの状態が，分岐の2つの枝のそれぞれから二重に得られることになる．これは解が二重に得られるというわずらわしさを生ずるのみではなく，その解に至るまでの branching 操作や ERC ルーチンの一部でも，全く同じことを二重に行なうことになり，計算の著しい非能率をも引き起こす．

以上の理由から，すでに選択された列の状態は −1 にすべきであるが，このとき，新しい列を選んで +1 をつけることによってのみでなく，すでに選択された列に −1 をつけることによっても，ERC ルーチンによって表の簡約が行なわれることに注意する必要がある．

branch and bound 法のもう1つの特徴である bounding 操作は，branching によって生ずるぼう大な tree の探索を打ち切るのに有効である．通常，それは現在の列状態ベクトルからえられる解の与える目的関数値の下限が，すでに得られた解の与える目的関数値を越えるならば，その先の branching は打ち切り，直前の分岐点までもどって別の列を選択することによって行なわれる．最小被覆問題では，目的関数において各変数にたいする重みに差異がないため，解の与える目的関数値は似たような値をとりやすく，そのため bounding の効果はあまり発揮されない．このことを考慮して，ここでのアルゴリズムでは，面倒な下限の計算は行なわず，すでに得られた可能解の与える目的関数値と，現在 +1 がつけられている列の個数とを比較して bounding を行なっている．後者が，その状態ベクトルから得られる解にたいする目的関数値の下限になっていることは自明であろう．さらに，良い下限の評価法は存在するが，**7.** で述べる heuristics としての問題点があるため，その採用によりアルゴリズムがより能率的になるかどうかは不明である．

以上，述べた branch and bound を行なうルーチンを B&B ルーチンとよぶ．

5. プログラム化の方針

前章に記述されたアルゴリズムをディジタル計算機のプログラムとして実現するため，筆者はつぎの方針をとった．

係数 c_{ij} や解 x_j は 0, 1 の 2 値であり，行，列状態ベクトルは 0, ±1 の3値であるから，これらを計算機の1語の各 bit へ格納して bit 処理命令を用いて処理することも考えられ，これを上手に利用すれば

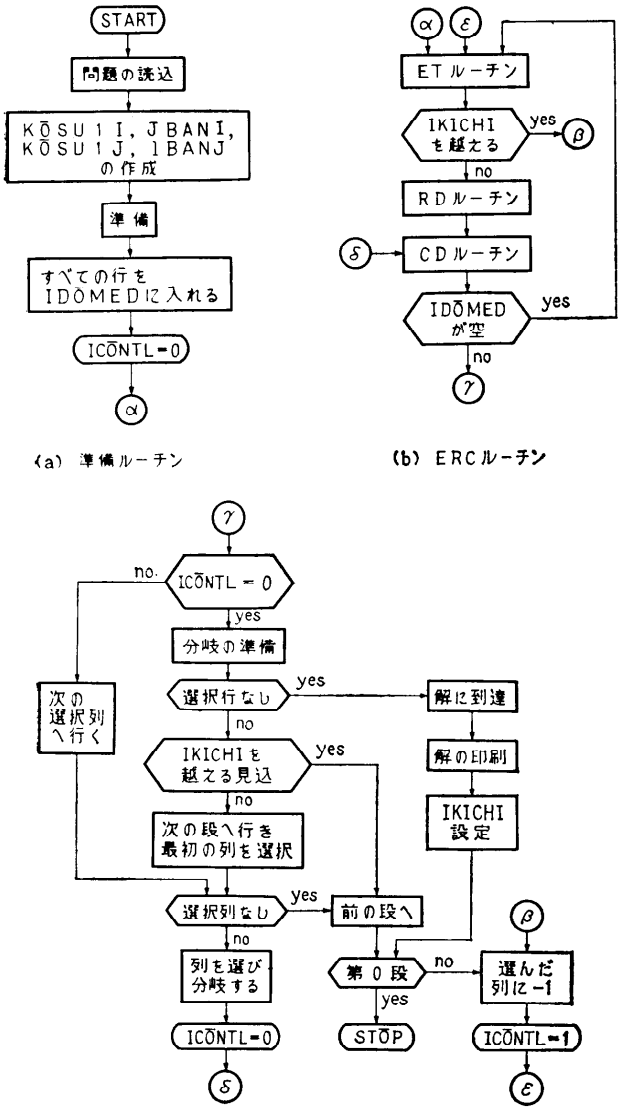
高速のプログラムを作ることもできるが，ここでは，bit 処理は採用しなかった．その最大の理由は，bit処理を行なうためにはアセンブラ語でプログラムを作成しなければならず，したがって，異なる計算機間で全く互換性がないという点にある．ここでは，機種間の互換性を考慮し，JIS FORTRAN 水準 7000 に従ってプログラムを作成した．

さらに，bit処理を用いないならば，c_{ij} を必ずしも 0，1 の表の形で表現する必要はない．むしろ，式(5)のような表現のほうが，記憶容量，計算時間の両面ですぐれている．すなわち，第 I 行にある 1 の個数を記憶した 1 次元配列 KŌSU1I(I) と，第 I 行の左から K 番目の 1 のついている列の番号を記憶した 2 次元配列 JBANI (I, K) を用意する方法である．ただし，このプログラムでは，さらに，第 J 列にある 1 の個数を記憶した 1 次元配列 KŌSU1J(J) と，第 J 列の上から K 番目の 1 のついている行の番号を記憶した 2 次元配列 IBANJ(J, K) をも用意してある．このように二重に表を記憶しているのは手順の能率化のためである．

この計算機プログラムでは，前章で述べたいくつかの機能をすべて開いたサブルーチンとして組み込む方針をとった．その大まかなフローチャートを**第3図**に示し，以下，それを簡単に説明する．

図中の変数 IKICHI はすでに得られている可能解の与える目的関数値，IDŌMED は ET を生ずる行の候補および優越される行の候補を共通に入れておく配列，ICŌNTL は，**4.2** で述べたように，すでに選択された列に −1 をつけたとき ERC ルーチンへ飛んで，しかも再び元の所へもどるように制御するための変数である．

準備ルーチンでは，IDŌMED，すなわち，ET をもつ行の候補ならびに優越される行の候補を表わす配列へすべての行を入れてから ERC ルーチンへいく．これによって，最初の 1 回のみは，すべての行について ET が，すべての列の対について RD が検査される．このプログラムでは，CD についてはすべての列の対の検査は行なわない．これは，論理回路の簡単化の場合には，最初の表として自動的に CD の存在しないものがえられるし，他の場合でも，問題の解析のさいに少し注意すれば CD の存在しない表が得られることに基づいている．



第3図　計算機プログラムのフローチャート

6. 計算例

いくつかの問題を解いた結果を要約して**第2表**に示す．計算時間は HITAC 5020 E によるものである．なお，この計算を行なった時点では，第3図のフローチャートから，RD 操作，および，すでに，選択された列に

第2表　計算例の要約

問題番号	行数 m	列数 n	1の個数*	最適値**	解の個数***	計算時間(秒) CPU time	計算時間(秒) USE time
1	3	3	2	2	2	0	0.167
2	6	6	2	3	2	0	0.251
3	11	11	2.3	5	2	1	0.447
4	26	26	3.8	8	9	0	0.990
5	34	33	2.9	13	2	2	1.104
6	35	15	3	8	10	2	2.689
7	30	30	3	8	6	3	3.052
8	30	30	3	9	3	3	3.332
9	50	50	3	16	246	229	232
10	50	50	3	15	27	170	171

* 1行あたりの1の平均個数
** 最適解にたいする目的関数値
*** このプログラムで得られた最適解の個数

-1 をつけた後 ERC ルーチンを実行する操作は除外されている．問題はすべて既約な形で与えられており，問題1～5は論理関数の簡単化より得られたもの，6～10は乱数を用いて直接人為的に作られたものである．両者でやや性質の異なることが，第2表の計算時間からもうかがわれる．

7. heuristics としての問題

現在，人工知能のいくつかの分野で heuristics が問題になっている．heuristics の定義についてはいくつかの説があるが，要するに，人間の直観，洞察，大局的把握といった機能をいかに人工知能で実現するかという問題と思われる．その意味で，整数型計画法の組合せ論的解法は，典型的な heuristics の問題を提供する．

整数型計画法の組合せ論的解法は，結局のところ，いかにして能率良くしらみつぶしを行なうかという問題に帰着する．能率を上げるためにとりうる1つの方法は，branching のさいに，できるだけ先の見通しをつけてから分岐する方法である．ERC ルーチンの利用もその1つであるが，他にも種々の工夫が考えられる．たとえば，ERC ルーチンによってできるだけ表が簡約されるような行を選んで分岐を行なうとか，できるだけ分岐の枝の少ない行を選んで行なうとかである．このような先の見通しをつける手順を付加すれば必ずアルゴリズムは改良され，計算時間は短縮されるように思われるが，果たしてそうであろうか．

計算時間が短縮されるか否かは，大ざっぱにいえば先の見通しをつけるために要する時間と，それによって節約される時間の平均値との大小関係によって決まる．しかし，ここで注意しなければならないことは，先の見通しをつけるために要する時間は，この操作を行なうごとに消費されるのにたいし，一般には，それによって必ず時間の節約が生ずるとは限らないことである．したがって，この大小関係は必ずしも自明でなく，また，問題によって著しく変動する．それゆえ，計算時間が短縮されるかどうかは問題ごとにきわめて異なり，よほど顕著なアルゴリズムの改良でないと，平均的にみて計算時間が短縮されるか否かは判断が下だしにくいのである．

しかも，人間は先の見通しをつけるというような大局的判断は非常に速いのにたいし，計算機ではこのような判断も1つ1つの論理動作に分解して行なわなければならないため，相対的に大局的判断の実行速度は遅くなる．したがって，人間が利用して解法の高速化をはかることのできる手順を，そのまま計算機に実行させたとしても同様な高速化が得られるとは限らないのである．

一例を挙げてみよう．branching のさい，行はすでに選ばれたとして，分岐すべき列を選ぶさいに，任意の順序に選ぶよりも，その列に存在する1の個数（すなわち，その列が満足する条件の個数）の大きい列から順に選ぶほうが，平均的には早く目的関数値の小さい解に到達し，したがって，より早い段階で bounding を行なうことができるはずである．これを行なうようにプログラムを改造した後の計算時間と改造前の計算時間を比較したのが**第3表**である．改良されたはずのプログラムが改良になっていないこと，問題によってその様子が非常に異なることが読みとられる．

第3表　プログラム改造の効果

問題番号	計算時間(秒) 改造前 CPU time	改造前 USE time	改造後 CPU time	改造後 USE time
1	0	0.167	0	0.188
2	0	0.251	1	0.176
3	1	0.447	0	0.355
4	0	0.990	3	2.887
5	2	2.689	1	3.584

現在作成したプログラムに到達するまでにいくつかのプログラムを作成したが，上記の理由で，明らかに計算時間の大幅な短縮に寄与したのは，**4.**で述べた，分岐後必ず ERC ルーチンを通すことと，すでに探索し終わった列に -1 をつけて二重探索を防ぐことの2つの操作のみであった．

組合せ論的解法の能率を上げるもう1つの手段は，

bounding を能率良く行なうことであるが，これについても上と同様の問題が存在する．

8. むすび

最小被覆問題を解くアルゴリズムを考察し，それを計算機プログラムとして実現した．これによって，整数型計画法の組合せ論的解法のもつ問題点が明らかにされ，また，解法にたいする指針が与えられた．

本プログラムで数個の問題を解いた結果と Gomory の方法で整数線型計画法を解いた結果[1]とを比較してみると，問題の性質の差異，計算機の違いなどのため大まかな比較であるが，30×30 の既約な表程度までは大差なく，それより大きな表では Gomory の方法のほうが速いようである．しかし，この範囲では，問題の性質の差異が大きく影響していると考えられる．

このプログラムには，まだいくつかの改善可能な点が残されているが，前章で論じた heuristics としての問題が存在するため，決定的な改良の方策は立っていない．

他のより一般的な 0-1 変数整数型計画法への拡張の可能性については，**1.** で論じた方向に沿って考える必要があろう．まず，重みつき最小被覆問題への拡張はほとんど問題がない．アルゴリズムで変更すべき点は，CD が存在しても，優越される列の重みが優越する列の重みより大ならば，前者を消すわけにはいかないことと，bounding を重みを考慮して行なわねばならないことのみである．それ以外の問題への拡張はアルゴリズムの大幅な変更を必要とする．しかし，最小被覆問題のアルゴリズムの考察によって得られた種々の指針は，他の問題のアルゴリズムの考察にも大いに役立つと思われる．

謝 辞 この問題の興味深い点に筆者の注意を向けてくださった，名古屋大学大学院工学研究科の吉田雄二，香村 求（現在，日本電信電話公社勤務）の両氏に感謝する．なお，本論文中の計算には東京大学大型計算機センターを利用したことを付記します．

参考文献

1) A. Cobham, R. Fridshal, J. H. North: "A Statistical Study of the Minimization of Boolean Functions Using Integer Programming", IBM Research Report, RC-756, 1962.
2) G. B. Dantzig: "Linear Programming and Extensions", pp. 319-321, Princeton Univ. Press, 1963.
3) F. Hadlock: "On Finding a Minimal Set of Diagnostic Tests, IEEE Tr., Vol. EC-16, p. 674 (Oct. 1967).
4) E. L. Lawler, D. E. Wood: "Branch and Bound Methods: A Survey", Opns. Res., Vol. 14, p. 699 (1966).
5) 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，50 巻，p. 1955(昭 42. 10).