有理数演算によるゴモリーの小数法の基礎的検討†

大柳俊夫†† 大内東††

全整数計画問題（IP 問題）の解法である Gomory の小数法は，カットと呼ばれる制約式を有限個加えて最適解を得ることを保証した方法である．しかし，この方法を浮動小数点演算を用いてインプリメントした場合，最適解が得られないことが頻繁に起こることが知られており，この方法の有効性については多くの研究者が否定的な意見である．最適解が得られない理由は，桁落ちや丸め誤差と考えられているが，そのことを明らかにした報告は見当らない．また，桁落ちや丸め誤差を排除し，理論どおりにインプリメントした場合の小数法の振舞いについても明らかにされていない．桁落ちと丸め誤差を排除する演算としては有理数演算がある．この演算は，計算機の性能の制約から今まではほとんど用いられていなかったが，最近の計算機の急速な進歩により次第に利用可能となりつつある．本論文は，有理数演算を用いて小数法をインプリメントし，(1)桁落ちと丸め誤差を排除した場合の小数法の振舞い，(2)浮動小数点演算を用いた場合に生じる桁落ちと丸め誤差の結果への影響，(3)カットの生成・除去方法の違いによる小数法の効率の違い，を調べたものである．その結果，小数法に特有の性質と浮動小数点演算による桁落ちと丸め誤差の影響を明らかにした．また，カット生成・除去方法の検討を行い，有効と考えられる除去方法を示した．

1. はじめに

線形計画問題（以下，LP 問題と略記）のすべての変数に整数条件を加えた問題を全整数計画問題（以下，IP 問題と略記）と呼び，この問題の解法として 1958 年に R. E. Gomory によって発明された小数法がある[1]．この方法は，IP 問題に切除平面（以下，カットと呼ぶ）と呼ばれる制約式を有限個加えることで最適解を得ることを理論的に保証した最初の方法で，現在，一般に切除平面法と総称されている方法の発端となったものである[2]．Gomory の発表以来，この方法に関するさまざまな研究が行われ，理論的には，

（a） 混合整数計画問題に対するカット生成方法の提案，

（b） Gomory とは違う，小数法の効率を上げるためのカット生成方法の提案，

（c） 最適解を得るまでは実行可能解が求まらない小数法の欠点を補う，主切除平面法の提案，

などがあり，多くの発展を遂げている[3]．

一方，小数法に対する数値実験も盛んに行われ，特殊構造を持ち変数の値が 0 か 1 に限定された問題に対しては有効であるという報告がある[4]．しかし，

（a） 最適解を得るまでに加えるカット数が膨大になる場合がある，

（b） いくらカットを加えても目的関数値が変化しない，いわゆる目的関数値の貼りつきが起こる，

（c） 桁落ちと丸め誤差の影響で最適解が得られないことが頻繁に起こる，

という報告が数多くあり[5)-7)]，小数法の有効性に関しては多くの研究者が否定的な意見である．

ところが，これらの結果はあくまでも浮動小数点演算を用いた実験から得られた経験的なものであり，これらの結果を裏付けるための，桁落ちと丸め誤差を排除した場合の小数法の振舞いを明らかにした報告や，桁落ちと丸め誤差の影響を詳しく調べた報告は見当らない．

浮動小数点演算で生じる桁落ちと丸め誤差を排除する演算として有理数演算がある．この演算は，一般に浮動小数点演算に比べ多くの計算量とメモリーを必要とし，計算機の性能の制約から今まではほとんど用いられていなかったものである．しかし，最近の計算機の急速な進歩によりこの演算も次第に利用可能となりつつあり，計算機言語の中には仕様でこの演算をサポートしているものもある．

このように，有理数演算を現実に利用できる環境が整いつつある現在，今まで浮動小数点演算を用いた実験からはその有効性に疑問が持たれていた小数法を有理数演算を用いてインプリメントし，桁落ちや丸め誤差を排除した場合の小数法の振舞いを調べることは可能なことである．そして実際にインプリメントを行い，小数法に特有の性質を明らかにすることは，今後の計算機の発達による小数法の実用の可能性を探求す

† Rational Arithmetic for Gomory's Fractional Method by TOSHIO OHYANAGI and AZUMA OHUCHI (Department of Information Engineering, Faculty of Engineering, Hokkaido University).

†† 北海道大学工学部情報工学科

る上で，重要な課題であると考えられる．

本論文は，小数法を有理数演算を用いて実現し，

（1） 桁落ちと丸め誤差を排除した場合の小数法の振舞い，

（2） 浮動小数点演算を用いた場合に生じる桁落ちと丸め誤差の結果への影響，

（3） カットの生成・除去方法の違いによる小数法の効率の違い，

を調べるものである．

以下，2章で小数法の概要を説明し，3章で今回の実験で用いた小数法のコードのインプリメントについて述べる．4章では，浮動小数点演算を用いた場合に小数法の典型的な振舞いを示した2題の問題を取り上げて，有理数演算を用いた場合の小数法の振舞いと，浮動小数点演算を用いた場合の桁落ちと丸め誤差の影響を調べる実験を行い，それらの結果について検討する．5章では，有理数演算を用いた場合の IP 問題の代表的なテスト問題に対する実験とカットの生成・除去方法の違いによる小数法の効率の違いを調べる実験について述べる．

2. ゴモリーの小数法

2.1 問題の定式化と記号の定義

一般に IP 問題は，

（P） 目的関数 $z=c^t x$➡最大化．

制約条件 $Ax=b,$ （1）

$x\geqq 0,\ x\in Z^{n+m}.$

と定式化される．ここで，Z^{n+m} は $(n+m)$ 次元整数列ベクトルの集合，c と x は $(n+m)$ 次元列ベクトル，b は m 次元列ベクトル，A は $m\times(n+m)$ 次元行列，t はベクトルの転置を表す*．

また，（P）より整数制約を除いた連続緩和問題を

（$\bar{\text{P}}$） 目的関数 $z=c^t x$➡最大化．

制約条件 $Ax=b,\ x\geqq 0.$ （2）

とする．（$\bar{\text{P}}$）の最適解 $\bar{x}$ を基底 $\bar{x}^B$ と非基底 $\bar{x}^N$ に分割し，$\bar{x}^B$, $\bar{x}^N$ に対応する係数行列 A の部分行列と目的関数の係数をそれぞれ B, N, c^B, c^N とすると，基底 $\bar{x}^B$ に対する最適実行可能基底表現形式は，

（T） $z=\bar{z}^B-(\bar{c}^N)^t(-\bar{x}^N),$

$\bar{x}^B=\bar{b}+\bar{N}(-\bar{x}^N).$ （3）

ただし，$\bar{z}^B=(c^B)^t B^{-1}b,$

$(\bar{c}^N)^t=(c^N)^t-(c^B)^t B^{-1}N,$

$\bar{b}=B^{-1}b,\ \bar{N}=B^{-1}N$ である．

となる．小数法の説明では，これに $\bar{x}^N=-(-\bar{x}^N)$ という自明な関係を入れた列タブローを考える（**図 1**）．これが小数法における初期の列タブローとなる．なお，図1では基底変数と非基底変数をそれぞれ $\bar{x}_i^B$, $\bar{x}_j^N$ と表し，$\bar{c}^N$, $\bar{b}$ の成分をそれぞれ $\bar{c}_j^N$, $\bar{b}_i$, $\bar{N}$ の要素を $\bar{a}_{ij}$ としている．

以下の説明では，図1のタブローの $\bar{b}_i$ の列を第0列と呼び，非基底変数の添字集合を J とする．

2.2 小数法のアルゴリズム

小数法の基本的なアルゴリズムを以下に示す．

```
BEGIN
S 1:  (P̄)を解く;
S 2:  IF (x̄ が得られる) THEN
S 3:    初期の列タブローを求める;
        LOOP
S 4:      IF (x̄ が整数解) THEN
S 5:        x̄ が(P)の最適解である;
S 6:        LOOP を抜け終了
          ELSE
S 7:        カットを生成する;
S 8:        カットを列タブローの最下行に追加
            した新しい LP 問題を辞書式双対単
            体法[7]を使って解く
          END; (* IF *)
S 9:      IF (x̄ が得られない) THEN
S10:        (P)には最適解が存在しない;
S11:        LOOP を抜け終了
          END (* IF *)
        END (* LOOP *)
      ELSE
S12:    (P)には最適解が存在しない;
      END (* IF *)
```

	1	$-\bar{x}_1^N$	$-\bar{x}_2^N$	・・・	$-\bar{x}_n^N$
z	$\bar{z}^B$	$-\bar{c}_1^N$	$-\bar{c}_2^N$	・・・	$-\bar{c}_n^N$
$\bar{x}_1=\bar{x}_1^N$	0	−1	0	・・・	0
$\bar{x}_2=\bar{x}_2^N$	0	0	−1	・・・	0
・	・	・	・		・
・	・	・	・		・
・	・	・	・		・
$\bar{x}_n=\bar{x}_n^N$	0	0	0	・・・	−1
$\bar{x}_{n+1}=\bar{x}_1^B$	$\bar{b}_1$	$\bar{a}_{n+1,1}$	$\bar{a}_{n+1,2}$	・・・	$\bar{a}_{n+1,n}$
・	・	・	・		・
・	・	・	・		・
・	・	・	・		・
$\bar{x}_{n+m}=\bar{x}_m^B$	$\bar{b}_m$	$\bar{a}_{n+m,1}$	$\bar{a}_{n+m,2}$	・・・	$\bar{a}_{n+m,n}$

図 1 列タブロー
Fig. 1 Column tableau.

* A, b, c の成分はすべて整数で，rank $(A)=m$ と仮定している．

END; (* BEGIN *)

なお，S7 のカット生成の直前で不要になったカットを必要に応じて除去し，列タブローの拡大を抑えることができる*.

2.3 カットの生成

ゴモリーの小数法は，前節のアルゴリズム中で S7 のカットの生成に特徴がある．ゴモリーが提案したカットは，列タブローの第 0 列成分の値が非整数値となっている行（これをカット生成行と呼ぶ）を使って，

$$\sum_{j\in J} f_{ij}\bar{x}_j^N \geqq f_{i0}. \qquad (4)$$

とするものである．ただし，

$f_{i0}=\bar{b}_i-[\bar{b}_i]$ $(0<f_{i0}<1)$;

$f_{ij}=\bar{a}_{ij}-[\bar{a}_{ij}]$ $(0\leqq f_{ij}<1, j\in J)$.

で，[ ]はガウス記号である．ここでこのカットは，文献3)で基本カットと呼ばれている以下の(5)式において $h=1$ の特別な場合である．

$$\sum_{j\in J}([h]\bar{a}_{ij}-[h\bar{a}_{ij}])\bar{x}_j^N \geqq [h]\bar{b}_i-[h\bar{b}_i]. \qquad (5)$$

ただし，h は任意の実数である．

このようにカットの生成は，カット生成行の各成分の整数部の値を捨て小数部の値のみを使って行われる．これが小数法と呼ばれる由来である．アルゴリズムの S8 では，カットのスラック変数 $s\geqq 0$ を導入して(4)式を，

$$s=-f_{i0}-\sum_{j\in J} f_{ij}(-\bar{x}_j^N). \qquad (6)$$

として列タブローの最下行に加える．

カット生成行の選択としては，アルゴリズムの有限収束性を保証するために，定期的にタブローの第 0 列成分の値が非整数値となる最初の行を選択する方法がある．このほかに，有限収束性は保証されていないが，小数法の効率を改善する実用的に有効な経験則がいくつか提案されている[3].

3. 小数法のインプリメント

本章では小数法のインプリメントについて説明する．インプリメントは，PC-9801 vx (CPU : 80286, CLOCK : 8 MHz) 上で Modula-2 言語を用いて行った．

3.1 浮動小数点演算によるインプリメント

現在の浮動小数点演算のように有効数字の桁数に制限がある場合には，整数部の値が 0 でない限り，取り出した小数部の値の有効数字の桁数は，元の数の有効数字の桁数に比べて小さくなる．小数法では，このような数を使ってさらに計算を続けるので有効数字の桁数は，カットを加えるに従って減少する．また，桁落ちせずに残った値も，一般に，有限桁で実数を表現しているために生じる丸め誤差を含んだ値である．このような桁落ちや丸め誤差を含んだ値を使った計算を続けるので，インプリメントで用いる LP 問題を解くアルゴリズムや 0 判定と整数判定の方法の違いにより得られる結果が異なる可能性がある．

今回のインプリメントでは，LP 問題を解くアルゴリズムとして，2 章のアルゴリズムの S2 ではコンパクトタブローを使った 2 段階単体法，S8 では図 1 の列タブローを使った辞書式双対単体法を用いた．S3 ではコンパクトタブローから列タブローへの変換を行うが，このとき基底に人工変数が残っていれば，まずそれを強制的に基底から追い出した．なお，コンパクトタブローおよび列タブローは，メモリー上に倍精度の 2 次元配列をとってインプリメントし，計算はすべて倍精度で行った．0 判定と整数判定は，微小定数 ε_1 と ε_2 を用いて次のように行った．ある変数 v に対して，

$$|v|<\varepsilon_1. \qquad (7)$$

ただし，| |は絶対値をとる記号

が成立するときその値を 0 と判定する．また，

$$\min(v-[v], 1-(v-[v]))<\varepsilon_2. \qquad (8)$$

ただし，min は小さいほうの値を返す関数

が成立するときその値を整数と判定する．

ここで，ε_1 と ε_2 の大小関係は，

$$\varepsilon_1\leqq\varepsilon_2. \qquad (9)$$

である．なぜなら，(7)式と(8)式の左辺の値を考えると，その有効数字の桁数は，(7)式のほうが(8)式よりも小さくなることはないからである．なお，ε_1 と ε_2 の値の設定は，実行時に設定できるようにした．

3.2 有理数演算によるインプリメント

Modula-2 言語では仕様として有理数演算を用意していないので有理数演算ライブラリーを作成しなければならない．有理数は，分子と分母を別々の整数とするレコード構造を用いて表現することができるが，分子分母の整数としてコンパイラが提供する固定バイト長の整数型を用いたのでは，計算途中での分子分母の中間膨張のためにすぐに整数のオーバフローを起こし，計算を続行することが不可能となる．ゴモリーも小数法の検証として，10 変数 10 制約式程度の問題に

* 問題によっては除去したカットを後で再度生成する必要が生じることもあり，そのまま残しておくこともできる．

対して有理数演算を用いた実験を行っているが，すぐに整数のオーバフローを起こしてしまうと報告している[2]．

そこで今回のインプリメントでは，整数のオーバフローをできる限り起こさないようにするために，任意多倍長整数（以下，BIGNUM と呼ぶ）を用いることにして，BIGNUM を扱うための四則演算，比較演算，入出力の手続を作成し，それをもとに有理数を扱うライブラリーを実現した．BIGNUM のデータ構造とそのデータ構造で表される整数の例をそれぞれ**図 2** の(a)と(b)に示す．

有理数演算を用いた小数法のインプリメントは，上で述べたライブラリーを使って，浮動小数点演算の場合と同様の LP 問題を解くアルゴリズムを用いて行った．

4. 小数法の振舞いに対する検討

小数法の振舞いを調べるための実験を以下のように行った．実験では，列タブローの第 0 列成分の値が非整数値となる最初の行を生成行とした．カットは，基底行列の行列式の値を D とすると，(5)式において $h=D-1$ とした以下の式を用いた[7]．

$$\sum_{j\in J} f_{ij}{}^{C} \geqq f_{i0}{}^{C}. \tag{10}$$

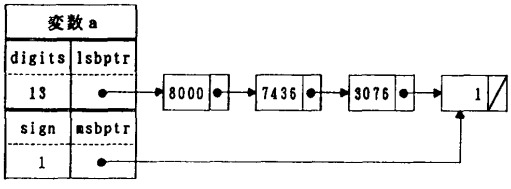
ただし，$f_{i0}{}^{C}=-1+f_{i0}$,

$$f_{ij}{}^{C}=\begin{cases}0 & (f_{ij}=0)\\ 1-f_{ij} & (f_{ij}\neq 0)\end{cases}$$ である．

```
TYPE
  CellPtr = POINTER TO Cell;        (* Cellへのポインタ *)
  Cell    = RECORD
              car : [ 0 .. 9999 ];(* 最大9999の整数   *)
              cdr : CellPtr       (* 上位のCellを指す *)
            END;
  BIGNUM  = RECORD
              lsbptr,             (* 最下位のCellを指す *)
              msbptr : CellPtr;   (* 最上位のCellを指す *)
              digits : CARDINAL;  (* 桁数 *)
              sign : [ -1 .. 1 ]  (* 符号：-1が負,1が正 *)
            END;
```

(a) BIGNUM型の定義

```
a : = 1307674368000; (* 15! = 1307674368000 *)
```



(b) BIGNUMの例

図 2 BIGNUM 型の定義と BIGNUM の例
Fig. 2 Definition of BIGNUM type and an example of BIGNUM.

このカットは，文献 7) で図 1 の列タブローを用いた場合の小数法の有限収束性の証明で用いられているものである．カットの除去は，アルゴリズムの S7 の直前に除去可能性を調べ，可能な場合にそれを除去した．

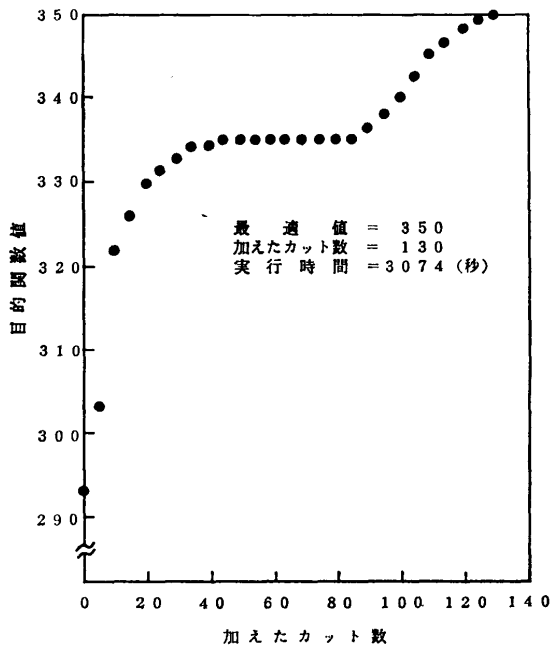


図 3 問題 P1 に対する加えたカット数と目的関数値
Fig. 3 Number of added cutting planes and the objective function values for problem P1.

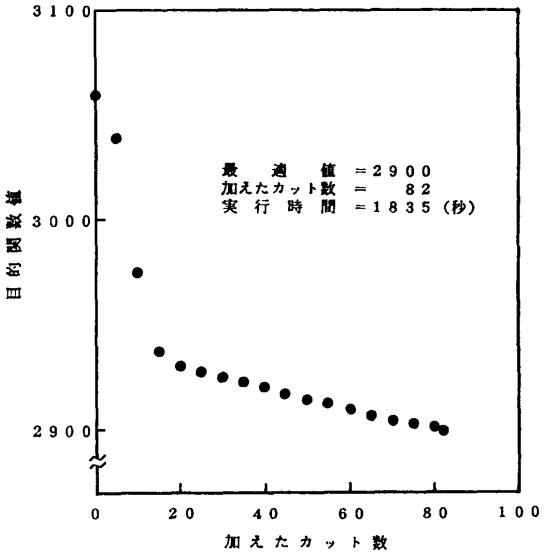


図 4 問題 P2 に対する加えたカット数と目的関数値
Fig. 4 Number of added cutting planes and the objective function values for problem P2.

このようなカット生成・除去方法を用いて行った実験の中から，以下では浮動小数点演算を用いた場合に小数法の典型的な振舞いを示した文献 5) のシステム設計問題 (P1) と文献 8) の予算問題 (P2) の2つの問題を取り上げる．P1は文献5) で浮動小数点演算による小数法では解の得られなかった例としてあげられている問題である．また，P2は文献8) の中で小数法ではなく部分列挙法を用いて解いているが，文献 8) に示されている最適解が誤っている問題である．

以下，4.1～4.3 節では実験とそれらの結果を示し，4.4 節でそれらの実験結果に対する考察を行う．

4.1 有理数演算を用いた場合の実験と結果

有理数演算を用いて桁落ちと丸め誤差を排除し，理論どおりの小数法の振舞いを調べるために，カットを加えるごとの目的関数値の変化に着目し，その変化を調べる実験を行った．P1 と P2 に対する実験結果をそれぞれ**図3**と**図4**に示す．両図においてはカットを5面加えるごとの目的関数値を示している．この結果，有理数演算を用いた場合には，P1は130面，P2は 82 面のカットを加えることで最適解を得ることができており，そのときの目的関数値はそれぞれ 350 と 2,900 である．なお，図3ではカットを加えていったとき目的関数値が増加傾向で，図4では減少傾向となっているが，その理由は，P1 が最小化問題で P2 が最大化問題であるためである．このように，浮動小数点演算では解の得られなかった問題でも，有理数演算では解が得られ，最適解を得るまでの目的関数値の変化の過程を正確に調べることが可能である．

4.2 浮動小数点演算を用いた場合の実験と結果

浮動小数点演算を用いた場合の小数法の振舞いを調べるために，0判定と整数判定で用いる ε_1 と ε_2 の設定を変え，得られる結果の違いを調べる実験を行った．P1 と P2 に対する実験結果をそれぞれ**表1**と**表2**に示す．表中の値は，左側が最終的な結果を得るまでに加えたカット数で，カッコ内は有理数演算による場合と目的関数値がほぼ同じ（差の絶対値が ε_1 以下）

表1 問題 P1 に対する浮動小数点演算による実験結果
Table 1 Computational results of problem P1 in floating-point arithmetic.

ε_2	10^{-3}		10^{-4}		10^{-5}		10^{-6}	
ε_1	カット数	z	カット数	z	カット数	z	カット数	z
10^{-3}	40 (7)	×	—	—	—	—	—	—
10^{-4}	500 (7)	386	51 (2)	×	—	—	—	—
10^{-5}	38 (7)	×	45 (2)	×	500 (2)	323	—	—
10^{-6}	38 (7)	×	9 (2)	×	500 (2)	320	500 (2)	320
10^{-7}	38 (7)	×	9 (2)	×	500 (2)	317	500 (2)	317
10^{-8}	38 (7)	×	9 (2)	×	500 (2)	321	500 (2)	319
10^{-9}	38 (7)	×	9 (2)	×	500 (2)	416	500 (2)	320
10^{-10}	38 (7)	×	9 (2)	×	500 (2)	323	500 (2)	318
10^{-11}	38 (7)	×	9 (2)	×	500 (2)	320	500 (2)	319
10^{-12}	500 (7)	334	9 (2)	×	500 (2)	319	500 (2)	317

表2 問題 P2 に対する浮動小数点演算による実験結果
Table 2 Computational results of problem P2 in floating-point arithmetic.

ε_2	10^{-3}		10^{-4}		10^{-5}		10^{-6}	
ε_1	カット数	z	カット数	z	カット数	z	カット数	z
10^{-3}	108 (81)	×	—	—	—	—	—	—
10^{-4}	109 (81)	×	109 (81)	×	—	—	—	—
10^{-5}	108 (81)	×	130 (81)	×	124 (81)	×	—	—
10^{-6}	123 (81)	×	157 (81)	×	169 (81)	×	136 (8)	×
10^{-7}	132 (81)	×	145 (81)	×	154 (81)	×	136 (8)	×
10^{-8}	104 (81)	2832	104 (81)	2832	105 (81)	2832	72 (8)	×
10^{-9}	104 (81)	2832	104 (81)	2832	181 (81)	×	76 (8)	×
10^{-10}	104 (81)	2832	104 (81)	2832	223 (81)	×	76 (8)	×
10^{-11}	104 (81)	2832	104 (81)	2832	111 (81)	×	76 (8)	×
10^{-12}	104 (81)	2832	104 (81)	2832	136 (81)	×	58 (8)	×

である間に加えたカット数を表す．ただし，カット数の下の下線は，解を得ることができずそのカット数で小数法の実行を打ち切ったことを示す．右側は，そのときの目的関数値で，×印になっている場合は実行可能解が存在しない状態となって小数法を終了したことを表す．また，表中で中線を引いている部分は，ε_1 と ε_2 の設定が(9)式の条件を満足しないものである．

4.3 桁落ちと丸め誤差の影響を調べる実験と結果

浮動小数点演算を用いた場合，桁落ちと丸め誤差が大きく影響すると言われている．そこで，その影響を調べる実験を行った．実験結果を**表3**に示す．実験は，P1 に対して $\varepsilon_1=10^{-6}$, $\varepsilon_2=10^{-3}$ とし，8面目のカットを生成する際に用いる生成行の各成分と f_{ij}^C の値を調べたものである．8面目のカットを選んだ理由は，表1より，この ε_1 と ε_2 の組合せのとき，7面目のカットを加えた後までは有理数演算による場合と目的関数値がほぼ同じであるが，それ以降は異なることがわかり，8面目のカットが有理数演算と浮動小数点演算による場合では異なっている可能性が大きいと考えたからである．

4.4 実験結果に対する考察

有理数演算を用いた場合の実験結果より，

(a) 最適解を得るまでに多くのカットを加える必要がある，

(b) 目的関数値がある値に長い間貼りついて変化しない場合が起こる，

(c) 目的関数値が非常に長い間ある値に貼りついても最終的にはその状態を脱し最適解を得ることができる，

(d) 目的関数値が貼りつく場合とほとんど貼りつかない場合があり，問題によって目的関数値の変化の様子が異なる，

ということが明らかとなった．

(a)と(b)の結果は，1章で述べたように，過去の多くの研究者による浮動小数点演算を用いた実験から経験的に言われていたことであるが，有理数演算を行っても同様の結果が得られたことから，小数法の実現方法には依存しない小数法に特有の振舞いと考えられる．(c)と(d)の結果は，今回の有理数演算を用いた実験で顕著になった現象である．(c)については，小数法の理論でこのような現象が起こることは予想されているが，浮動小数点演算ではこれほど長い間貼りつきが起こった後に最適解に収束した例は，著者の知る限りこれまで観測されていない．また(d)についても，

表3 有理数演算と浮動小数点演算によるカット生成行の値
Table 3 Values of the source row in rational arithmetic and floating-point arithmetic.

列番号	有理数演算		浮動小数点演算	
	値	f_{ij}^C	値	f_{ij}^C
0	928/3	2/3	309.33814	0.66186
1	−25	0	−25.00000	0.00000
2	−1	0	−1.00000	0.00000
3	−14	0	−14.00000	0.00000
4	−25	0	−25.00000	0.00000
5	−17/3	2/3	−5.66710	0.66710
6	−16	0	−15.99869	0.99869
7	−5/3	2/3	−1.66710	0.66710
8	−5	0	−5.00000	0.00000
9	−2	0	−2.00000	0.00000
10	−2	0	−2.00000	0.00000
11	−2	0	−2.00000	0.00000
12	−7/3	1/3	−2.33355	0.33355
13	−10	0	−10.00000	0.00000
14	0	0	0.00000	0.00000

浮動小数点演算では正確に調べることが困難で，これまで明らかにされなかった現象である．これらの結果は有理数演算を行うことの有効性を示すものである．

浮動小数点演算を用いた場合の実験結果より，問題P1 に対しては，

(e) 実行可能解が存在しない，

(f) 計算を打切り，

という結果を得て計算を終了しており，P2 に対しては，P1 に対する(e)と，

(g) 最適解以外の整数解を得た，

という結果を得て計算を終了していることがわかる．このように，有理数演算では最適解が得られた問題でも，浮動小数点演算では得られないことがある．そしてこのように得られる結果に違いが生じることから，浮動小数点演算を用いた場合には，小数法から得られる結果の信頼性はあまり高くないこととなり，IP 問題の解法として小数法を用いることは，従来言われているようにあまり有効ではないと考えられる．

このように得られる結果に違いが生じた原因は，桁落ちと丸め誤差の影響にあると考えられ，それを実際に調べた結果が表3である．この結果，f_{ij}^C が0以外の場合には有効数字が高々3桁しかなく，あまり多くのカットを加えていない場合でさえも，桁落ちと丸め誤差の影響が大きいことがよくわかる．そして，この影響で生成されるカットが有理数演算の場合とは異な

ることになり，そのために(e)から(g)のような結果が生じたと考えられる．この結果は，厳密な計算が行える有理数演算を用いることにより，明らかにすることが可能となったものである．

5. 有理数演算を用いた小数法に対する実験

5.1 代表的なテスト問題に対する実験結果

前章では，浮動小数点演算を用いた場合に小数法の典型的な振舞いを示した2題の問題を取り上げて，有理数演算を用いた場合の小数法の振舞いについて述べた．

本節では，今回行った実験の中で代表的なテスト問題として知られている問題に対する実験結果について述べる．それらの問題は，9題の配置問題 (Alloc 1～Alloc 9)，10 題の Haldi の固定費用問題 (Haldi 1～Haldi 10) そして5題の Haldi の IBM テスト問題 (IBM 1～IBM 5) の計 24 題である[9]．これらの問題は，サイズはそれほど大きくないが，解くことが難しい問題として知られている．

表 4 にそれらの問題に対して文献9)で示されている5つのコード (IPM 3～IPSC) による結果とわれわれのコード (RGFM) による結果を合せて示す*．

この結果，すべての問題に対して最適解の得られたコードは LIP 1 と RGFM の2つであることがわかる．

表 5 に RGFM による実験の詳細な結果を示す．なお，実行時間の単位は秒である．この結果，前章で得られた結果に加え，問題によってはカットを多く加えなくても最適解を得る場合があることが確認された．しかし，Haldi 10 に対する結果のようにカットを5千面以上加えなければ最適解を得ることができない場合もあった．このことから，最適解を得るまでに加えなければならないカットの数が問題によってかなり違

表 4 6種類のコードによる実験結果 (○と×で最適解が得られたか否かをそれぞれ示す)
Table 4 Computational results by six different program codes. (○ and × indicate whether the optimal solution is obtained or not.)

問題	コード					
	IPM 3	LIP 1	LIP 2-1	LIP 2-2	IPSC	RGFM
Alloc 1	○	○	○	○	○	○
Alloc 2	○	○	○	○	○	○
Alloc 3	○	○	○	○	○	○
Alloc 4	○	○	○	○	○	○
Alloc 5	○	○	○	○	○	○
Alloc 6	○	○	○	○	○	○
Alloc 7	○	○	○	○	○	○
Alloc 8	○	○	○	○	○	○
Alloc 9	○	○	○	○	○	○
Haldi 1	○	○	○	○	○	○
Haldi 2	○	○	○	○	○	○
Haldi 3	○	○	○	○	○	○
Haldi 4	○	○	○	○	○	○
Haldi 5	×	○	×	×	○	○
Haldi 6	×	○	×	○	○	○
Haldi 7	×	○	×	×	×	○
Haldi 8	×	○	×	○	×	○
Haldi 9	○	○	×	○	○	○
Haldi 10	○	○	×	×	×	○
IBM 1	○	○	○	○	○	○
IBM 2	○	○	○	○	○	○
IBM 3	○	○	○	○	○	○
IBM 4	○	○	○	○	○	○
IBM 5	○	○	×	○	○	○

表 5 代表的なテスト問題に対する実験結果
Table 5 Computational results of typical test problems.

問題名	サイズ ($m \times n$)	カット数	実行時間 (s)	最適値
Alloc 1	11×10	1	5	50
Alloc 2	11×10	8	21	52
Alloc 3	11×10	6	18	57
Alloc 4	11×10	2	11	62
Alloc 5	11×10	0	5	67
Alloc 6	11×10	11	70	68
Alloc 7	11×10	48	317	70
Alloc 8	11×10	15	108	75
Alloc 9	11×10	0	2	85
Haldi 1	6× 5	8	15	7
Haldi 2	6× 5	11	20	8
Haldi 3	6× 5	22	46	10
Haldi 4	6× 5	3	8	8
Haldi 5	6× 5	300	504	76
Haldi 6	6× 5	86	171	106
Haldi 7	4× 5	0	1	110
Haldi 8	4× 5	0	1	140
Haldi 9	9× 6	23	63	9
Haldi 10	16×12	5, 361	190, 001	17
IBM 1	7× 7	0	1	8
IBM 2	7× 7	2	3	7
IBM 3	3× 7	5	11	187
IBM 4	15×15	9	38	10
IBM 5	15×15	1, 650	12, 300	15

* IPM 3 と LIP 1 は小数法，LIP 2-1, LIP 2-2, IPSC は全整数法のコードである[9]．

い，桁落ちや丸め誤差を排除しても，実質的に解を得られない場合も起こりうることが予想される．

5.2 カット生成行の選択方法とカット除去規則に関する実験

最適解を得るまでに加えるカット数は，カット生成行の選択とカットの除去規則に影響を受けることが従来の浮動小数点演算による実験より経験的に知られている．

そこで，このことを調べる実験を行った．実験で用いたカット生成行の選択は，

Ⅰ：非整数値をとる最初の行，

Ⅱ：f_{i0}^{C} が最大の行，

の2つである．また，カットの除去規則は，

a：カットを除去可能なときに除去，

b：列タブローの行数が $m+2n+1$ に達したときに除去，

の2つである．ここで，カットの除去規則のaを用いた場合は，同時に考慮しなければならないカットの最大数が n であり，列タブローの行数が $m+2n+1$ を越えることがないことが理論的に示されている[7]．このことを考慮し，さらに生成したカットをできる限り残すようにしたのがカット除去規則のbである．

実験では，4章の問題P1，P2と前節の問題の中で最適解を得るためにカットの生成が必要であった問題に対して，上で述べたカット生成行の選択とカット除去規則のすべての組合せ（4通り）を考え，最適解を得るまでに加えたカット数を調べた．なお，組合せの中で（Ⅰ，a）は，4章と前節の実験で用いたものである．

実験結果を**表6**に示す．表中の値は最適解を得るまでに加えたカット数で，カッコ内は実行時間（単位は秒）である．なお，表中で値の示されていない部分は，今回の実験では，メモリーや実行時間の制約から最適解を得ることができなかった場合を表す*．

* メモリーの制約とは，有理数を扱うために利用できるヒープ領域のサイズのことで，今回のインプリメントでは約400 KBである．実行時間の制約とは，計算を打ち切る時間のことで，今回は（Ⅰ，a）の組合せの約2倍の時間とした．

表6 生成行選択方法と除去規則に関する実験結果
Table 6 Computational results for source row selection and cut removal rule.

選択除去	(Ⅰ, a)	(Ⅰ, b)	(Ⅱ, a)	(Ⅱ, b)
問題	カット (時間)	カット (時間)	カット (時間)	カット (時間)
P1	130 (3,074)	154 (7,930)	—	—
P2	82 (1,835)	12 (178)	2 (58)	2 (57)
Alloc 1	1 (5)	1 (6)	2 (7)	2 (8)
Alloc 2	8 (21)	8 (22)	12 (45)	12 (62)
Alloc 3	6 (18)	6 (19)	4 (19)	4 (19)
Alloc 4	2 (11)	2 (11)	2 (11)	2 (11)
Alloc 6	11 (70)	11 (81)	9 (50)	9 (65)
Alloc 7	48 (317)	32 (392)	49 (355)	35 (617)
Alloc 8	15 (108)	14 (152)	9 (39)	16 (126)
Haldi 1	8 (15)	8 (18)	9 (19)	9 (24)
Haldi 2	11 (20)	11 (25)	11 (37)	11 (49)
Haldi 3	22 (46)	20 (65)	6 (13)	5 (16)
Haldi 4	3 (8)	3 (7)	3 (10)	3 (11)
Haldi 5	300 (504)	301 (740)	147 (425)	132 (574)
Haldi 6	86 (171)	86 (202)	88 (203)	105 (439)
Haldi 9	23 (63)	22 (88)	20 (63)	19 (90)
Haldi 10	5,361 (190,001)	5,238 (202,149)	—	—
IBM 2	2 (3)	2 (2)	2 (3)	2 (3)
IBM 3	5 (11)	5 (12)	18 (34)	19 (74)
IBM 4	9 (38)	9 (38)	12 (57)	12 (63)
IBM 5	1,650 (12,300)	1,853 (30,911)	—	—

この実験結果から，

（a） アルゴリズムの有限収束性を保証している生成行選択Ⅰは，カットの除去規則によらず，すべての問題に対して最適解が得られている（(Ⅰ, a)，(Ⅰ, b)），

（b） 生成行選択のⅡは，カットの除去規則によらず，問題P2とHaldi 5に対して特に有効であるが，最適解の得られない問題が3題ある（(Ⅱ, a)，(Ⅱ, b)），

（c） カットを除去しないで残すことの効果は，生成行選択がⅠのとき問題P2に対しては顕著に現れているが，他の場合にはそれほど大きく現れていない（(Ⅰ, a)と(Ⅰ, b)，(Ⅱ, a)と(Ⅱ, b)），

（d） カットを残して加えるカット数が少なくなっても，実行時間が長くなる場合がある（(Ⅱ, a)と(Ⅱ, b)の問題 Alloc 7），

ことがわかる．

（a）のカット除去規則によらず最適解を得ることは，カット生成行選択がⅠのときは理論的に保証されていることで，今回の実験でもそのような結果となっ

ている．(b)については，カット生成行選択のIIは浮動小数点演算を用いた従来の実験より経験的に有効な方法であると言われていたが，この結果からはその有効性をはっきりと結論付けることは困難である．これは，今後の検討課題である．

(c)と(d)の結果より，カットの除去については，除去しないで残しておくことが著しく有効に働くこともあるが一般にはそうとも言えない．したがって，タブローの拡大を抑えるように，除去可能なときはいつでも除去するほうが良いと考えられる．

6. おわりに

本論文では，小数法を有理数演算を用いて実現し，

(a) 最適解を得るまでに多くのカットを加える必要があることと目的関数値の貼りつきが起こる場合があることは，小数法に特有の振舞いである，

(b) 浮動小数点演算では解けなかった問題が，有理数演算を用いると，小数法の理論が保証するようにカットを加え続けると解くことができた，

(c) 浮動小数点演算を用いた場合に生じる桁落ちや丸め誤差の影響は，カットをそれほど多く加えなくても大きくなることがある，

(d) 加えるカット数が膨大になる場合があり，実質的には解けない問題がありうる，

(e) カットを除去しないで残しておくことの効果はあまり大きいとは言えず，除去可能なときはいつでも除去したほうが良いと考えられる，

ことを実験により明らかにした．

以上の結果は，有理数演算を用いて桁落ちや丸め誤差を排除して小数法を実現し，正確な実験を行うことにより得ることができた結果で，有理数演算の有効性を示すものである．

今後の課題としては，

(1) (5)式の基本カットをもとにしたカットの生成の検討，

(2) カット生成行選択のIIに対する検討，

(3) 本論文で用いたカット生成行選択とはまったく別の選択方法[10]の検討，

(4) IP 問題に対する小数法の実用性に関する検討，

等を行うことが挙げられる．

謝辞 本研究を行うにあたり，多くのご指導・ご助言をいただいた北海道工業大学電気工学科加地郁夫教授に深謝いたします．

参考文献

1) Gomory, R. E.: Outline of an Algorithm for Integer Solutions to Linear Programs, *Bulletin of American Mathematical Society*, Vol. 64, pp. 275-278 (1958).
2) Gomory, R. E.: An Algorithm for Integer Solutions to Linear Programs, in *Recent Advances in Mathematical Programming* (Graves, R. L. and Wolfe, P. eds.), pp. 269-302, McGraw-Hill, New York (1963).
3) Garfinkel, R. S. and Nemhauser, G. L.: *Integer Programming*, John Wiley & Sons, New York (1972).
4) Martin, G. T.: An Accelerated Euclidean Algorithm for Integer Linear Programming, in *Recent Advances in Mathematical Programming* (Graves, R. L. and Wolfe, P. eds.), pp. 311-318, McGraw-Hill, New York (1963).
5) Plane, D. R. and McMillan, C. Jr.: *Discrete Optimization*, Prentice-Hall, New Jersey (1971).
6) 今野：整数計画法，pp. 58-84, 産業図書，東京 (1981).
7) 今野，鈴木：整数計画法と組合せ最適化，pp. 107-119, 日科技連，東京 (1982).
8) Mao, J. C. T. and Wallingford, B. A.: An Extension of LAWLER and BELL's Method of Discrete Optimization with Examples from Capital Budgeting, *Manage. Sci.*, Vol. 15, No. 2, pp. B 51-B 60 (1968).
9) Trauth, C. A. Jr. and Woolsey, R. E.: Integer Linear Programming: A Study in Computational Efficiency, *Manage. Sci.*, Vol. 15, No. 9, pp. 481-493 (1969).
10) Srinivasan, A. V.: An Investigation of Some Computational Aspects of Integer Programming, *J. ACM*, Vol. 12, No. 4, pp. 525-535 (1965).

(平成元年6月21日受付)
(平成2年2月13日採録)

大柳 俊夫（正会員）

昭和37年生．昭和60年北海道大学工学部電気工学科卒業．同年，北海道大学工学部電気工学科助手，昭和62年同情報工学科助手，現在に至る．システム工学，ソフトウェア工学の研究に従事．昭和62年度日本オペレーションズ・リサーチ学会事例研究奨励賞（ソフトウェア部門）受賞．電気学会，日本オペレーションズ・リサーチ学会各会員．

大内　東（正会員）

昭和20年8月19日生．昭和49年北海道大学工学部大学院工学研究科博士課程修了．工学博士．北海道大学工学部情報工学科教授．システム情報工学，応用人工知能システム，医療システムの研究に従事．人工知能学会，電気学会，電子情報通信学会，計測自動制御学会，日本 OR 学会，医療情報学会，病院管理学会，IEEE-SMC 各会員．