ナンバープレイス解法アルゴリズムとFortranによる実装

伊　藤　　誠

Abstract

Number Place is a puzzle which buries the 9 × 9 squares by the number of 1-9 according to the rules. The solution of Number Place is explained by various homepages and books. It seems that however, what explained the algorithm of solution concretely is not found. Then, in this paper, the algorithm of Number Place solution devised uniquely is introduced. Furthermore, the program which actually created the algorithm using Fortran which is one of the programming languages is also introduced. The logical type array of 9 × 9 × 9 of Fortran (it is called a logical cube in this paper) is used for expressing the 9 × 9 squares of Number Place. When this 3-dimensional logical type array was used together with the functions for array offered after Fortran90, it turns out that the processing which had to use many do loops in Fortran77 becomes unnecessary, and we can program algorithm of Number Place solution very simply

1　はじめに

プログラミング言語を学ぶ上で最も重要なことは，そのプログラミング言語の文法を習得することである。しかしよく言われることであるが，文法を学んだだけでは様々な問題を解決することはできない。ある問題を解決するためには，その問題を解決するための効率的なアルゴリズムを考え，何らかのプログラミング言語を用いてそれを実装する必要がある。本稿では最近外国でも人気がでてきたナンバープレイス[1)]と言われるパズルを解くアルゴリズムを紹介し，実際にプログラミング言語の一つであるFortranを使って実装する。

まずナンバープレイスとは，９×９のます目をルールにしたがって１～９の数字で埋めていくパズルである。ナンバープレイスの解法についてはいろいろなホームページや書籍で解説されている。しかしその解法アルゴリズムを具体的に解説したものは見当たらない

1）パズル出版会社「ニコリ」がナンバープレイスを「数独（数字は独りが好き）」と名付け，その名前で外国に紹介されたため，外国では「Sudoku」と呼ばれている。ただしこのパズルの発祥の地はアメリカである。

ようである。そこで本稿では今回独自に考案したナンバープレイス解法アルゴリズムを紹介する。さらにそのアルゴリズムをプログラミング言語の一つであるFortranを用いて実際に実装したプログラムも紹介する。ナンバープレイスの９×９のます目を表すのに，Fortranの９×９×９の**論理型配列**を用いる。Fortran90以降に提供された配列用関数と合せてこの３次元の論理型配列を用いると，Fortran77では多数の do ループを回さなければならなかった処理が不要となり，ナンバープレイス解法アルゴリズムが，実にシンプルに実装できることが分かった。配列用関数は論理型配列単独，あるいは論理型配列と組み合わせて整数型・実数型等の配列を操作することができる。論理型配列は要素の値としてＴ（真）あるいはＦ（偽）の２種類の値をとるが，値がＴである論理型配列の要素，あるいは値がＴである要素に対応する整数型・実数型等の配列の要素に対してまとめて様々な操作をすることができる。

このためナンバープレイスを９×９×９の論理型配列を用いることにより，いろいろな解法を単純明快にプログラムすることが可能となった。これらの関数の使用方法も本文内で紹介する。

２　ナンバープレイスのルール

ナンバープレイスは図１のように，ヒントとしてあらかじめ記入されている数字を手がかりに，残りのます目を以下のルールにしたがって１～９の数字で埋めていくパズルである。

	8			7			3	
7			2		3			8
		9		6		4		
		4					2	
8		2				9		1
	3						6	
		6		8		1		
2			9			5		6
	7			3			4	

図１：ナンバープレイスの例

１．太線で囲まれた９個の３×３の領域にはそれぞれ１～９の数字が一つづつ入る。

２．横方向の９個の行にはそれぞれ１～９の数字が一つづつ入る。

３．縦方向の９個の列にはそれぞれ１～９の数字が一つづつ入る。

ナンバープレイスのルールは非常に単純で分かりやすいが，実際にパズルを解くためには以下で紹介する基本的な解法や高度な解法を駆使しなければならない。

３　ナンバープレイスの用語と表現方法

解法アルゴリズム等を解説するに当り，ナンバープレイスの用語と表現方法を決めてお

くことにする。ただしここで用いる用語は必ずしも一般的なものでないことをあらかじめ断っておく。

3.1 用語

1. 9 × 9 = 81個のます目の一つ一つを**セル**と呼ぶ。
2. 太線で囲まれた 3 × 3 = 9 個のセルの集まりを**領域**と呼ぶ。
3. 横方向の 9 個のセルの並びを**行**と呼ぶ。
4. 縦方向の 9 個のセルの並び**列**と呼ぶ。

これ以外にも本文中で使用する用語があるが，それらについては必要に応じて説明する。

3.2 プログラム言語での表現方法

単純に考えると 9 × 9 のセルをプログラミング言語で表現するには 9 × 9 の 2 次元配列を用意すればよいように思われる。しかし本稿で説明する解法では，「各セルに**入る可能性のない数字**を順次除去していき，ルールにしたがってそのセルに入る数字を確定する」という手続きを繰り返すことでナンバープレイスを解いていく。したがって各セルには最初は 1 ～ 9 の数字が入る可能性があるため，各セルにも 9 個の配列要素（1 ～ 9 の数字に相当）を用意し，そのセルに入る可能性がなくなった数字の要素にフラグを立てるとよい。つまり実際には 9 × 9 × 9 の 3 次元配列を用意するのである。またFortran90/95の配列用の関数を利用するにはこの配列を論理型で宣言するとよい。そしてまだ入る可能性のある数字には `.true.`(以後 `T` で表す) を，入る可能性のない数字には`.false.`(以後 `F` で表す) を代入することにする。すなわち

```
1:      logical cube(9,9,9)
2:      cube = .true.
```

のように宣言し初期化しておく。そして例えば `(2,3)` の位置のセルに数字 1 が入る可能性がなくなった場合

```
1:      cube(2,3,1) = .false.
```

とするのである。今後の説明のためここで宣言した配列 `cube` を**論理立方体**と呼ぶことにする。

なお上で示したプログラムの左端の数字は説明用のもので，実際には入力しない。Fortranのプログラムの形式としてはFortran77の固定形式を踏襲する。ところで最初のプログラムコード内の `2:`では配列`cube`の全ての要素に `T` を代入している。Fortran77では 3 重ループを回さなければならなかったが，Fortran90/95ではこのようにシンプルに記述

できるのである。

論理型の配列には数字を記憶することができないので，確定した数字を記憶させるために論理立方体とは別に，整数型の 9 × 9 の 2 次元配列を用意する。

```
1:      integer array(9,9)
```

3.3　論理立方体の配列要素と領域の参照方法

前小節で宣言した論理立方体の配列要素をプログラムで参照する方法(添字の変数名)を決めておくことにする。まず行を表す第一添字を l = 1 ～ 9 とし，上から順に 1 行目，2 行目,,, l行目とする。列を表す第二添字は m = 1 ～ 9 とし，左から順に 1 列目，2 列目,,, m 列目とする。すなわちセルの座標を (l,m) で表す。これを**大域座標**と呼ぶことにする。また数字を表す第三添字は num = 1 ～ 9 とし，論理立方体の高い方から順に数字 1，2,,, num に対応するとする。この第三添字の方向を，行・列に対応して**深さ方向**と呼ぶことにする。

ナンバープレイスではルールのところで述べたように，行・列に加え 3 × 3 の領域を調べる必要がある。したがって各領域内の配列要素の参照方法も決めておくことにする。まず領域内の行を i= 1 ～ 3 で，列を j= 1 ～ 3 で表す。すなわち領域内のセルの座標を (i,j)で表す。これを大域座標と区別するために**領域内座標**と呼ぶことにする。また 9 個ある領域は上段左から下段右に順に k= 1 ～ 9 で表すことにする。数字を表すのは上と同様でよい。

(i,j,k) の組で k 番目の領域内のセル (i,j) を特定することができるが，この組で論理立方体の配列要素の座標を直接参照できるわけではない。あくまでもセルの位置を直接参照できるのは大域座標 (l,m) であるので，(i,j,k) と (l,m) を相互に変換する必要がある。この変換は

```
   (i,j,k) → (l,m)
1:      l = (k-1)/3*3 + i
2:      m = mod(k-1,3)*3 + j
```

であり，逆変換は

```
   (l,m) → (i,j,k)
1:      i = mod(l-1,3) + 1
2:      j = mod(m-1,3) + 1
3:      k = (l-1)/3*3 + (m-1)/3 + 1
```

となる。なお上記のプログラムで 3 で割ってから 3 を掛けるという一見無意味なことをし

ているように見えるが，整数を整数で割った場合小数点以下が切り捨てられるので，元の整数を越えない最大の3の倍数となる。

行と列を調べるには do ループを l と m に対して1～9まで回せばよいが，今のままでは領域内を調べるには i と j の2重ループになってしまう。そこで領域内のセルを一つの変数 n で参照できると便利である。領域の番号のつけ方と同様に左上から右下に番号 n をつける。 n から (i,j) への変換は

```
      n → (i,j)
1:        i = (n-1)/3 + 1
2:        j = mod(n-1,3) + 1
```

となる。

領域は上で説明したように，k で参照することができるが，「高度な解法」のところで説明する「2つの領域にまたがった行と列」では，行方向の3つの領域と，列方向の3つの領域をそれぞれ参照する必要がある。この場合領域番号 k で領域を参照するよりも領域の座標 (u,v) で参照したほうが理解しやすい。そこで (u,v) を k に変換するルーチンを用意しておく。その変換は

```
      (u,v) → k
1:        k = (u-1)*3 + v
```

である。

3.4 紙面上での表現方法

以下の節で解法を説明するために論理立方体を紙面上で表現しなければならない。しかし3次元の論理立方体を紙面上に直接描くことはできないので，図2のように各セルに入る可能性が残っている深さ方向の要素（数字）をそのセル内に全て書き出すことによって表現することにする。逆に入る可能性のなくなった要素（数字）は紙面上から除去することにする。

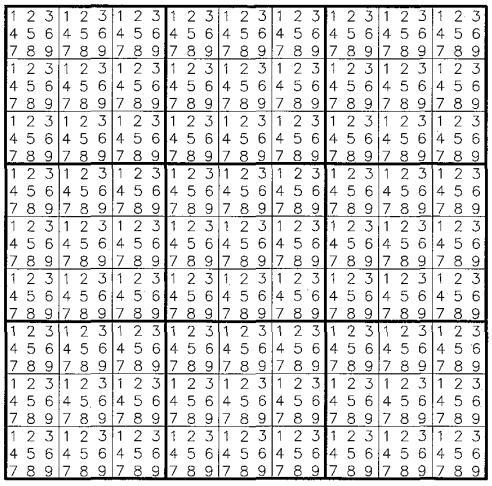

図2：論理立方体の紙面上での表現

4　基本的な解法

4.1　数字の除去と確定

図２はまだ一つも数字が決定していない状態の論理立方体である。例えば大域座標(2,3)のセルに５が入ると確定したとすると，図３に示すように太線で囲まれた１番目の領域，２行目，３列目にはルールからもはや５が入ることはないので論理立方体から除去することができる。なお確定した数字は大きな太文字で表すことにする。一般に座標(l,m)のセルに数字numが確定すると，その数字が入る可能性のなくなった領域，行，列内のセルから数字を除去する。処理するプログラムは，

```
      (remove and substitute number)
1:        (l,m) → (i,j,k)
2:        (1,1,k) → (l1,m1)
3:        cube(l1:l1+2,m1:m1+2,num)= .false.
4:        cube(l,:,num) = .false.
5:        cube(:,m,num) = .false.
6:        cube(l,m,:)   = .false.
7:        array(l,m) = num
```

となる。1:は(l,m)から(i,j,k)への変換ルーチンを呼び出している。ここでは(l,m)のセルが含まれる領域の番号kを求めることが目的である。2:ではk番目の領域の左上隅のセルの領域内座標(1,1)から，その大域座標(l1,m1)を求めている。これによってk番目の領域内の全てのセルを表す部分配列cube(l1:l1+2,m1:m1+2,num)に3:で一度にFを代入することができ，doループを回すことなく数字numを除去することができる。次に4:で1行目から，5:でm列目からnumを除去している。注意しなければならないことは，数字が確定した(l,m)のセルの深さ方向の他の数字はもはやそのセルに入ることができないので，(l,m)のセル内の他の数字も除去する必要がある。したがって6:のcube(l,m,:) = .false.とするのを忘れてはいけない。最後に確定した数字は7:で別の配列array(l,m)

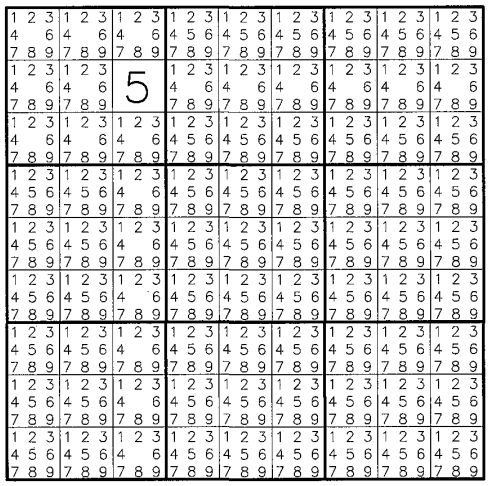

図３：(2,3) に５が入った場合の論理立方体

に代入して記憶しておく。ここで紹介するプログラムでは，数字が確定したセルの論理立方体の深さ方向の要素には全てFが代入されることを注意しておく[2)]。つまりこのプログラムでは，論理立方体の要素が全てFで埋め尽くされれば解答に到達したことになる。

4.2 領域内で数字を確定する

図1を例に基本的な解法を説明する。図4が初期状態の論理立方体である。まず領域内で数字を確定する方法を考えてみよう。図4の左上の1番目の領域を見ると，○で囲んだ3が領域内に1つしかないことが分かる。ルール1によって大域座標(3,1)には3が入ることが確定する。ところが左下の7番目の領域を見ると，○で囲んだ3が領域内に2つあることが分かる。したがってこの領域内では3を確定することはできない。

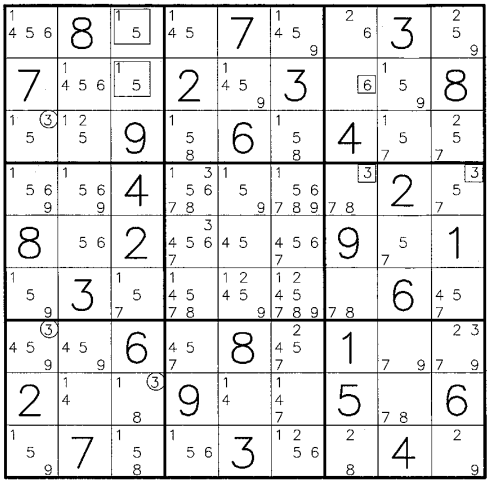

図4：図1の論理立方体の初期状態

4.3 行内，列内で数字を確定する

前小節で7番目の領域内では3を確定することができなかった。しかし図4の8行目を見てみよう。8行目内には○で囲んだ3が1つしかないことが分かる。これでルール2により大域座標(8,3)には3が入ることが確定する。同様のことは列に関しても成り立つ。

4.4 深さ方向で数字を確定する

次に図4の大域座標(2,7)内の□で囲んだ6に着目してみよう。このセルには6しか残ってないため，他の数字が入ることはなくなり，セルに入る数字は6で確定するのであるが，上で述べた領域内，行内，列内で数字を確定する方法では確定することはできない。なぜなら図4から分かるように，この6を含む領域内，行内，列内には6がそれぞれ2度出現しているからである。人間の目には明らかであるが，アルゴリズムとして規定する必要がある。つまりルールにはないことであるが，深さ方向で，他の全ての数字が入る可能性がなければそのセルに入る数字が確定することをプログラムしておかなければならないのである。

2）数字が確定したセルには，入る可能性のある残された数字の個数が0個であるということを表している。

4.5　基本的な解法のアルゴリズム

基本的な解法のアルゴリズムは特に難しい点はない。まず数字が確定する領域，行，列，セルを見い出し，数字の除去と確定を行えばよい。領域に対しては

1．ある数字 num に対して，k 番目の領域を表す論理立方体 cube の部分配列内にある T の数が 1 であれば 2．に進み，そうでなければ次の領域に進む。（なおこの部分配列は (1,1,k) → (l1,m1) という変換で求めた領域の左上隅のセルの大域座標を使ってcube(l1:l1+2,m1:m1+2,num) で表すことができる。）
2．T が入っているセルの領域内座標 (i,j) を求め，変換ルーチンで (i,j,k) から大域座標 (l,m) に変換する。
3．k 番目の領域，l 行目，m 行目から数字 num を除去し，セル (l,m) に数字 num を確定する。（つまり (l,m,num) を引数として (remove and substitute number) ルーチンを呼び出す。）
4．全ての領域を調べ終えたら次の数字に進み，全ての数字を調べ終えたら終了し次の解法へ進む。

行に対しては，

1．ある数字 num に対して，l 行目を表す論理立方体 cube の部分配列 cube(l,:,num) 内にある T の数が 1 であれば 2．に進み，そうでなければ次の行へ進む。
2．T が入っている列番号を求め，それを m とする。
3．(l,m) を含む領域，l 行目，m 行目から数字 num を除去し，セル (l,m) に数字 num を確定する。（つまり (l,m,num) を引数として(remove and substitute number) ルーチンを呼び出す。）
4．全ての行を調べ終えたら次の数字に進み，全ての数字を調べ終えたら終了し次の解法へ進む。

という手続きをとればよい。列に対してはこの処理の行と列の役割を交換してやればよい。なお 1．の部分配列は cube(:,m,num) となる。同様に深さ方向に対しては，

1．あるセル (l,m) の深さ方向を表す論理立方体 cube の部分配列 (cube(l,m,:)) 内にある T の数が 1 であれば 2．に進み，そうでなければ次のセルに進む。
2．T が入っている添字を求め，num とする。
3．(l,m) を含む領域，l 行目，m 行目から数字 num を除去し，セル (l,m) に数字 num を確定する。（つまり (l,m,num) を引数として (remove and substitute number) ルーチンを呼び出す。）
4．全てのセルを調べ終えたら終了し次の解法へ進む。

とすればよい。

4.6 Fortranによる実装

ではこれらの基本的な解法に対しFortranでの実装を考える。まずある数字 num に対して,領域内で数字 num が入る可能性が残っているセルの数（T の数）を調べる必要がある。これには配列用関数である count 関数を用いる。count 関数は論理型配列を引数とし,その配列内の T である要素の個数を返す関数である。例えば k=2 の領域内に 5 の入る可能性のあるセルの個数を調べるには論理立方体 cube の (1,4,5) 要素から (3,6,5) 要素までの部分配列内の T の個数を調べればよいので

```
1:      count(cube(1:3,4:6,5))
```

となる。この値が 1 であれば 5 がその領域内で確定する。次に 5 が入る場所を特定するには配列用関数である maxloc 関数を用いる。maxloc 関数は第一引数に整数型または実数型の配列をとり，第二引数に第一引数と同じ形の論理型配列をとる。そして第二引数の論理型配列の T である要素に対応する第一引数の要素内の最大値をとる要素の位置を 1 次元配列(大きさは引数の配列の次元数に等しい) で返す関数である。いま k=2 の領域内で 5 の入る可能性のあるセルの個数が 1 つであったとすると，部分配列 cube(1:3,4:6,5) 内の T の個数は 1 つであり，その T が入っている場所が 5 が入る場所になる。したがって cube(1:3,4:6,5) を第二引数とし，適当に初期化された 3 × 3 の整数型配列（例えば全ての要素を 1 で初期化する）を用意しそれを第一引数として maxloc 関数に渡せば，cube(1:3,4:6,5) の T の位置が大きさ 2 の配列で返されることになる。ただし返される位置は大域座標 (l,m) ではなく領域内座標 (i,j) であることに注意する。つまり

```
1:      integer tmparray(3,3),ij(2)
2:      tmparray = 1
3:      ij = maxloc(tmparray,cube(1:3,4:6,5))
```

とすれば配列 ij に k=2 の領域内で 5 の入る領域内座標(ij(1),ij(2)) が代入される。以上の手続きをまとめると以下のようになる。

```
1:      integer tmparray(3,3),ij(2)
2:      tmparray = 1
3:      do num = 1,9
4:      do k = 1, 9
5:      (1,1,k) → (l1,m1)
6:      if( count(cube (l1:l1+2,m1:m1+2,num)).eq. 1) then
```

```
 7:      ij = maxloc(tmparray,cube(l1:l1+2,m1:m1+2,num))
 8:      (ij(1),ij(2),k)  →  (l,m)
 9:      (remove and substitute number) (l,m,num)
10:      end if
11:      end do
12:      end do
```

3:と 4:から始まる do ループで１～９の数字を１～９番目の領域で順に調べる。5:では k 番目の領域の左上隅のセルの位置である (1,1,k) を (l1,m1) に変換している。6:では k 番目の領域内に含まれる数字 num の個数が１かどうかチェックしている。１であれば if ブロック内が実行され, 7:で num の入る位置を調べ, 8:でそれを座標 (l,m) に変換し, 9:で先に説明した数字の除去と確定をするルーチンを呼び出す。

行と列方向を調べるのは cube の部分配列が１次元になるので簡単である。行方向の場合は

```
 1:      integer tmparray(9), m(1)
 2:      tmparray = 1
 3:      do num = 1, 9
 4:      do l = 1, 9
 5:      if ( count(cube(l,:,num)) .eq. 1 ) then
 6:      m = maxloc(tmparray,cube(l,:,num))
 7:      (remove and substitute number) (l,m,num)
 8:      end if
 9:      end do
10:      end do
```

となる。なおmaxloc関数は配列を返すため, 1:で大きさ１の配列 m(1) を宣言している。列方向は上記のプログラムで l と m の役割を入れ替えればよい。深さ方向も同様で

```
 1:      integer tmparray(9), num(1)
 2:      tmparray = 1
 3:      do m = 1, 9
 4:      do l = 1, 9
 5:      if ( count(cube(l,m,:)) .eq. 1 ) then
 6:      num = maxloc(tmparray,cube(l,m,:))
 7:      (remove and substitute number) (l,m,num)
```

```
 8:        end if
 9:        end do
10:        end do
```

となる。

従来のFortran77であれば，部分配列内のTの個数を数えたり，その位置を調べたりするにはいくつもの do ループを回さなければなかった。しかしFortran90/95になって配列用関数が提供されたため上記のように実に簡潔に記述できるようになったのである。

5　高度な解法

簡単な問題であれば前節で解説した解法でパズルを解くことができる。しかし少し難しい問題になると基本的な解法では正解にたどり着かなくなる。この節では高度な解法を紹介し，Fortranで実装する。

5.1　2個の数字の巡回セル

図4の1番目の領域の大域座標 (1,3) と (2,3) のセルには，両方とも□で囲んだ1と5の2つの数字が残されている。これは (1,3) のセルに1が入れば (2,3) のセルには5が入り，逆に (1,2) のセルに5が入れば (2,3) のセルには1が入ることを意味している。つまり1と5はこの領域内ではこの二つのセル以外には入ることはできないのである。したがってこの領域内の別のセルから1と5を除去できる。ただし (1,3) と (2,3) のどちらのセルに1と5が入るかを確定することはできない。このようなセルを**2個の数字の巡回セル**と呼ぶことにする。

このことは行，列でも成立する。8行目を見てみるとこの行には1と4の2個の数字の巡回セルがあるので，8行目の別のセルから1と4を除去できる。

最初の1と5の巡回セルは両方とも3列目にあるので，実は3列目の別のセルからも1と5を除去することができる。

5.2　3個以上の数字の巡回セル

前小節で説明したことは，3個以上の数字の組みでも成立する。同一の領域，行，列内で，例えば1・2・3の3個の数字のみが含まれるセルが3つあれば，それらは3つの数字の巡回セルになり，それ以外のセルから1・2・3を除去できることになる。ここで注意しなければならないことは3つのセルに必ずしも3個の数字が全て残っていなければな

らないわけではない。例えば3つのセルにそれぞれ1・2，2・3，3・1のように3個の数字の内2個ずつ残されていたとしてもこれらは3個の数字の巡回セルになる。同様に4個以上の数字の巡回セルも考えられる。

では一般的に領域内のN個の数字の巡回セルを見つけるアルゴリズムは次となる（行内や列内も同様である)。

1．領域内からN個のセルを選択する。

2．選択したN個の全てのセルには2個以上の数字が入る可能性が残っている（0個と1個では巡回セルにならない)。

3．選択したN個のセルの深さ方向の論理立方体の部分配列(cube(l1,m1,:), cube(l2,m2,:),, cube(lN,mN,:)) に対して論理和をとったとき，Tの数がNであればそれらのセルはN個の数字の巡回セルを構成する。

では最も簡単なN=2の場合をFortranで実装してみよう。まず領域を調べるプログラムを考える。領域内のセルを参照するには変数nを使うことにする。2個の数字の巡回セルを見つけるわけだから2個のセルを参照する必要がある。そこで1つ目のセルをn1で，2つ目のセルをn2で参照する。2個のセルの全ての組合せを調べるため，

```
1:      do n1 = 1, 9 - 1
2:      do n2 = n1 + 1, 9
3:      .......
4:      end do
5:      end do
```

のようにdoループを回す。セル内の残された数字の数を調べるには先ほどと同様count関数を用いる。セル内に残っている数字が何であるかを調べるには配列用関数packを使う。pack関数は第一引数に任意の型の配列をとり，第二引数に第一引数と同じ大きさの論理型配列をとる。そして第二引数の論理型配列のTである要素に対応する第一引数の要素が配列より取り出されて1元配列で返される。したがってセル内に残っている2個の数字を取り出すには，

```
1:      integer num(2),number(9)
2:      number(:) = (/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /)
3:      (n1,k)  → (l1,m1)
4:      (n2,k)  → (l2,m2)
5:      num = pack(number,cube(l1,m1,:).or.cube(l2,m2,:))
```

とすれば大きさ2の配列numに2個の数字が返される。N=2の場合は5:のように論理立

方体の2つの部分配列の論理和をとる必要はないが，N>2の場合へ拡張を考慮して論理和をとった。ここで3:はn1→ (i1,j1)，(i1,j1,k) → (l1,m1)の2つの変換を連続して呼び出すことを表している。以上をまとめると次のようになる。

```
1:      integer num(2), number(9)
2:      number(:) =(/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /)
3:      do k = 1, 9
4:      do n1 = 1, 9 - 1
5:      do n2 = n1 + 1, 9
6:      (n1,k)  → (l1,m1)
7:      (n2,k)  → (l2,m2)
8:      if( ( count(cube(l1,m1,:) .ge. 2 ) .and.
9:     -    ( count(cube(l2,m2,:) .ge. 2 ) .and.
10:    -    ( count(cube(l1,m1,:).or.cube(l2,m2,:)) .eq. 2 ) ) then
11:     num = pack(number,cube(l1,m1,:).or.cube(l2,m2,:))
12:     (remove number) (l1,m1,l2,m2,k,num)
13:     end if
14:     end do
15:     end do
16:     end do
```

8:～10:でアルゴリズムの2と3の条件を満たしているかどうかを調べている。12:はk番目の領域内の巡回セル（(l1,m1), (l2,m2)）以外のセルから2つの数字numを除去している。除去する方法はアルゴリズム的に難しいことはないのでここでは具体的なプログラムについては述べない（以下同様）。

行に対しては同様に

```
1:      do l = 1, 9
2:      do m1 = 1, 9 - 1
3:      do m2 = m1 + 1, 9
4:      if( ( count(cube(l,m1,:) .ge. 2 ) .and.
5:     -    ( count(cube(l,m2,:) .ge. 2 )  .and.
6:     -    ( count(cube(l,m1,:).or.cube(l,m2,:)) .eq. 2 )) then
7:      num = pack(number,cube(l,m1,:).or.cube(l,m2,:))
8:      (remove number) (l,m1,m2,num)
```

```
 9:        end if
10:        end do
11:        end do
12:        end do
```

とすればよい。列に関しては上記のプログラムで l と m の役割を入れ替えればよい。N>3 についても同様にプログラムすることができる。

5.3 領域内の行と列

図4の6番目の領域に着目してみよう。領域内の上の行に□で囲んだ3が2個残っているのが分かる。そしてこの領域内の他のセルには3は残っていない。すると3はこれらの2つのセルのどちらかに入ることが分かる。したがってこれら2つのセルが含まれる4行目では他のセルには3が入らないことになる。よって大域座標 (4,4) にある3は除去することができる。つまり領域内のある1つの行内だけに，ある数字が2個以上入る可能性が残っていて，他の行にはその数字の入る可能性が残っていなければ，行方向の別の領域の同じ行からその数字を除去することができる。このことは列に関しても同様である。

では次に1つの行にだけある数字が2個以上残っていることを見い出すことを処理することを考える。それには2進数の各桁にセルを対応させていく。つまり領域内の1行目の3個のセルに $1=2^0$，$2=2^1$，$4=2^2$を，2行目のセルに8，16，32を，3行目のセルに64，128，256を重みとして与える。そしてある数字が残っているセルだけその数を加えて重みを算出する。するとその和が3，5，6，7であれば1行目だけに，24，40，48，56であれば2行目だけに，そして192，320，384，448であれば3行目だけにその数字が2個以上残っていることが分かる。

数字が残っているセルの重みの和を求めるには配列用関数 sum を用いる。sum は第一引数に実数型，整数型，複素数型の配列をとり，第二引数に第一引数と同じ大きさの論理型配列をとる[3)]。そして論理型配列の T である要素に対応する第一引数の要素の値の和が返される。例えば6番目の領域に3が残っているセルの重みの和を求めるには，

```
1:        integer weight(3,3)
2:        weight(1,:) = (/  1,   2,   4 /)
3:        weight(2,:) = (/  8,  16,  32 /)
4:        weight(3,:) = (/ 64, 128, 256 /)
5:        sum(weight, cube(4:6,7:9,3))
```

3）正確には省略可能な第二引数を省略したときの第二引数。

とすればよい。さらに和の値によって場合分けするため，Fortran90 より導入された select case 文を用いる。以上のことを踏まえてプログラムを作成すると，

```
 1:      integer weight(3,3)
 2:      weight(1,:) = (/  1,   2,   4 /)
 3:      weight(2,:) = (/  8,  16,  32 /)
 4:      weight(3,:) = (/ 64, 128, 256 /)
 5:      do num = 1, 9
 6:      do k = 1, 9
 7:      (1,1,k)  →  (l,m)
 8:      select case( sum(weight,cube(l:l+2,m:m+2,num)) )
 9:      case ( 3, 5, 6, 7 )
10:      l = l
11:      case ( 24, 40, 48, 56 )
12:      l = l + 1
13:      case ( 192, 320, 384, 448 )
14:      l = l + 2
15:      case default
16:      l = 0
17:      end select
18:      if ( l .ne. 0 ) (remove number) (l,k,num)
19:      end do
20:      end do
```

となる。ここでは 7 :で k 番目の領域の左上隅の座標 (l,m) を求め，8:から始まる select case 文で，重みの和に応じて数字が残っている行（ l の値）を設定している。いずれにも該当しない場合は 15:から始まる case default 文で l に 0 を代入する。18:は l 行目にだけ数字が残っていれば k 番目の領域以外の l 行目から，数字 num を除去するルーチンを呼び出すことを示している。列に関しては上記のプログラムで l と m の役割を入れ替えればよいが，列方向では重みの設定が行方向と変わってしまうため，転置行列を求める配列用関数 transpose を用いて 8:を

```
 8:      select case( sum(transpose(weight),cube(l:l+2,m:m+2,num)) )
```

と書き換えれば設定値をそのまま使うことができる。

5.4　２つの領域にまたがった行と列

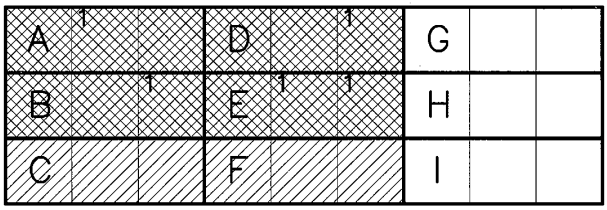

図５：行方向の３つの領域

図５は行方向の３つの領域を描いたものである。さらに各領域内の行方向の３つのセルをまとめて太線で囲み，それぞれをＡ～Ｉのアルファベットで区別した。今濃い網掛けをしたＡ，Ｂ，Ｄ，Ｅの４つの部分に注目する。これらの４つの部分全てに，それぞれの部分を構成する３つのセルの１つ以上のセルに同一の数字が入る可能性が残っている（図５の例では１）。そして薄い網掛けをしたＣとＦにはその数が入る可能性が残っていない。すると１がＡ内のどこかのセルに入るとすると，中央の領域ではＥ内のどこかのセルに１が入ることになり，逆に１がＢ内のどこかのセルに入るとすると，中央の領域ではＤ内のどこかのセルに１が入ることになる。いずれの場合も右側の領域のＧとＨの部分には１が入る可能性がなくなり，これら６個のセルから１を除去することができるのである。これは別の行の組合せでも同様であるし，３つの領域の別の組合せでも同様である。つまりある数字について行方向の３つの領域の内，２つの領域内の共通する２つの行のセル内にのみ入る可能性が残っている場合，残りの領域の共通行内からその数字を除去できるのである。このことは列に関しても同様に成立する。ではこのような行を見つけるアルゴリズムを考える。いまＡ～Ｉを構成する９個の部分それぞれの３つのセルをまとめて考えて，３つのセルの中のいずれかにある数字が残っていればＴ，残っていなければＦとする。Ａ～Ｉの部分を表す３×３の論理型配列を用意すればこの状態を表現することができる。例えば

```
1:      logical area3(3,3)
```

とする。この論理型配列の要素がＴであるかＦであるかを判定するには，論理立方体のそれぞれのセルを表す３つの要素の論理和をとればよいが，ここでは配列用関数であるany関数を用いる。anyは論理型配列を引数にとり，その配列の要素内に１つでもＴがあればＴを，１つもＴがなければＦを返す関数である。いまＡの部分の左端のセルの座標が(l,m)であったとすると，ある数numに対するＡの３つのセルを表す論理立方体の部分配列はcube(l,m:m+2,num)であるから

```
1:      area3(1,1) = any(cube(l,m:m+2,num))
```

とすればよいことが分かる。では例えば図５のような状態になっていることをどのように判断すればよいであろうか。まず右側の領域Ｇ，Ｈ，Ｉの状態はどのようになっていてもかまわないので，左側と中央の領域だけに着目する。そしてＡ，Ｂ，Ｄ，Ｅの部分がＴで，

CとFの部分がFであることを判断するには，前小節と同様に各部分に適当に重みをつけてTである部分の和をとれば判断することができる。例えば重みとして

```
1:      integer weight(3,3)
2:      weight(1,:) = (/  1,  2, 0 /)
3:      weight(2,:) = (/  4,  8, 0 /)
4:      weight(3,:) = (/ 16, 32, 0 /)
```

とする。ここで重みを表す配列の3列目を全て0することによって，論理型配列area3の3列目の状態を無視することができる。そして

```
1:      sum(weight,area3)
```

の値が15であれば図5の状態になっていることが分かる。同様にこの値が51であればA，C，D，Fの部分がTでBとEの部分がF，60であればB，C，E，F，の部分がTでAとDの部分がFであることが分かる。これで左側と中央の領域に対して，全ての行の組合せを調べることができる。

別の領域の組合せを調べるには，3種類の重みの配列を用意するという方法もあるが，ここでは配列用関数のcshift関数を用いることにする。cshift関数は配列の要素を循環的に移動(シフト)する関数である。第一引数にシフトしたい配列，第二引数にシフトする大きさ，第三引数にシフトする次元(2次元配列の場合，1であれば列方向，2であれば行方向にシフトする)をとる。シフトする方向は第二引数の正負で指定する(行方向のシフトであれば，正の数で左に，負の数で右にシフトする)。つまりcshift関数で論理型配列area3を左に1だけシフトすれば中央と右側の領域を同じ重みの配列で調べることができ，同様に左に2だけシフトすれば左側と右側の領域を調べることができる。3だけシフトすると元に戻るわけだが，1，2，3と順にシフトすればシフトした大きさが除去するべき領域の位置（以下のプログラムのvshift）を示すことになるのでプログラムしやすくなる。以上をまとめて行方向の3つの領域を調べるプログラムを作成すると以下のようになる。

```
1:      integer u,v,vshift,weight(3,3)
2:      logical area3(3,3)
3:      weight(1,:) = (/  1,  2, 0 /)
4:      weight(2,:) = (/  4,  8, 0 /)
5:      weight(3,:) = (/ 16, 32, 0 /)


6:      do num = 1, 9
```

```
 7:      do u = 1, 3

 8:      do v = 1, 3
 9:      do i = 1, 3
10:      (u,v)  → k,(i,1,k) → (l,m)
11:      area3(i,v) = any(cube(l,m:m+2,num) )
12:      end do
13:      end do

14:      do vshift = 1, 3
15:      select case(sum(weight,cshift(area3,vshift,2)))
16:      case(15)
17:      (i1,i2) ← (1,2)
18:      exit
19:      case(51)
20:      (i1,i2) ← (1,3)
21:      exit
22:      case(60)
23:      (i1,i2) ← (2,3)
24:      exit
25:      end select
26:      end do
27:      if( vshift .ne. 4 ) (remove number) (u,vshift,i1,i2,num)

28:      end do
29:      end do
```

まず 1:〜5:で必要な変数と配列の宣言と初期化を行っている。6:と 7:で 1 × 9 までの数字と，上段から下段まで u の値を順に変えて調べる do ループを回している。8:〜13:で論理型配列 area3 を初期化している。10:は領域の座標 (u,v) を k に変換し，k 番目の領域の i 行目の一番左のセル (i,1) の座標 (l,m) を求めることを示している。次に 14:〜26:の do ループで論理型配列 area3 を行方向に左シフトしながらその状態を select case 文で調べている。重みの和が上記で述べた15，51，60のいずれかであれば i1 と i2

に除去対象となる行を代入し，exit 文で do ループを抜ける。exit 文で do を抜けると，変数 (u,vshift) の組が除去対象となる領域の座標を表していることに注意して欲しい。除去対象となる条件を満たさなければ，do ループは最後まで実行され vshift の値は 4 になる。27:は vshift の値が 4 以外であれば，除去対象となる領域の座標 (u,vshift)，除去対象となる行 (i1,i2)，除去する数字numを引数として除去ルーチンを呼び出すことを示している。

列方向の 3 つの領域を調べるプログラムは u と v，i と j，l と m の役割を入れ替えればよいが，前小節と同様に15:は

```
15:       select case(sum(transpose(weight),cshift(area3,ushift,1)))
```

と書き換える必要がある。

5.5 四角の対角線

図 6 の 3 行目と 7 行目にはまだ確定していない 2 つのセルがそれぞれの行に残っている。3 行目の 2 つのセルには 3 または 7 が入り，7 行目の 2 つのセルには 3 または 9 が入る。そしてこれらのセルには数字 3 が共通に残されており，また列の位置がそろっていて丁度四角形の四隅(対角線) に位置している。これは図の○で印をつけた対角線上のセルに 3 が入るか，●で印をつけたセルに 3 が入るかのいずれしかないことを意味している。このことを列方向から見ると，○か●のセルにしか 3 が入り得ないため，列方向の他のセルから 3 を除去できるのである。これは必ずしも 2 つの行に未確定のセルが 2 つだけ残っていなければならないわけではない。一般的に，ある数字 num が入ることのできるセルが 2 つだけである行が 2 行あって，それぞれの行のセルの列位置が 2 つとも一致していれば，それらのセルを含む 2 つの列の他のセルから数字 num を除去できるのである。ただし 4 つのセルが同一の領域内にあってはならない。

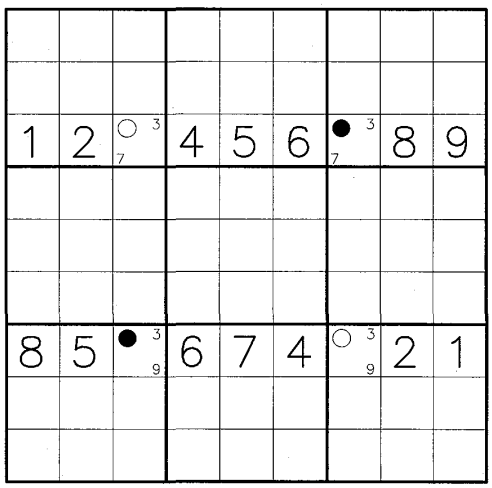

図 6 ：四角の対角線の例

ではこのような四角の対角線をなすセルを見つけるアルゴリズムを考えると，

1．2 つの行を選択する。

2．ある数字 num に対して，行方向の論理立方体の部分配列 (cube(l1,:,num),cube(l2,:,num)) 内の T の数が 2 つの行とも 2 であれば 3. へ進み，そうでなければ次の行の組合せに進む。

3．その2つの部分配列の論理積をとったときもTの数が2であれば4.へ進み，そうでなければ次の行の組合せに進む。

4．4つのセルが同一の領域内になければこのTのある2つの列について調べた2つの行以外のセルから数字numを除去する。

5．全ての行の組合せを調べ終えたら次の解法へ進む。

となる。これは列方向についても同じである。以上をまとめると行方向に対して以下のようなプログラムを作ることができる。

```
 1:       integer m(2),number(9)
 2:       number(:) = (/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /)
 3:       do num = 1, 9
 4:       do l1 = 1, 9 - 1
 5:       do l2 = l1 + 1, 9
 6:       if ( ( count(cube(l1,:,num)) .eq. 2 ) .and.
 7:      -     ( count(cube(l2,:,num)) .eq. 2 ) .and.
 8:      -     ( count(cube(l1,:,num).and.cube(l2,:,num)) .eq. 2 ) ) then
 9:       m = pack(number,cube(l1,:,num))
10:       if ( ( (l1-1)/3   .ne. (l2-1)/3   )  .and.
11:      -     ( (m(1)-1)/3 .ne. (m(2)-1)/3 ) ) then
12:       (remove number) (l1,l2,m,num)
13:       end if
14:       end if
15:       end do
16:       end do
17:       end do
```

プログラムの構造としては巡回セルのプログラムとほぼ同じである。4:と5:で2つの行を選択し，6:～8:で上の2と3の条件を満たすかどうかをif文を使って調べている。9:ではpack関数を用いて何列目に数字numが残っているかを調べ，大きさ2の配列mに格納している。10:と11で，4つのセルが同一の領域内にないことを調べ，なければ12:でl1行目とl2行目のセル以外の，m(1)列とm(2)列内のセルから数字numを除去するルーチンを呼び出す。列方向に関してはlとmの役割を入れ替えればよい。

5.6 浜田ロジック

図7は浜田ロジックと呼ばれる解法の例である。図から○と●の印をつけたセルには3または9が入るが，位置関係から○には同じ数字が入ることが分かる。2行目の△には3または7が入るが，○に3が入るとすると，2行目に3の入るセルがなくなってしまう。したがって○には9が，●には3が入ることになる。つまりアルゴリズムとしては

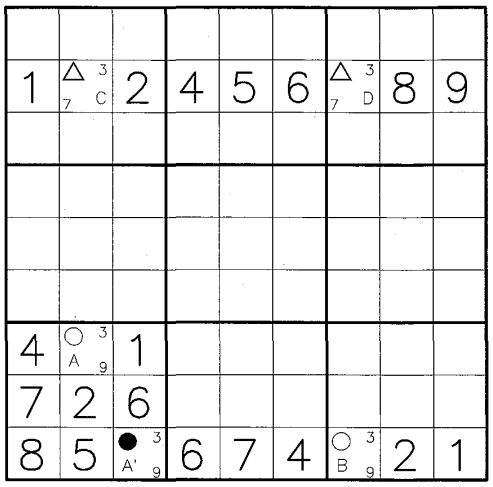

図7：浜田ロジックの例

1．ある領域に2個の数字 num1 と num2 の巡回セルAとA'がある。
2．A'と同じ行のセルB（別の領域である）が2個の数字 num1 と num2 の巡回セルになっていれば3．へ進み，そうでなければ次の領域に進む。
3．A'，Bと別の行に2個の数字 num1 と num3 の巡回セルCとDがあれば4．へ進み，そうでなければ次の領域に進む。
4．AとBの列位置がCとDと同一であれば，AとBから共通に残っている数字 num1 を除去する。
5．全ての領域を調べ終えたら次の解法へ進む。

となる。

```
 1:      integer l1tmp(2),m1tmp(2),num(2),number(9)
 2:      logical mpos(9)
 3:      number(:) = (/ 1, 2, 3, 4, 5, 6, 7, 8, 9 /)
 4:      do k1 = 1, 9
 5:      if ( (find2pairs) ( k1, l1tmp, m1tmp ) .eq. .true. ) then
 6:      do k2 = 1, 9
 7:      if ( k1 .ne. k2 ) then
 8:      do n2 = 1, 9
 9:      if ( (find1pair) ( k2, n2, l2, m2 ) .eq. .true ) then
10:      if ( count(cube(l1tmp(1),m1tmp(1),:)*cube(l2,m2,:)).eq. 2 ) then

11:      lflag = 0
12:      if ( l1tmp(1) .eq. l2 ) then
13:      lflag = 1
```

```
14:       l1 = l1tmp(2)
15:       m1 = m1tmp(2)
16:       else if ( l1tmp(2) .eq. l2 ) then
17:       lflag = 1
18:       l1 = l1tmp(1)
19:       m1 = m1tmp(1)
20:       end if
21:       if ( lflag .eq. 1 ) then
22:       num = pack(number,(cube(l1,m1,:)))
23:       mpos = .false.
24:       mpos(m1) = .true.
25:       mpos(m2) = .true.
26:       do l = 1, 9
27:       if ( (l .ne. l1) .and. (l .ne. l2) ) then

28:       do i = 1, 2
29:       if ( ( count(cube(l,:,num(i))) .eq. 2 ) .and.
30:      -     ( count(mpos(:).and.cube(l,:,num(i))) .eq. 2 ) ) then
31:       cube(l1,m1,num(i)) = .false.
32:       cube(l2,m2,num(i)) = .false.
33:       end if
34:       end do

35:       end if
36:       end do

37:       end if

38:       end if
39:       end if
40:       end do
41:       end if
```

```
42:       end do

43:       end if
44:       end do
```

このプログラムでは 4:と 6:で始まる do ループで，k1 番目の領域に 2 つの数字の巡回セルがあり，k2 番目の領域に同じ行の巡回セルを構成するセルがあるかどうかを調べている。この場合 k1 と k2 番目の領域は対等ではないので，巡回セルや四角の対角線などで用いた do ループの使い方はできない。5:の(find2pairs) (k1, l1tmp, m1tmp) は，k1 番目の領域内に 2 つの数の巡回セルがあるかどうかを調べる関数を呼び出している。この関数は 2 つの数の巡回セルがあれば l1tmp, m1tmp にその座標を返し，関数の戻値として Tを返すものとする。そしてなければ F を返すものとする。この関数に関しては領域内で 2 個の数字の巡回セルを見い出すプログラムと本質的に同じなのでここでは省略する。k1 番目の領域に対して，残りの全ての領域を調べる必要はない(同じ行，列にある領域だけを調べればよい) のだが，プログラムが複雑になるので 6:で始まる do ループでは，全ての領域に対してループを回すことにする。ただし k1 と同じ領域を調べるのは無意味なので 7:の if 文で実行しないようにしている。9:の (find1pair) (k2, n2, l2, m2) は k2 番目の領域内の n2 番目のセルに数字が 2 個残っているかどうかを調べる関数を呼び出している。この関数はセル内に数字が 2 個残っていればその座標 (l2,m2) を返し，関数の戻値として T を返すものとし，2 個残っていなければ F を返すものとする。この関数に関しても (n2,k2) → (l2,m2) の変換と，count(cube(l2,m2,:)) が 2 であるかどうかを調べるだけのプログラムなのでここでは省略する。10:では k1 内の巡回セルの 1 つと，別の領域内のセルに残っている数字が同じであるかどうか（同一行あるいは同一列の巡回セルを構成する可能性があるかどうか）を調べている。11:～ 20:では図 7 においてＡとＢが同一行の巡回セルになるか，A'とＢが同一行の巡回セルになるかをみて，巡回セルになれば数字を除去する可能性のあるセルの座標を (l1,m1) に代入している。これで数字を除去する可能性のある列の位置が m1 と m2 に格納されたことになる。このとき lflag が 1 になれば上で述べたアルゴリズムの 1 と 2 の条件が満たされたことになる。

条件が満たされれば22:で巡回セルの 2 つの数字を num に代入する。23:では大きさ 9 の論理型配列 mpos を F で初期化し 24:と 25:で mpos の m1 と m2 の要素に T を代入している。これで数字を除去する可能性のある列位置が mpos に格納されたことになる。次に26:と 27:で l1, l2 以外の全ての行に対して do ループを回す。28:で 2 つの数字 num(1) と num(2) について do ループを回す。この 2 つの数字の内 1 つを num(i) とし，

この num(i) について29:では l 行目に num(i) が2個残っているかを調べ，30:では l 行目に残っている数字の列位置が m1 と m2 に等しいかを mpos と cube(l,:,num(i)) との論理積をとって判断している。判断処理を通れば上で述べた3と4の条件を満たすので，31:と 32:で数字 num(i) の除去処理となる[4)]。列方向に関しては l と m の役割を入れ替えればよい。

5.7 仮置き

前小節までで現在知られている全てのナンバープレイスの解法アルゴリズムとFortranによるプログラムを紹介した。しかしこれらの解法を全て適用してもなお解くことができない例もある。このような場合は未確定のセルに，入る可能性のある数字を試しに確定(仮置き）して，再び解法を適用してみるしかない。問題が無矛盾に作られているのであれば，いつか正解にたどり着くことができるはずである。

6 全体の流れ

4節と5節で説明した解法プログラムを用いた全体的なプログラムの流れを以下に示す。

```
 1:      logical cube(9,9,9)
 2:      integer array(9,9),status
 3:      call initialize( cube, array )
 4:      status = 0
 5:      do while( status .eq. 0 )

 6:      call solver( status, cube, array )

 7:      select case (status)
 8:      case (1)
 9:      call solver2( status, cube, array )
10:      exit
11:      case (2)
12:      exit
```

4）本当はこの2つのセルは num(i) でないもう一つの数字で確定するのであるが，4節で解説した確定ルーチンと区別するため，ここでは除去することにした。

```
13:       case (3)
14:       exit
15:       end select

16:       end do

17:       select case (status)
18:       case (1)
19:       call answer( 'unable to solve', array )
20:       case (2)
21:       call answer( 'inconsistent', array )
22:       case (3)
23:       call answer( 'answer', array )
24:       end select

25:       end
```

1:と 2:で必要な変数を宣言している。3:で初期化を行うルーチンを呼び出す。このルーチンで行うことは,

1．パズルのデータを入力し，cube と array を初期化する。この時 array は図 1 のように，cube は図 4 のように初期化される[5)]。

2．終了判断を行うためにダミーの論理立方体 dummycube を全て T で初期化する。

である。次に 4:で変数 status を 0 で初期化し，5:で始まる do while ループに入る。6:では 4 節と 5 節で解説した解法プログラムを順に実行するルーチン solver を呼び出している。このルーチンではすべての解法を一度だけ順に適用した後に，論理立方体 cube の状態を調べ終了判断を示す変数 status を返す。ここで行う終了判断は以下の通りである。

1．論理立方体 cube の要素に T が残っているが，前回 solver ルーチンを実行する前のcubeをコピーしておいた dummycube と，実行後の cube を比較して，その要素に変化があれば，このパズルはまだ解けていないとして status=0 を返し，

5）図 1 で数字が入っていない部分は例えば仮に 0 を代入しておく。cube は図 4 で数字が残っている要素には T が，残っていない要素には F が，数字が確定しているセルの全ての要素には F が代入される。

dummycube に cube の内容をコピーする。

2．論理立方体 cube の要素に T が残っているが，前回 solver ルーチンを実行する前の cube をコピーしておいた dummycube と，実行後の cube を比較して，その要素に変化がなければ，このパズルは解けないとして status=1 を返す。

3．論理立方体 cube の全ての要素が T であるが，解 array の要素に矛盾があった場合，矛盾があるとして status=2 を返す。

4．論理立方体 cube の全ての要素が T であり，解 array の要素に矛盾がなければパズルが解けたとして status=3 を返す。

7:から始まる select case 文で，変数 status の値によって処理を分岐させている。値が0であれば do while ループが再び実行され solver が呼ばれる。値が2と3であれば do while ループを抜ける。値が1の場合は例外である。値が1の意味は，4節と5節で説明した解法をすべて適用してももはや除去できる数字がないという意味であって，このパズルが矛盾していて解けないという意味ではない。値が1の場合は 9:の solver2 ルーチンを呼び出し，5.7節で説明した仮置きを行う。solver2 ルーチンでは未除去の数字を1つ確定したとして solver ルーチンを呼び出し，解が求まるかどうかを調べる。解けない場合，次の未確定の数字を確定したとして再び solver ルーチンを呼び出し，解が求まるまで繰り返すのである。すべての未確定の数字を仮置きしても解が求まらない場合は，このパズルは解けないとして status=1 を返し，解が求まれば status=3 を返し，10:で do while ループを抜ける。このようにして do while ループを抜けたら，17:行目から始まる select case 文で変数 status の値によってメッセージと最終の array を表示する answer ルーチンを呼び出してプログラムを終了する。

7　結語

ナンバープレイスの本は多数出版されている。その中でも難解なものを集めた本に収録されているパズルを，今までに500問程度解いてみたが，ここで紹介した解法アルゴリズムで解けない問題はまだない。解法に関しては本稿で紹介したもので尽きるようである。

また Fortran90 以降に導入された配列用関数と，論理型配列を用いることによって，無用な do ループを回すことなく解法アルゴリズムを実にシンプルに実装することができることが分かった。最後に本プログラムの開発は，Turbolinux10上で Intel Fortran Compiler 9 を用いて行った。Intel Fortran Compiler は非商用利用であれば無償で利用可能である。無償提供して頂いた Intel 社に感謝する。

参考文献

西尾徹也, 2006,『西尾徹也のナンプレ』世界文化社
ウェイン・グールド, 2006,『ナンプレ２』角川書店
西尾徹也, 2005,『ナンプレ超上級編10』世界文化社
西尾徹也, 2006,『ナンプレ超上級編11』世界文化社
西尾徹也, 2006,『ナンプレ超上級編12』世界文化社