Program Promenade

連載

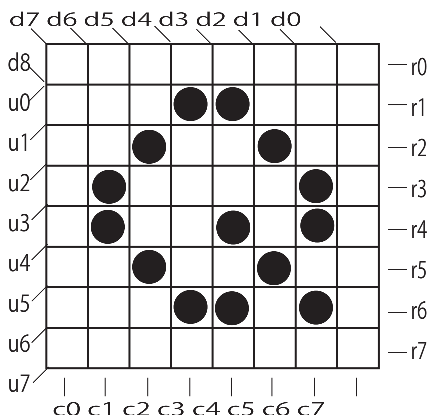
ディジタルトモグラフィ

川合 慧（東京大学 総合文化研究科）
kawai@graco.c.u-tokyo.ac.jp

■ディジタルトモグラフィ

今回の問題は，ACM Programming Contest, South Pacific Regionals 2000 の 第 8 問，"Discrete Digital Tomography" である．図-1 に，問題文に出ていた図を示す．図は全体としてはピザボックスを表していて，外からは見えないが，内部が格子状の区画に区切られている．それぞれの区画には，"ピザ饅" が高々 1 つ入っている．ピザボックスの上面と下面はアルミホイルが張ってあるが，側面はただの紙であり，内部の区切りも紙でできている．この状態で，内部のピザ饅の配置を調べるのがこの問題である．

調べるためには β 線 (!) を使う．ピザ饅は β 線を若干吸収するので，側面の片方から β 線を照射して他方で測定すると，その間にあるピザ饅の数が分かる．ただし，どの位置にあるかまでは分からない．測定は左右（r 方向），上下（c 方向），斜め（u および d 方向）に行う．この測定値から，内部のピザ饅の配置を決定する．すぐ分かるように，この設定は，医師が泣いて喜んだという話もある CT (Computed Tomography) の簡略版である．



rproj = 0 2 2 2 3 2 3 0
cproj = 0 2 2 2 3 2 3 0
uproj = 0 0 0 0 3 2 0 2 1 2 3 0 1 0 0
dproj = 0 0 0 0 3 2 0 4 0 2 3 0 0 0 0

図-1　ディジタルトモグラフィ

■問題の性格

まず若干の考察をする．1 つの照射器から測定器に至る直線上に並んでいる区画の列を**射線**と呼ぶことにすると，r 行 c 列のピザボックスの場合，射線は，

横向きに r 本，
縦向きに c 本，
右上がりの斜めに $r+c-1$ 本，
右下がりの斜めに $r+c-1$ 本，

だけ設定できる，このことから，得られる条件の総数が $3(r+c)-2$ であるように思えるが，どんな方向で測ってもピザ饅の総数は同じなので，独立な条件の数はこれより 3 だけ少ない．これに対して，それぞれの区画にあるピザ饅の数を未知数とすると，その数は全部で rc である．連立方程式によって解が求められるためには，条件の数が未知数の数以上であればよいので，$3r+3c-5 \geq rc$ となるが，変形すると $(r-3)(c-3) \leq 4$ となる．ただしこの問題では未知数の値は 0 か 1 に限られ，たとえば $a+b+c=0$ からただちに $a=b=c=0$ と決まるので，少しゆるい条件でも解ける可能性はあるが，ここではこれ以上立ち入らない．

この式で面白いのは，r（あるいは c）が 3 であれば，他方がいくら大きくても解けることである（図-2）．

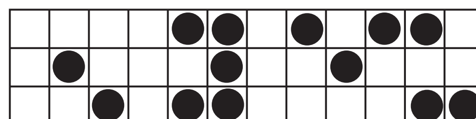


図-2　r=3，c=12の場合の例

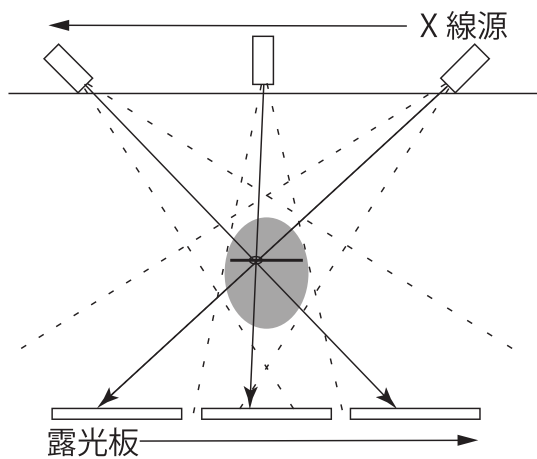


図-3　断層写真の原理

ただし r と c が 4 以上であると，この条件を満たす場合はほとんどなくなるので，他の方法を使わざるを得なくなる．

■断層写真

人体の断面図を撮影する方法は，CT の発明以前にも存在していた．断層写真と呼ばれるその方法の原理を図 -3 に示す．

要点は，上部からの X 線の照射と下にある露光板を，同期をとって反対方向に動かすことである．このようにすると，露光板の特定の場所には，常に被写体の特定の場所を通過した X 線が照射される．もちろんこの X 線は別の場所も通過してくるが，それらの場所の像は露光板上を移動するのでぼやけるのに対し，目的の場所の像は動かないので，相対的に最もはっきりと露光される．もちろんこれは「相対的に」であり，そばに強い像を作る部分があると肝心の部分の断層像の質が低下する．断層写真の原理は，ある特定の場所を含むいろいろな方向から調べることによって，その場所の様子を求めることにある．

■逆射影法

射影（projection）とは，一般的に言うと，情報を落して次元を減少させる演算である．たとえば，陽の光でできる樹木の影は，もとの樹木が 3 次元の物体であるのに対し，2 次元の像となっている．影はもとの物体のある側面は表しているが，もとの物体を完全に再現することはできない．別の物体がまったく同じ影を生ずることがあるからである．ただし，いろいろな方向への影を総体的に見ると，もとの物体の状況がだんだん明確に分かるようになる．射影像をもとの空間へ引き戻すことを逆射影（back projection）と呼ぶ．今問題としている状況では，この逆射影を使うことが考

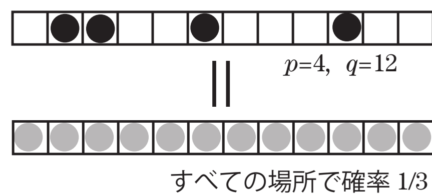


図-4　射影値とその意味

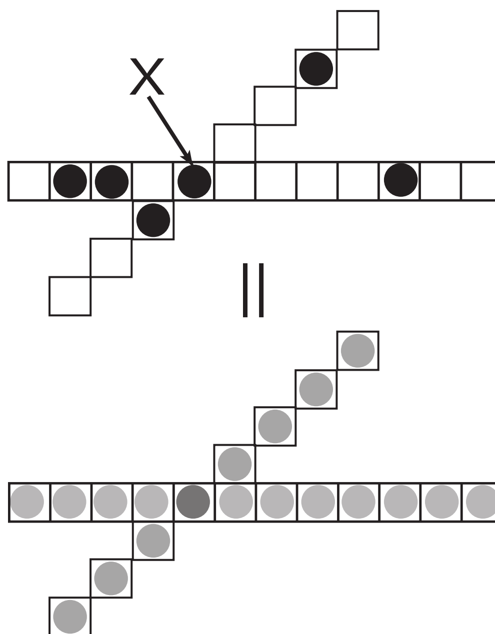


図-5　射影値と共通区画

えられる．断層写真の場合と同じように，多方向の射影を勘案して特定の部分のもとの値を推定するわけである．

今，長さ q の射線沿いにピザ饅が p 個存在するものとしよう（図 -4）．つまり，q 個の場所のうちのどれか p 個の位置にピザ饅が存在するという状況である．この情報から言えることは，せいぜい

　　q 個のすべての場所について
　　ピザ饅が存在するらしい確率が p/q

であろう．ほかに情報がない．

次に，この射線と区画 X で交差する別の射線を考える．この射線沿いにピザ饅が p' 個存在するものとし，その射線の長さを q' とすれば，さきほどと同様なことが言える（図 -5）．

それではこの状況で，区画 X については何が分かるのであろうか．実は，確定的なことは何も言うことができない．区画 X にピザ饅があってもなくても，この 2 つの状況は起こり得るからである．

逆に言うと，何も確定的なことが言えないのなら，区画 X に「p/q と p'/q' から導かれる値に応じてピザ

饅があるらしい」と考えても構わないことになる．別の言い方をしてみよう．区画 X を通る多数の射線がピザ饅を検知した場合には，区画 X にピザ饅がある可能性が高いと言ってもよい．

■逆射影の重ね合わせ

「p/q と p'/q' から導かれる値」について考えよう．この値，すなわち区画 X にピザ饅がありそうな度合を示す値を $F(X)$ とすると，p/q が大きくなれば（p'/q' が同じであっても）大きくなるのが自然である．p'/q' についても同様である．また，片方の値がゼロであれば，区画 X にピザ饅はない．つまり $F(X)=0$ のはずである．これらの性質は，単純に

$$F(X) = (p/q)(p'/q')$$

とすれば満たされる．この値を**合成射影値**と呼ぼう．この考え方でプログラムを作ってみよう．言語は Java であるが，特別な機能は用いていない．

■射影の扱い

問題は，4 つの 1 次元配列に与えられた射影の値から，$r \times c$ の 2 次元配列の内容を求めることである．射影はそれぞれ

```
int [] rproj = new int[r],      // 行方向
       cproj = new int[c],      // 列方向
       uproj = new int[r+c-1], // 右上方向
       dproj = new int[r+c-1]; // 右下方向
```

とする．また，部分的推測解のために

```
int [][] guess = new int[r][c];
```

を，逆射影の計算のために

```
double [][] image = new double[r][c];
```

を，それぞれ用意する．逆射影の計算は整数値では収まらないため，倍精度実数（double）で計算する．

各射線の長さは，行方向と列方向とでは一定値であるが，斜め方向では変化する．角をかすめるような射線では短く，中央を通過する射線では長くなる（**図-6**）．これらの値を，射影に対応する配列に計算しておく．

```
int [] rband = new int[r],
       cband = new int[c],
       uband = new int[r+c-1],
       dband = new int[r+c-1];
void makeband() {
  int k, m, min;
  for(k=0; k<r; k++) rband[k]=c;
  for(k=0; k<c; k++) cband[k]=r;
  for(k=0, m=r+c-2; k<r && k<c;
                     k++, m--) {
```

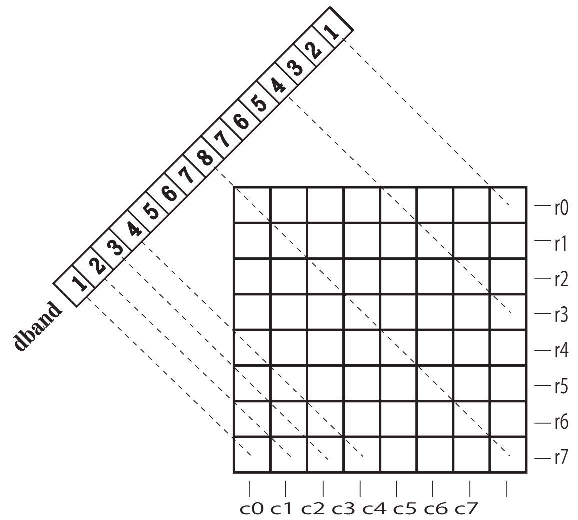


図-6　射線の長さ

```
    uband[k]=k+1; uband[m]=k+1;
    dband[k]=k+1; dband[m]=k+1;
  }
  min=k;
  for(k=k; k<=m; k++, m--) {
    uband[k]=min; uband[m]=min;
    dband[k]=min; dband[m]=min;
  }
}
```

これらの値は，ピザ饅の位置を 1 つ推測するごとに，その区画を通る射線について 1 だけ減らす．この値を**射線の実効長**と呼ぼう．rband や cband のような，値が一定のものも配列に入れているのは，この実効長の変化に対応するためである．

■計算の全体

計算は，全体としては以下のようになる．

- データを入力する
- 推測解をクリアする
- ピザ饅の数だけ
  - 逆射影を行なう
  - 最もありそうな位置を求め，推測解に加える
  - 射影の値を調整する

を繰り返す

この方法の中核である逆射影は次のようになる．

```
void backprojection() {
  int i, j;
  for(i=0; i<r; i++)
    for(j=0; j<c; j++)
      if(guess[i][j]==1) // 既に「あり」と
        image[i][j]=0.0; // 推測した場所
      else {
        image[i][j] =
           (double)rproj[i]/rband[i]
          *(double)cproj[j]/cband[j]
```

```
            *(double)uproj[i+j]/uband[i+j]
            *(double)dproj[i-j+c-1]
                    /dband[i-j+c-1];
      }
}
```

「最もありそうな位置」を単純に求めるには，配列全体を走査して image の最大値の位置を探せばよい．

```
int maxi, maxj;
void findmax() {
  int i,j;
  double w=-1.0;
  for(i=0; i<r; i++)
    for(j=0; j<c; j++)
      if(image[i][j]>w) {
        w=image[i][j];
        maxi=i;
        maxj=j;
      }
}
```

findmax で最大の逆射影値を与える位置 (maxi, maxj) が求められたら，以下の手続きを decreaseproj (maxi, maxj) と呼んで射影の調整を行う．

```
void decreaseproj(int i, int j) {
  rproj[i] --; // 射影値を 1 減らす
  rband[i] --; // 射線の実効長を減らす
  cproj[j] --;
  cband[j] --;
  uproj[i+j] --;
  uband[i+j] --;
  dproj[i-j+c-1] --;
  dband[i-j+c-1] --;
}
```

例題を処理する過程を見てみよう．なおこれ以降では第 s 行の第 t 列に位置する区画を [s,t] で表すことにする．

```
---- 入力 ----
r = 8, c = 8
rproj = 0 2 2 2 3 2 3 0
cproj = 0 2 2 2 3 2 3 0
uproj = 0 0 0 0 3 2 0 2 1 2 3 0 1 0 0
dproj = 0 0 0 0 3 2 0 4 0 2 3 0 0 0 0
---- 射線の長さの計算 ----
rband = 8 8 8 8 8 8 8 8
cband = 8 8 8 8 8 8 8 8
uband = 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
dband = 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
---- 逆射影（1 回目）----
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0124 0187 0000 0000 0000
0000 0000 0187 0000 0000 0093 0000 0000
0000 0124 0000 0000 0000 0029 0187 0000
0000 0187 0000 0000 0100 0000 0281 0000
0000 0000 0093 0029 0000 0187 0000 0000
0000 0000 0000 0187 0281 0000 0234 0000
0000 0000 0000 0000 0000 0000 0000 0000
```
（小数点は左端にあり，省かれている）

ここですでに，最終的な答に対応する区画での合成射影値が大きいことが分かる．この配置に関する限り，合成射影値の大きい方から 14 個を選べば，即，正解が得られる．ただし，正解には含まれていないが値がゼロではない区画が [3,5]，[5,3] と 2 つもあることには注意しなければならない．この場面では，合成射影値が最大 (0.0281) である区画 [4,6] が第 1 番目の推測区画となる．

```
---- 射影と長さの更新（変更は [ ] で示す）----
rproj = 0 2 2 2[2]2 3 0
rband = 8 8 8 8[7]8 8 8
cproj = 0 2 2 2 3 2[2]0
cband = 8 8 8 8 8 8[7]8
uproj = 0 0 0 0 3 2 0 2 1 2[2]0 1 0 0
uband = 1 2 3 4 5 6 7 8 7 6[4]4 3 2 1
dproj = 0 0 0 0 3[1]0 4 0 2 3 0 0 0 0
dband = 1 2 3 4 5[5]7 8 7 6 5 4 3 2 1
---- 逆射影（2 回目）----
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0075 0187 0000 0000 0000
0000 0000 0187 0000 0000 0093 0000 0000
0000 0124 0000 0000 0000 0017 0142 0000
0000 0142 0000 0000 0076 0000 0000 0000
0000 0000 0093 0029 0000 0156 0000 0000
0000 0000 0000 0187 0234 0000 0178 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

今度は，合成射影値が最大 (0.0234) である [6,4] が推測区画となる．

…途中は省略して…

---- 推測状況（図 -7）----

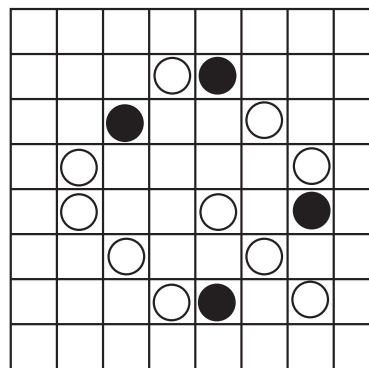


図-7　途中結果

図 -7 では推測済みを●で，正解のうち未推測の部分を○で，それぞれ表している．

```
---- 射影と長さ ----
rproj = 0 1 1 2 2 2 2 0
rband = 8 7 7 8 7 8 7 8
cproj = 0 2 1 2 1 2 2 0
cband = 8 8 7 8 6 8 7 8
uproj = 0 0 0 0 2 1 0 2 1 2 1 0 1 0 0
uband = 1 2 3 4 4 5 7 8 7 6 3 4 3 2 1
dproj = 0 0 0 0 2 1 0 3 0 1 3 0 0 0 0
dband = 1 2 3 4 4 5 7 7 7 5 5 4 3 2 1
```

射影（proj）と実効長（band）とが，適切に調整されているのが分かる．

```
---- 逆射影（5 回目）----
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0035 0000 0000 0000 0000
0000 0000 0000 0000 0000 0044 0000 0000
0000 0062 0000 0000 0000 0017 0119 0000
0000 0085 0000 0000 0029 0000 0000 0000
0000 0000 0053 0017 0000 0089 0000 0000
0000 0000 0000 0142 0000 0000 0116 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

ここでは，最大 (0.0142) である [6,3] が推測区画となる．先ほどの（正解ではない）区画については値が 0.0017 となっており，正解区画の最小値 (0.0029) に迫っているので，一抹の不安を覚える．

この例題では，このあと順調に推測を続けて，最終的には正解を与えて終了する．

■ゼロ射線による効率化

ある区画の合成射影値が 0 であることは，その区画を通る射線のうちの少なくとも 1 つについて，対応する射影値が 0 であること，つまりその射線上にはピザ饅がないことを意味している．これまでのアルゴリズムでは，この性質を，その区画が選ばれないことだけに利用していた．ここでもっと積極的に，その射線上にピザ饅がないことを利用する方法を考えてみよう．なおこの方法は，同人某氏の指摘による．

計算の途中で，ある射線に対応する射影値が 0 になったものとしよう．その場合，その射線上のすべての区画について，それを共有する他の射線の実効長が減ることになる．これをプログラムにするには，すべての区画について，それに関係する射線が分かる必要がある．この

射線 → 区画 → 射線

という計算の順序を実現するには，これまでの射影値のリストだけでは効率的に無理がある．そこで考えを新たにして，

ピザ饅が存在し得る場所

を区画対応に用意しよう．

```
  int [][] exmap = new int[r][c];
```

exmap[i][j]=1 が，区画 [i,j] に（まだ）ピザ饅があり得ることを示す．各射線の実効長は，この exmap から計算する．dband を計算する手続きだけを詳しく示す．

```
void rbandcal() { 省略 }
void cbandcal() { 省略 }
void ubandcal() { 省略 }
void dbandcal() {
  for(k=0; k<r+c-1; k++) {
    dband[k]=0;
    for(j=c-1; j>=c-1-k; j--)
      if(j>=0 && k+j-c+1<r)
        dband[k] += exmap[k+j-c+1][j];
  }
}
/* 全体計算 */
void bandcal() {
  rbandcal();
  cbandcal();
  ubandcal();
  dbandcal();
}
```

ゼロ射線が見つかったときには，その射線上の exmap をクリアし，上記の bandcal を実行する．decreaseproj の改訂版を示す．

```
void decreaseproj(int i, int j) {
    "rproj, cproj, uproj 分は省略"
  if(dproj[i-j+c-1]-- == 0) {
        // ゼロ射線検知！
    dclearmap(i-j+c-1);
        // 射線に沿って exmap をクリア
    bandcal(); // 実効長さを再計算
  }
  else dband[i-j+c-1] --;
}
```

ゼロ射線による効率化の効果は大きい．たとえば，これまでの例題について，exmap の値は初期状態の「全部 1」から，与えられた射影の値を使っただけで図 -8 のように劇的に変化する．

そして各射線の実効長も，以前は

```
rband = 8 8 8 8 8 8 8 8
cband = 8 8 8 8 8 8 8 8
uband = 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
dband = 1 2 3 4 5 6 7 8 7 6 5 4 3 2 1
```

であったものが

```
rband = 0 2 2 3 3 3 3 0
cband = 0 2 2 3 3 3 3 0
uband = 0 0 0 0 3 2 0 2 3 2 3 0 1 0 0
dband = 0 0 0 0 3 3 0 4 0 3 3 0 0 0 0
```

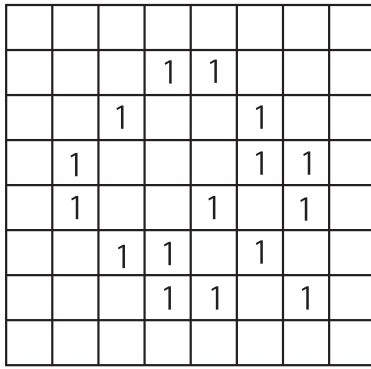


図-8　ゼロ射線による効率化

に絞られている．1回目の逆射影を示す．

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 4444 9999 0000 0000 0000
0000 0000 9999 0000 0000 6666 0000 0000
0000 4444 0000 0000 0000 0987 6666 0000
0000 9999 0000 0000 3333 0000 6666 0000
0000 0000 6666 0987 0000 4444 0000 0000
0000 0000 0000 6666 6666 0000 9999 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

（9999 は 1.0000 を示す．）

この逆射影の中で9999と表示されている場所は，「そこに確実にピザ饅がある」ことを意味している．したがって最大値選択をするまでもなく，その場所を推測区画として選んでよいことになる．同じ考えで，ある射線について，実効長と射影の値が一致した場合，exmapを頼りに「残っている区画全部を推測区画とする」ことが可能である．この「イチ射線による効率化」は読者への課題としておこう．

■虚像

ここまでのアルゴリズムを読んで，「万全だ」と思った人は楽天家である．冒頭でも触れたとおり，逆射影は「それなりの妥当性」を与えはするが，ここでのアルゴリズムで常に正しい解が得られるという保証はない．問題の1つはスプリアスと呼ばれる幻影である．処理経過の中で，正解にはない区画で合成射影値がかなり大きくなる例があった．このように，そこにはピザ饅がないにもかかわらず，四方からの逆射影の値によって，あたかもあるように見えることがある．例を示そう．

正解が図-9であるとき，射影の値は次のようになる．

```
rproj = 0 0 2 1 0 2 2 1
cproj = 0 1 2 1 1 1 1 1
uproj = 0 0 0 0 1 1 1 1 1 0 1 0 1 0 1
dproj = 0 0 0 0 0 1 1 3 0 1 1 1 0 0 0
```

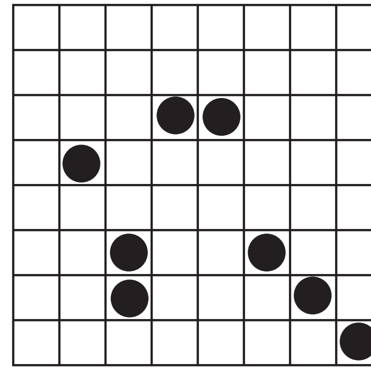


図-9　スプリアスが生じる例

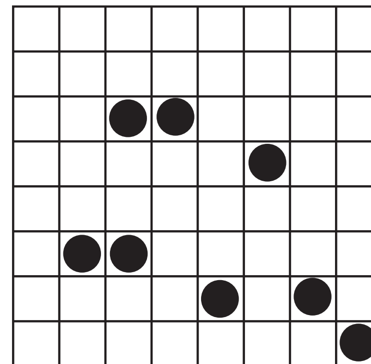


図-10　別解の存在

実はこの配置は，逆射影が区画[2,2]に集中するようにしたものである．そこにはピザ饅は，ない．

```
---- 逆射影（1 回目）----
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 1333 0833 0246 0000 0000 0000
0000 0156 0000 0125 0208 0092 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0222 1333 0083 0000 0266 0000 0222
0000 0000 0493 0000 0185 0000 1333 0000
0000 0000 0000 0092 0000 0092 0000 1000
```

目論見通り区画[2,2]にスプリアスが現れる．すなわち，正解とは外れた結果を与えてしまっている．これはアルゴリズムに不具合があるからであろうか．試みにこの例を最後まで計算すると，結果として図-10の解が「何事もなかったように」得られる．そしてこの結果は，入力として与えられた射影をすべて満たすのである．

与えられる入力は射影だけなので，この「結果」が間違いであると言うことはできない．つまり，ここでの入力には解が複数存在することになる．射影によって失われる情報により，複数の配置が同じ射影を与えるからである．コンテストのもとの問題では，解が

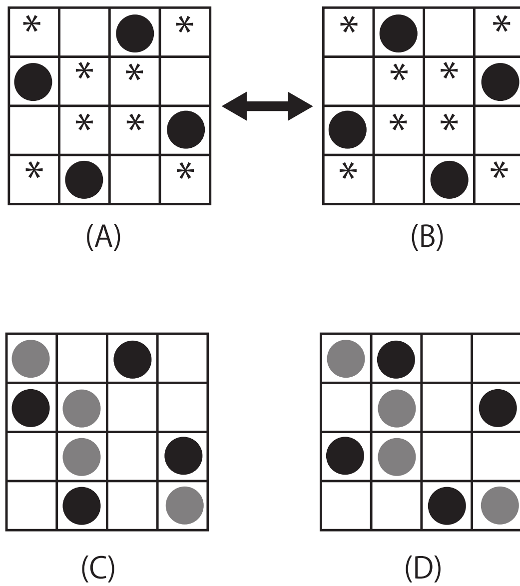


図-11　4×4配置における射影同値核

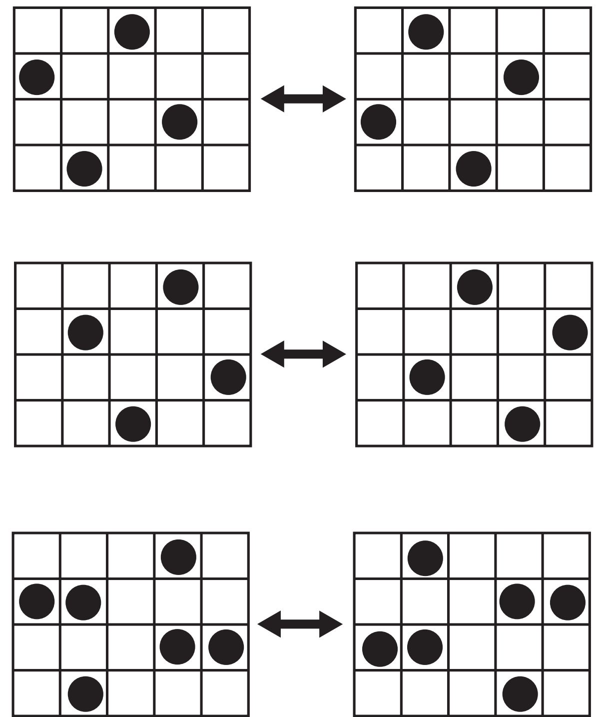


図-12　4×5配置における射影同値核

1つに決まるように入力データが選ばれたそうである．こうすれば，判定が容易になるからであろう．

■射影同値

同一の射影から複数の「解」が得られる場合，これらの解を互いに**射影同値**であると呼ぶことにしよう．射影同値であるための条件を探るために，小さなピザボックスでの実験を行ったところ，射影同値核とでも呼べる特定の配置があることが判明した．

図-11の左側(A)の配置については，＊印で示した8カ所の状況がどうであろうとも，右側(B)の配置にその8カ所の状況をそのまま複写した配置と射影同値になる．たとえば(C)と(D)は射影同値である．4×4の場合はこれ以外に射影同値である配置はない．4×5の場合のすべての射影同値核を図-12に示す．

■アルゴリズムの性質

スプリアスに起因した問題は，「別解」，そして射影同値性という新たな状況を生じさせた．もう少しこのアルゴリズムの性質を調べておこう．

◎射影の減らし過ぎ問題

計算では射影の値を順々に減らしてゆくが，0なのにさらに減らしてしまうことはないだろうか．これが起こらないことは，$F(X)$の作り方から保証される．区画Xを通る射線のどれか1つでも射影値が0であれば，$F(X)=0$となり，最大値選択で選ばれることがないからである．言い替えれば，射影の値が0の射線上に区画を推測することはない．

◎射影の不整合問題

たとえばrproj[2]とdproj[12]は共通の区画を持たないから，

```
rproj = 0 0 1 0 0 0 0 0
dproj = 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
```

という状態は解を持たない「不整合な」状態である．アルゴリズムを実行しているうちに，このような不整合な状態になる恐れはないだろうか(図-13)．

ある段階での射影が整合的であり，その次の段階で上記のような不整合状態になったものと仮定しよう．すると，直前の射影が整合的であることから，上記のrproj[2]と共通区画を持つdproj（たとえばdproj[4]）と，上記のdproj[12]と共通区画を持つrproj（たとえばrproj[5]）の両方に射影がなければならない．

```
rproj = 0 0 1 0 0 1 0 0
dproj = 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0
```

この状態では，rproj[2]とdproj[4]の共有区画[2,5]と，rproj[5]とdproj[12]共有区画[5,1]の2つの区画に，それぞれピザ饅が1つずつ存在することによって，整合的になっていたことになる．

そしてこの[2,5]あるいは[5,1]のどちらについても，

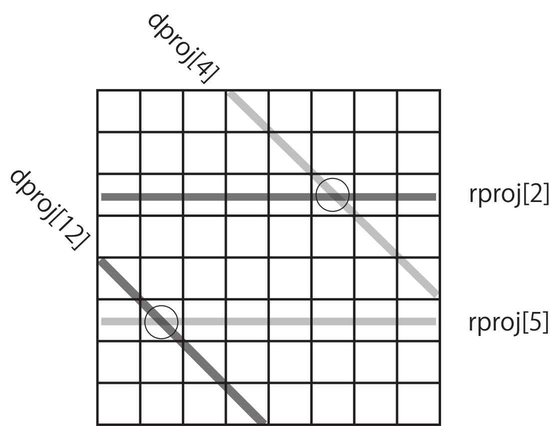


図-13　不整合な射影状態

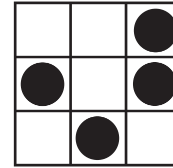


図-14　不適切な推測の例

推測の結果として不整合な状態になることはない．

■アルゴリズムの正しさは結局？

これまでの議論は，いささか状況証拠的であることは否めない．射影がどういう場合に整合的であり，どういう場合にそうでないかについても，きちんと示されているわけではない．合成射影値の大きい順に選択してゆく方法がたいていの場合うまくゆくらしい，というだけの話である．ちなみに，3×3，3×4，4×4，4×5のそれぞれのピザボックス場合には，ここで示したアルゴリズムは射影同値の範囲内で正しい解を与える．よりきちんとしたアルゴリズムにするには，射影が整合的であるか否か，すなわちその射影を与える配置が存在するかどうかの判定方法を求め，整合的でなくなった場合には推測をバックトラックするプログラムにする必要がある．

最後に，合成射影値が0ではない区画を選択することによって，不整合状態に陥る例を示しておこう．

図-14の配置では

```
rproj = 1 2 1
rband = 2 2 1
cproj = 1 1 2
cband = 1 2 2
uproj = 0 1 1 2 0
uband = 0 2 1 2 0
dproj = 1 1 0 2 0
dband = 1 2 0 2 0
```

であり，区画[0,1]（上辺中央）の合成射影値は，他の●がある区画での値よりもかなり小さい（最小値）が0ではない．仮にこの[0,1]を選んだとすると

```
rproj = 0 2 1
rband = 0 1 0
cproj = 1 0 2
cband = 0 0 1
uproj = 0 0 1 2 0
uband = 0 0 0 1 0
dproj = 1 0 0 2 0
dband = 0 1 0 0 0
```

となる．この状況，たとえば rband[1] < rproj[1] は，推測可能な区画が不足していることを意味しており，この射影を満足する配置が存在しないことを示している．

バックトラックによる完全なアルゴリズムは，今後の課題としたい．

（平成17年2月23日受付）