

## 箱詰めパズルのプログラムについて

東大 理学部 川合 慧

## 1. はじめに

箱詰めパズル (packing puzzle) とは、一定の形状をした箱 (board) に、それをちょうど埋めつくすだけの面積 (あるいは体積) の船を持つ小片 (駒, piece) の集合を、互に重ならないように収容するという問題である。この種の組合せ問題を計算機を使用して解く場合には、賢明な人間が行なうようなしるみつぶし法を、計算機に物言わせて実行するのが一般的である。その際、箱の中で各 piece が単独にとることのできる位置の集合をまず求め、その直積集合を作り、そのすべての要素 (piece の配置法) について piece の重なりがあるかどうかを調べることによって、この問題は原理的に解くことができる。ところが、本文で述べる Tetraball パズル (piece 数 11) に例をとってみても、この直積集合の要素数は約  $10^{22}$  と非常に大きい。一年が約  $3 \times 10^{13}$   $\mu$  秒

しかないことから考えてもわかるように、このようなしらみつぶし法は、数学的（原理的）解法ではあっても実際の解法とはなり得ない。そこで、逐次的なはめこみを行ない、失敗したら一段階戻るといふ、backtracking の手法を用いることになる。これは、piece の重なり状態に関する何らかの強力な定理が発見されない限り、解を見出す、もしくは数え上げるための唯一の方法である。

Tetraball パズルについては [1] において、パズルの記述と数え上げプログラムの実際、および効率についてのいくつかの考察を発表したが、今回は、探索プログラムの振舞といくつかの実験結果、および対称解の除去方法について述べる。

## 2. Tetraball のプログラム

Tetraball の解の数え上げプログラムについてまず簡単に述べる。プログラムとパズルの詳細については [1] を参照のこと。

(i) Tetraball パズル。各 piece は単位球を 4 個、tetrahedron および tetromino の形式で平面状につないだもので、11 種類ある。board は単位球 44 個を正八面体状に組上げた形。

(ii) Piece のデータ。各 piece について、それが位置し得

る場所 (site) の表を作る。各 site は piece の要素 (unit)<sup>†</sup> が位置する場所 (cell)<sup>†</sup> の番号の組で表わされる。

(iii) 探索順序。 Backtracking を行なう場合、各段階における目標は、一定の規準により決定される一つの空 cell (target-cell) を埋めることである。未使用 piece の site で、この target cell を含むものがすべて試される。target cell の決定法としては、最も回りを囲まれている cell、すなわち、最も袋小路に近い cell をとる方法 (Most Closely Surrounded Cell: MCSC<sup>†</sup> 法) と、最も番号の若い cell をとる方法 (Youngest Empty Cell: YEC<sup>†</sup> 法) の 2 つを採用した。

(iv) 実行結果。 TOSBAC-40 (加算時間 5.6  $\mu$  秒) において、MCSC 法で 120 時間、YEC 法で 27 時間を費して数え上げを行なった。両方法とも、解の総数として同じ数、7,482 が得られた。これにより、解の信頼性はかなり高いものと思つてよいであらう。

### 3. 探索プログラムの振舞

上で述べたように、MCSC 法と YEC 法では総探索時間に大きな開きがある。この原因を明らかにする目的で、探索プロ

<sup>†</sup> [1] では箱を frame, piece の要素を element, frame の要素を box と叫んだが、以後これを board, unit および cell と呼ぶことにする。

グラムが実際にどのように動いているかを計測した。その方法の概略も下に示す。このプログラムは、いくつかの *piece* がすでにはめ込まれている状態をパラメータとして与えられ、それを含むすべての解を数え上げる。

```

proc search(n); integer n;
  begin determine target cell;
    for all unused piece do
      for all site of piece
        whichincludes target cell do
          begin D1[n] := D1[n] + 1;
            if locatable
              then begin D2[n] := D2[n] + 1;
                      locate;
                      search(n+1);
                      remove
                    end
              end
          end
        end
      end
    end
  end
end

```

D1 はこの手続きの呼び出し一回につき試される *site* の数、D2 は同じく成功した *site* の数を数える。もちろん、最終結果はこれらの総和を表わすことになる。計測した結果は表 1 の通りである。また、その大体の様子を図 1 に示す。

まず第一に気付くことは、D1 のグラフの形状が、MCSC 法と YEC 法でほとんど同じだということである。また、総探索回数もほぼ等しい（YEC 法の方が約 4% だけ多い）。このことから、単位操作の回数に関して見る限り、この 2 方

表-1 探索回数

| n    | D1          | D2         | D1          | D2         |
|------|-------------|------------|-------------|------------|
| 2    | (MCSC) 372  | 182        | (YEC) 372   | 286        |
| 3    | 15,043      | 4,423      | 24,253      | 10,152     |
| 4    | 337,772     | 65,134     | 585,310     | 193,778    |
| 5    | 4,485,241   | 579,577    | 8,597,038   | 1,891,952  |
| 6    | 33,782,113  | 3,004,889  | 62,084,829  | 10,450,954 |
| 7    | 142,719,435 | 8,480,673  | 204,212,363 | 29,253,669 |
| 8    | 310,114,084 | 11,653,097 | 352,338,894 | 33,291,452 |
| 9    | 296,389,355 | 6,392,729  | 249,284,038 | 17,580,934 |
| 10   | 96,855,877  | 979,897    | 50,815,744  | 2,226,013  |
| 11   | 6,150,874   | 7,482      | 1,762,973   | 7,482      |
| SUM. | 890,850,166 |            | 929,705,814 |            |

法はほぼ同等であるということが言える。したがって、4倍以上という計算時間の差は、MCSCを決定するための手間に归せられる。pieceを置くのに必要な操作は、YEC法では site で示される4つの cell に piece 名を書き込むことだけであるが、MCSC法ではこれに加えて、このまわりの空 cell の「囲まれ数」を増加させる操作が必要となるからである。MCSC法は「制限

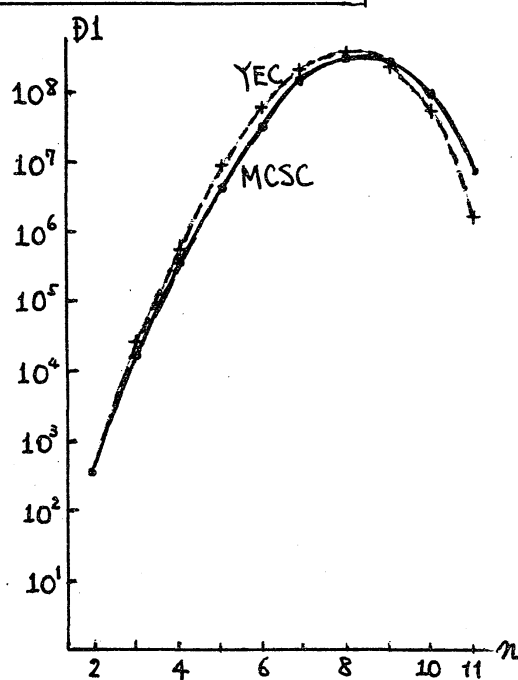


図1. 試行 site 数

の強い場所から試す」という，backtrackingにおける効率化条件を満たすべく採用したが，本パズルでは，ほぼ同じ効率を持つ YEC 法と較べて overhead が大きすぎたようである。

つぎに特徴的なこととして，探索過程が圧倒的に  $n=7, 8, 9$  の所に集まっていることがあげられる。実際， $D1[2] + \dots + D1[6]$  は MCSC 法では全体の 4.3%，YEC 法では 7.7% にすぎないのに対し， $D1[7] + D1[8] + D1[9]$  は 84.1% (MCSC) 及び 86.7% (YEC) を占めている。探索回数に関するこのグラフの形状については，データが少いので確定的なことは言えないが，一般的にほぼ似たような状況にあると思われる。これは，探索効率を高めるための枝刈り操作を導入するに当って，考えに入れておかなければならない事実である。すなわち，出現頻度の小さい段階（tetraball では piece 数が 6 以下の段階）で精力的に枝刈りを行えば，その手間が少々大きいものであっても，全体に対する寄与が大となる可能性が大きいのである。MCSC 法では，「囲まれ数」操作の副産物として，piece を置いたことにより生ずる孤立した空 cell の存在を検知することが出来る。実際のプログラムではこれを枝刈りに用いているが，上述と同様の計測を行なった結果，これが有効となるのは piece 数が 8 以上の段階であることがわかった。したがって，この check が実行時間の短縮にほとん

ど寄与してはいないのは明らかである。

前回, [1]において探索効率の見積りを行なったが, ここでは次の表式を導いた。

$$\alpha_n = S \left(1 - \frac{n}{N}\right) \quad \text{----- (*)}$$

$$\beta_n = S \left(1 - \frac{n}{N}\right)^M \quad \text{----- (**)}$$

ここで  $\alpha_n, \beta_n$  は piece が  $n$  個おかれている一つの配置に対して, 試される site の数と成功した site の数,  $S$  は piece 当りの平均 site 数,  $N$  は piece 数,  $M$  は piece の大きさである。ここでは, すでに置かれている  $n$  個の piece と, これから試す piece の各 unit が, board 中に一様に存在するものという, きわめて粗い近似を用いている。この  $\alpha_n$  と  $\beta_n$  に対応するものとして, つぎの量を計算した。

(i)  $D1[n]/D2[n-1]$

これは, 平均的には  $\alpha_n$  に一致する量と考えられる。MCSC 法の場合にはかなりきれいな直線となっており,  $\alpha_n$  の式の形とよく合っている。YEC については, 一様分布近似が成り立たないことがわかる。このグラフ

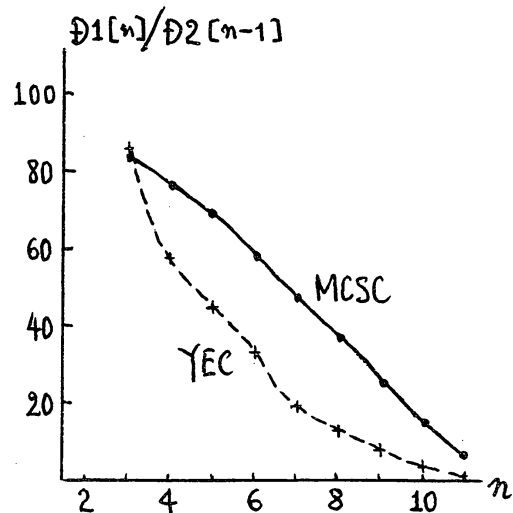


図2. 一段あたりの平均試行 site 数。

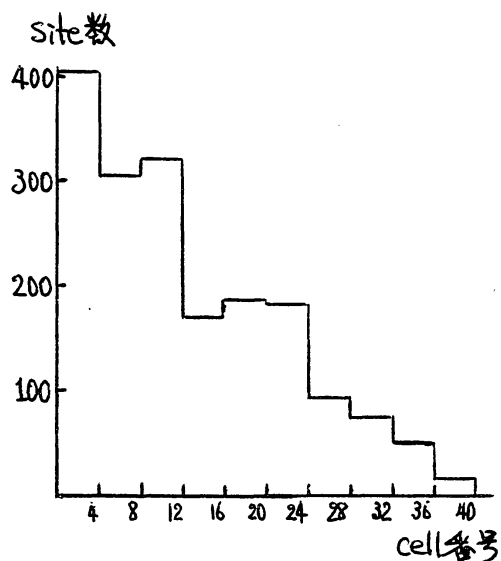


図3. 先頭 cell番号による site数の分布

(ii)  $[\text{D2}[n]/\text{D1}[n]]^{1/3}$

これは前述の  $(**) \div (*)^{1/3}$  に対応し、上の表式では  $1 - \frac{n}{N}$  に対応する。直線的な部分が長いので、非常に荒い近似では  $(*)$ ,  $(**)$  が成立しているとも言えるが、(i) で述べたように YEC では  $(*)$  が成立していないので、この図からだけでは結論はでない。

図1は、本プログラムが扱った search tree の各レベルの mode 数と見ることができ、D.E. Knuth が [2] において、この形状および全体の mode 数の見積り法について述べている。

の形は、図3に示す先頭 cell番号による site数分布の形によって説明される。YEC 法において、置かれている piece数と YEC の番号は必ずしも比例するわけではないが、おおよその形状は一致するものと思われる。

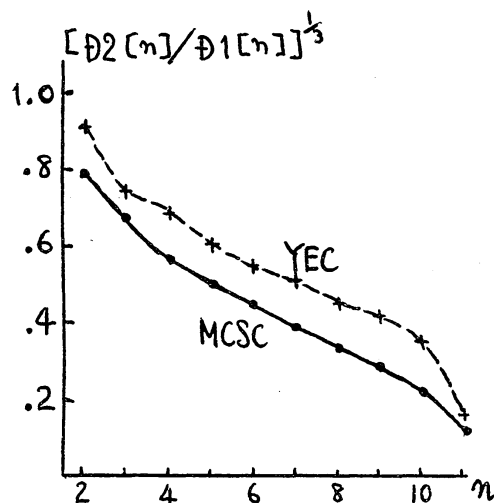


図4. 一段あたりの平均 試行成功率 (各 unitについて)

#### 4. 対称解の取り扱いについて

箱詰めパズルでは，回転や反転でお互に移り変り得る解は，まとめて1個と数えられるのが普通である。これを取り扱うには，特定の piece の置き場所を制限するのが一般的なやり方であるが，場合によっては複数列の piece を考慮に入れる必要が生ずることもある。ここでは，対称解の間の変換が，board をそれ自身に移す対称変換に限られることに着目した取扱いを述べる。

##### 4-1 site と配置に対する変換

まず，board の対称変換群を

$$G = \{g_0, g_1, \dots, g_n\} \quad \text{-----}(1)$$

とする。 $g_0$  は恒等変換である。各  $g_i$  は cell に対する置換の一つである。つぎに，piece  $P$  の  $j$  番目の site を

$$S_j^P = \{c_{j1}^P, \dots, c_{jM}^P\} \quad \text{-----}(2)$$

で表す。 $c$  は cell の番号， $M$  は piece の大きさである。また

$$S_0^P = \{0, 0, \dots, 0\} \quad \text{-----}(3)$$

を piece  $P$  が未使用であることを示す site として導入しておく。これにしたがって， $G$  の要素  $g_i$  は  $\{0\}$  を  $\{0\}$  に， $\{1, \dots, V\}$  を  $\{1, \dots, V\}$  に移すものと拡張しておく。

site に対する変換を，要素 cell に対する変換の結果をまとめた  $M$ -組であると定義すれば，つぎの性質が存在する。

$$\forall g \in G, \forall p \forall j : \exists k, g S_j^p = S_k^p \quad \text{-----}(4)$$

任意の piece 配置は、つぎのように表わされる。

$$T = (S_{j_1}^1, S_{j_2}^2, \dots, S_{j_N}^N) \quad \text{-----}(5)$$

$N$  は piece 数である。つぎに、site と配置も不変に保つ変換の集合を定義する。

$$h(S_j^p) = \{g \in G \mid g S_j^p = S_j^p\} \quad \text{-----}(6)$$

$$H(T) = \{g \in G \mid g T = T\} \quad \text{-----}(7)$$

ただし、配置に対する変換は、各 site も変換したものの  $N$ -組である。  $T_0 = (S_0^1, \dots, S_0^N)$ ,  $T = (S_{j_1}^1, \dots, S_{j_N}^N)$  とすると

$$H(T_0) = G \quad \text{-----}(8)$$

$$H(T) = \bigcap_{p=1}^N h(S_{j_p}^p) \quad \text{-----}(9)$$

が成り立つ。

#### 4-2 対称解の除去

今、2つの解  $R_1$  と  $R_2$  ( $R_1 \neq R_2$ ) が

$$R_2 = g R_1$$

なる関係にあったとする。数え上げ過程でどちらか1つだけも取扱うためには、両者を区別するものが必要であるが、それは明らかに

$$g \notin h(S_j^p)$$

を満たす  $R_1$  中の site である（一般に複数個存在する）。

backtracking で  $T_0$  から  $R_1$  を得るには,  $T_0$  中の各  $S_0^p$  を (ある順番で)  $S_j^p$  ( $j \neq 0$ ) に置き換えてゆくわけであるが,

$$g \in H(T_0), \quad g \notin H(R_1)$$

であるから, はじめに対称性  $g$  もこわす site を抜いた時に, それに  $g$  をほどこした site に印をつけ, それに関する探索は禁止するようにすれば, 対称解  $R_2$  は数えられることになり。これが対称解除法の原理である。

#### 4-3 実際の操作法

Tetraball パズルでは  $G$  の要素は 24 個あるが, "J" と呼ばれる piece のすべての site に

ついて  $h(s) = \{g_0\}$  が成り立っている。したがって "J" の site (144 個) を  $G$  で分割したものの代表 (6 個) のみを扱うことにより, 対称解も完全に除くことができる。

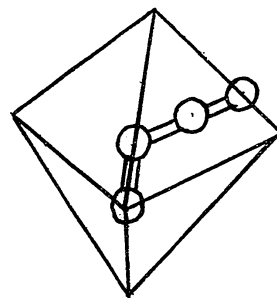


図5. Tetraballの board と piece "J"

しかし, 問題によってはこのような性質を持つ piece が存在しないこともある。実際, pentocube パズルにおいて, 4 個の piece を置いても対称性がくづれない例が [1] に報告されている。

対称解の除去をプログラムで行なう方法としては, 各 site 間の  $G$  による関係をあらかじめ全部計算しておく方法と,

backtracking を進めながら実行時に処理する方法とが考えられるが、実行時に行なうのがプログラムが明解であり、3. で述べた考察によりそれほど効率は落ちないと予測される。概略のプログラムを下に示す。

```

proc prelocating(piece); integer piece;
  for all site of piece do
    if not mark[piece, site] and locatable
      then begin locate;
               marking; change symmetry;
               if yet symmetric
                 then prelocating(piece + 1);
                 else search;
               recover symmetry;
               remove
            end
  end

```

marking は、site  $S_{site}^{piece}$  に  $G$  の要素を作用させ、結果の site に mark を付ける。主プログラムは prelocating(1) を呼ぶ。prelocating は、対称性がなくなるまで自分自身を呼び続け、それから search を呼ぶ。対称性が残っている間は試行 piece の順番は固定されている。

この方式のプログラム実験はまだやっていないが、この方法が確立すれば、プログラミングの中核は

(i) piece と変換  $G$  の表現

(ii) site 表の作成

の2点になり、速度の記録を競う場合以外の標準的手法となる。

う。

## 5. まとめ

当初, “絶体的高能率アルゴリズム” であると思って採用した MCSC 法が, そほどの成果を上げなかったこと, および, 1974 年 3 月の研究会で通研の竹内氏が話された「10 億回」という数字に興味を持ったことから, 総試行回数を計ってみた。その結果は以上に述べた通りであるが, その他に, 竹内氏のプログラム [1] が扱った探索 tree と Tetra ball のとれとが, ほぼ同じサイズであることも判明した (表 1 参照)。このあたりが, 計算機で数時間程度で数え上げが可能な問題の標準的大きさである。これから考えて, 計算機による数え上げが (現実問題として) 可能な最大のサイズは約  $10^{12}$  程度であると思われる。

## 参考文献

- [1] “ 計算機によるゲーム・パズルの具体化の検討, ”  
京都大学数理解析研・講究録 217 (1974)
- [2] D.E.Knuth, “ Estimating the Efficiency of  
Backtrack Programs ”, Mathematics of Computation,  
vol/29, no 129, Jan. (1975) pp. 121-136.