高崎経済大学論集　第49巻　第３・４合併号　2007　83頁～92頁

役割分担の重複がもたらす問題に関わるコンピュータ・シミュレーション

藤　本　　哲

A computer simulation on a problem caused by overlaps of role assignment.

Fujimoto Tetsu

Summary

A computer simulation model on a problem caused by overlaps of role assignment was made. This model had 3 member, 3 roles, and was represented by matrices. And overlaps of role assignment were also represented by a matrix. The increase and decrease of effective energy used in single garbage can model program was included in each member of this model. The state of taraimawasi of the case of including each member effective energy was worse than not including.

序

組織構造の集約的次元は三つあり、それらは複雑性（complexity）、公式化（formalization）、集権化（centralization）である（Hage、1965；Hall、1996；野中ほか、1978）。それらをどのように記述するかについて、これまで考察してきた（藤本、1998a、1998b、2001a、2001b、2002）。そのうち複雑性次元について考察した藤本（2001b）は、調査対象組織の性質により、分化した組織の調整を取り扱うようなデータを見いだすことが出来ず、結果として不調に終わった。

複雑性という概念の要素は、水平分化（horizontal differentiation）、垂直分化（vertical or hierarchical differentiation）、地理的分散（spatial dispersion）の３つである（Hall、1996）。水平分化は専門分化と作業分割の二つに分けられ（Hall、1996）、専門分化の測定尺度としては職種（あるいは仕事）の数（Hage、1965；Hall、1996）、訓練期間の長さ（Hage、1965）が用いられてきた。そして垂直分化の程度を知る測定尺度としては、階層が何段階あるかが用いられてきたが、この測定尺度は権威が階級に応じて配分されているという前提を要する（Hall、1996）。さらに地理的分散の測定尺度としては、組織が持つ事務所や工場の数あるいは本社から離れて働く人数

（Hall、1996）が尺度として用いられてきた。組織構造の複雑性の程度が高まれば、分化の程度が高まっていく。

上記の分化および分散のうち専門分化あるいは作業分割を、どのように表現するかについて示唆を与えてくれるのが Cohen ら（1972）における組織構造の表現である。そこではアクセス構造（access structure）および決定構造（decision structure）が行列によって表されている。アクセス構造とは問題と選択との間の関係であり、決定構造とは意思決定者と選択との間の関係である。構造の行列表現には、無分節決定（unsegmented decisions）、階層的決定（hierarchical decisions）、専門的決定（specialized decisions）の三つの純粋型がある（Cohen ら、1972）。無分節決定は全ての成員が全種類の決定に対応可能であることを表す。階層的決定はより上位の成員になるほど対応可能な意志決定の種類が多くなることを表す。専門的決定は特定の成員は特定の種類の決定にのみ対応可能であることを表す。図１にその行列行現を引用するが、一見して分かる通り、こうしたやり方を使えば、専門分化あるいは作業分割（以下役割分担）をモデルとして表すことが出来そうである。

1111111111	1111111111	1000000000
1111111111	0111111111	0100000000
1111111111	0011111111	0010000000
1111111111	0001111111	0001000000
1111111111	0000111111	0000100000
1111111111	0000011111	0000010000
1111111111	0000001111	0000001000
1111111111	0000000111	0000000100
1111111111	0000000011	0000000010
1111111111	0000000001	0000000001
無分節決定	**階層的決定**	**専門的決定**

図１　決定構造の三つの純粋型（Cohen et al., 1972, pp.6-7）

組織における曖昧さについての論考はこれまでに多くあるが、代表的モデルはゴミ箱モデル（Cohen ら、1972）であろう。このモデルは、Simon の意思決定理論のようにはいかない、現実の意思決定を表そうとしている。問題と解と決定の間に関係がなく流動的で、参加者も流動的で、なぜか決定が行われたり流れたりする。曖昧な状況を表そうとしている。Cohen ら（1972）では FORTRAN によるプログラムを使ってシミュレーションが行われた。

高橋（1993）は、Cohen ら（1972）のゴミ箱モデルを元に、単純ゴミ箱モデル・プログラム（Single Garbage Can Program）を作成した。その理由として２つを挙げている。第一に、Cohen ら（1972）のプログラムは、決定構造やアクセス構造を組み込んでいるために複雑で長いものになっており、そのためにプログラムの内容を理解するのが難しく、修正が難しい。第二に、乱数列をデータとして与えているために、長い期間のシミュレーションには適さない。これらの問題点に対処するために、高橋（1993）では次のように定式化した。第一に、一つの選択機会にのみ注目する

（即ち single garbage can）。第二に、参加者の出現・退出を、参加者からのそれぞれ正負のエネルギーで表す。第三に、問題の出現・退出を、それぞれ正負のエネルギー必要量で表す。以上の方針に基づいて、BASIC によるプログラムが作成され、シミュレーションが実施された。

役割分担の曖昧さが引き起こす様々な問題の１つに、たらい回しがある。たらい回しを国語辞典で引くと「なれあいで、権利・地位などを次つぎに他に受け渡しすること」（『新明解国語辞典』第三版）とあるが、官僚制組織や専門の細分化がもたらす弊害の１つとして言及されることもある。例えば、救急患者搬送のたらい回し（NHK プロジェクト X 制作班編、2003）は、かつては頻発していたようである。また私自身も、旧日本育英会（現日本学生支援機構）の書類に関して、庶務課と学生課の間で簡単なたらい回しに遭った。奨学金の在職届の手続きをしてもらうために、庶務課へ行くと、奨学金に関することだから学生課へ行ってくれと言われた。学生課へ行くと、教員のことだから庶務課だと言われた。もう一度庶務課へ行くと、先程は思い違いをしていて、確かに庶務課の担当ですと言われた。上記の２つの例を見ても、官僚制組織や専門の細分化がもたらすたらい回しには、幾つかの種類がありそうなことが分かる。

私の経験した簡単な書類のたらい回しは、担当者が自分の職務範囲を一瞬勘違いしてしまったことから発生した。ここで役割分担が曖昧だったらどうなるだろうか考えてみる。２つの部署で役割範囲が一部重なっているとすると、どのような現象が起きるのであろうか。連帯責任は無責任という諺がある。特定の職務を担当するのが一人だけならその人は責任を持って仕事を担うだろうが、複数いれば誰か他の人がやってくれるだろうと考えて自分ではやらなくなってしまう、というたらい回しが起きるだろう。ここに関わる要因としては例えば、忙しさあるいはやる気があるのだろう。

ここで、役割分担の曖昧さを、決定構造の表記方式で、表現してみる。先ずは単純な状態から考えていくため、成員数３名、職務の種類３種で作ってみる。その前に、案件を定めてみる。図２は案件を表現するための行列である。案件は３種類あり、各々の行列がその性質を表している。例えば第１行（100）は一番目の種類の性質を持つ案件であり、第２行（010）は二番目の性質を持つ案件であることを表す。それに対し能力分布構造は図３のように定めてみる。第１行は成員１の、第２行は成員２の、第３行は成員３の有する能力を表している。例えば成員１は、第２種および第３種の能力を有すことを、第１行（011）は表している。

```
100
010
001
```

図２　案件行列の例

```
011
110
101
```

図３　能力分布構造（行列）の例

方　法

藤本（2006）と同様に Perl でプログラムを作り（表１）、シミュレーションを行う。先ずサブルーチンsettei01においては、仕事の種類は３種類として案件行列を作り、組織成員の数は３種類として能力分布行列（担当範囲分布行列）を作る。次にサブルーチン anken_sentaku において、案件行列から一つの案件つまり行を無作為に選ぶ。

表１　シミュレーションを行うプログラム（taraimawasi.pl）

```
#!/usr/bin/perl

sub settei01{
      #仕事の種類の数は
      $m=3;#列の数になる

      #案件行列は対角行列とする。値は0か1とする。
      $a[0][0]=1;$a[0][1]=0;$a[0][2]=0;
      $a[1][0]=0;$a[1][1]=1;$a[1][2]=0;
      $a[2][0]=0;$a[2][1]=0;$a[2][2]=1;
      $num_a = 3;#案件の数、行列aの行数

      #能力構造skill structure
      $ss[0][0]=0;$ss[0][1]=1;$ss[0][2]=1;
      $ss[1][0]=1;$ss[1][1]=1;$ss[1][2]=0;
      $ss[2][0]=1;$ss[2][1]=0;$ss[2][2]=1;
      #組織成員の数、行列ssの行数
      $n = 3;

      #列の数を指定する変数を用意する。scalabilityのため
      $e = $m - 1 + 1;#$ss[$tn][$e]をeeetに割り当てる
      $r = $m - 1 + 2;#その都度発生させた乱数を入れておくため。$ss[$tn][$r]
      $y = $m - 1 + 3;#$ss[$tn][$y]をやる気フラグに割り当てる
}

sub test_print {
#     print "anken gyoretu¥n";
#     print "$a[0][0]¥t$a[0][1]¥t$a[0][2]¥n";
#     print "$a[1][0]¥t$a[1][1]¥t$a[1][2]¥n";
#     print "$a[2][0]¥t$a[2][1]¥t$a[2][2]¥n";
#     print "skill structure¥n";
#     print "$ss[0][0]¥t$ss[0][1]¥t$ss[0][2]¥n";
#     print "$ss[1][0]¥t$ss[1][1]¥t$ss[1][2]¥n";
#     print "$ss[2][0]¥t$ss[2][1]¥t$ss[2][2]¥n";
#     print "erabareta anken¥t";
      print "$a[$gyou][0]¥t$a[$gyou][1]¥t$a[$gyou][2]¥n";
#     print "tantosya¥t";
      print "$ss[$tn][0]¥t$ss[$tn][1]¥t$ss[$tn][2]¥n";
      print "matching¥t¥t$match¥n";
#     print "eeet¥n";
#     print "$ss[0][$r]¥t$ss[1][$r]¥t$ss[2][$r]¥n";
#     print "$ss[0][$e]¥t$ss[1][$e]¥t$ss[2][$e]¥n";
#     print "$ss[0][$y]¥t$ss[1][$y]¥t$ss[2][$y]¥n";
#     print "¥n";
}

sub saikoro{#0から引数の値（最大値）までのさいころ
      #参考http://www.rfs.jp/sb/perl/02/04.html
```

```
	my($max,@array,$index);
	($max) = @_;#左辺は括弧で括らないと駄目
	@array = 0..($max);
	$index = rand @array;#@arrayはscalar contextで評価される
	$array[$index];#戻り値になる
}

sub anken_sentaku {
	&saikoro($num_a - 1);#戻り値になる
}

sub tantosya_erabi {
	&saikoro($n - 1);#戻り値になる
}

sub eeet {
	#effective energy of each tantosya
	my($upperlimit,$lowerlimit) = (1,-1);#eeetの上限と下限
	foreach (0..($n-1)) {
		$ss[$_][$r] = (rand(1) - 0.5);
		$ss[$_][$e] += $ss[$_][$r];
	}
	#eeetが上限と下限を超えたら丸める
	foreach (0..($n-1)) {
		if ($ss[$_][$e] > $upperlimit) {
			$ss[$_][$e] = $upperlimit;
		}
		if ($ss[$_][$e] < $lowerlimit) {
			$ss[$_][$e] = $lowerlimit;
		}
	}
	#やる気の有無を判別
	foreach (0..($n-1)) {
		if ($ss[$_][$e] > 0) { #やる気（一応）あり
			$ss[$_][$y] = 1;
		} else {                    #やる気なし
			$ss[$_][$y] = 0;
		}
	}
}

sub a_t_matching {
	#選ばれた案件と選ばれた担当者との適合を調べる
	my($match) = 0;
	foreach (0..($m-1)) {
		$match += ($a[$gyou][$_] * $ss[$tn][$_]);
	}
	$match;
}

sub a_t_y_matching {
	#選ばれた案件と選ばれた担当者とやる気との適合を調べる
	my($match) = 0;
	foreach (0..($m-1)) {
		$match += ($a[$gyou][$_] * $ss[$tn][$_]);
	}
	$match *= $ss[$tn][$y];#たらい回しが起きたら0、起きなくて担当者が処理したら1となる
	$match;
}

sub match_statistics {
	foreach (@match_results) {
		$match_stat[$_]++;
	}
	#$match_stat[0]はたらい回しが起きた回数、$match_stat[1]は案件が処理された回数
```

```
	#print "match_stat¥t(0)$match_stat[0],(1)$match_stat[1]¥n";
	print "$match_stat[0] $match_stat[1] ";#第1フィールド（添え字は0）と第2フィールド
（添え字は1）

	my($count,$temp) = (0,0);
	while (@match_results) {
		$temp = shift(@match_results);
		if ($temp == 0){#たらい回しが起きた場合、連続する場合
			$count++;
		}else{#$temp == 1 たらい回しが起こらず、担当者が対応した場合
			$mawashi[$count]++;#たらい回し連続$count回の所をincrement
			$count = 0;
		}
	}
	foreach (@mawashi){#配列のうち、値の無い所に0を埋める
		if ($_ == undef){
			$_ = 0;
		}
	}
	print "@mawashi¥n";#第3フィールド（添え字は2）以降
}

#以下main
foreach (1..10000) {
	&settei01;
	$gyou = &anken_sentaku;#選ばれた案件の行number

	$match = 0;
	while ($match == 0) {
		&eeet;
		$tn = &tantosya_erabi;#tantosya number
		$match = &a_t_y_matching;
		#$match = &a_t_matching;
		#&test_print;
		push(@match_results,$match);#match results
	}
}
#print "@match_results¥n";
&match_statistics;
```

次にサブルーチン eeet においては、高橋（1993）を参考に、先ず乱数を発生させて－0.5から0.5の範囲に変換して各成員の effective energy の値に加えて行く。その加えた値が上限と下限を超えたら上限値と下限値にそれぞれ丸める。その結果が負の値ならやる気無しと、正の値ならやる気有りと定め、やる気無しなら０を、やる気有りなら１を、各成員のフラグ（やる気フラグ）に代入する。

更にサブルーチン tantosya_erabi において、能力分布行列から一人の成員つまり行を無作為に選ぶ。

そしてサブルーチン a_t_y_matching においては、選ばれた案件と選ばれた担当者とやる気との適合を調べる。選ばれた案件行の要素と、選ばれた担当者行の要素とをそれぞれ掛け算して足して行くと、その値は０か１になる。０ならば選ばれた担当者は選ばれた案件を処理する能力を持っていないことになり、１ならば選ばれた担当者は選ばれた案件を処理する能力を持っていることになる。その後、選ばれた担当者のやる気と掛け算すると、その値は０か１になる。０ならば選ばれた担当者は選ばれた案件を処理しないし（たらい回しが起きた）、１ならば選ばれた担当者は選ばれた案件を処理する（たらい回しは起きなかった）ことになる。この結果を、配列 match_results に

スタックする。

また while 文の条件節「＄match ＝＝ 0」は、たらい回しが起きた場合には、このループを繰り返させる。そして main ルーチンの foreach 文の条件節に書かれている回数だけ案件が処理されるまで繰り返される（たらい回しの回数は含まれない）。

最後にサブルーチン match_statistics では、先ずたらい回しが起きた回数と処理された件数を勘定する。次にたらい回しが連続で起きた場合に、X 回連続が何回発生したかを調べる。例えば、たらい回し 0 回で処理されたのが何回か、たらい回し 1 回で処理されたのが何回か、たらい回し連続 2 回で処理されたのが何回か、たらい回し連続 3 回で処理されたのが何回か、等々、という風に勘定する。

このプログラムを MacOSX のターミナル上で、シェル・スクリプト（表 2）から実行し、結果をファイルに記録する。そして結果を整理するプログラム（表 3）を使って、平均たらい回し連続回数と、表 1 のプログラムを 1 セッション実行した時の最長たらい回し連続回数の平均値と、最長たらい回し連続回数の度数分布を調べた。

使用した operating system は MacOSX10.4.7 であり、プログラム言語等は Perl version 5.8.6 及び、GNU bash、 version 2.05b.0(1)-release（powerpc-apple-darwin8.0）である。

表 2　シェル・スクリプト（1000sets）

```
#!/bin/bash

#first time
perl taraimawasi.pl > 1000sets_results

#second times 以降
for ((i=1;i<1000;i++))
do
  perl taraimawasi.pl >> 1000sets_results
done
```

表 3　結果を整理するプログラム（kekka.pl）

```
#!/usr/bin/perl

open RESULTS, "< 1000sets_results";
while (<RESULTS>) {
      chomp;
      my @fields = split;#区切り文字が空白文字の場合
      my $temp = @fields -2 -1;#配列@fieldsの要素数から最初の二つを除くと、taraimawasi.plにおける配列mawashiの部分になり、配列mawashiは零始まりなので、一セッションの最長たらい回し回数を得るには、もう1だけ引くことになる。
      #print "$temp¥n";
      push (@longest_taraimawasi,$temp);
      $summary[0] += $fields[0];
      $summary[1] += $fields[1];
}
close RESULTS;

#１案件処理にかかる平均たらい回し回数
#print "$summary[0],$summary[1],";
$average_taraimawasi = $summary[0] / $summary[1];
```

```
print "Average Taraimawasi = $average_taraimawasi¥n";

#最長たらい回し回数
#print "@longest_taraimawasi¥n";
	#平均最長たらい回し回数、及び、度数分布frequency distribution
	$sum = 0;
	foreach $longest_t (@longest_taraimawasi) {
		$sum += $longest_t;
		$longest_taraimawasi_fd[$longest_t]++;
	}
	$average_longest_taraimawasi = $sum / @longest_taraimawasi;#配列@longest_taraimawasiの
要素数はセッション数であり、この場合は1000となる。
	print "Average Longest Taraimawasi = $average_longest_taraimawasi¥n";
	#度数分布の配列のうち、未定義となっている所に0を埋める
	foreach (@longest_taraimawasi_fd){
		if ($_ == undef){
			$_ = 0;
		}
	}
	print "Frequency Distribution of Longest Taraimawasi = @longest_taraimawasi_fd¥n";
```

結　　果

たらい回しにならずに処理されるのが一万回を一セッションとし､それを千セッション実行した。千セッションでの平均たらい回し連続回数は2.4814945回であった。一セッション毎の最長たらい回し連続回数の度数分布は表４の通りである。度数分布表及びそれを元にしたヒストグラム（本稿では掲載せず）を見ると、一セッション毎の最長たらい回し連続回数の平均値は82.837であり、最頻値は77でその時の度数は46であった。また、山なりの曲線は、正規分布ではないが、分布が右にずれているような分布であった。

表４　一セッション毎の最長たらい回し連続回数の度数分布

0	0	20	0	40	0	60	3	80	32	100	5	120	0	140	1
1	0	21	0	41	0	61	3	81	31	101	8	121	0	141	0
2	0	22	0	42	0	62	8	82	22	102	8	122	3	142	0
3	0	23	0	43	0	63	13	83	22	103	7	123	1	143	0
4	0	24	0	44	0	64	12	84	28	104	8	124	1	144	0
5	0	25	0	45	0	65	13	85	22	105	12	125	2	145	0
6	0	26	0	46	0	66	12	86	32	106	3	126	1	146	0
7	0	27	0	47	0	67	22	87	19	107	3	127	1	147	0
8	0	28	0	48	0	68	25	88	27	108	4	128	0	148	0
9	0	29	0	49	0	69	30	89	26	109	8	129	0	149	1
10	0	30	0	50	0	70	27	90	14	110	8	130	0		
11	0	31	0	51	0	71	33	91	18	111	4	131	0		
12	0	32	0	52	0	72	37	92	22	112	4	132	0		
13	0	33	0	53	0	73	29	93	19	113	0	133	0		
14	0	34	0	54	0	74	21	94	19	114	4	134	0		
15	0	35	0	55	0	75	29	95	14	115	0	135	0		
16	0	36	0	56	1	76	33	96	26	116	4	136	0		
17	0	37	0	57	0	77	46	97	7	117	2	137	0		
18	0	38	0	58	1	78	34	98	11	118	1	138	1		
19	0	39	0	59	1	79	37	99	13	119	0	139	1		

（左：一セッション毎の最長たらい回し連続回数、右：出現回数）

各々の担当者の有効エネルギー（effective energy of each tantosya）の作用を調べるため、それを計算から外して試行した。具体的には、サブルーチン a_t_y_matching の所で、a_t_matching を使用すると、やる気フラグを外すことになるので、effective energy of each tantosya の影響を排除できる。これを使って同様に千セッション実行した。

千セッションでの平均たらい回し連続回数は0.4996419回であった。おおよそ、たらい回しが起きずに処理されるのが２回につきたらい回しが１回起きる勘定である。これは今回使用した能力分布構造からしても予想通りと言える。また一セッション毎の最長たらい回し連続回数の度数分布は（0 0 0 0 0 0 9 203 398 238 101 35 10 2 1 2 1）である。一セッション毎の最長たらい回し連続回数の平均値は8.402であり、最頻値は８で、その時の度数は398であった。

各々の担当者の有効エネルギーを入れた結果と比べると、千セッションでの平均たらい回し連続回数で約５倍、一セッション毎の最長たらい回し連続回数の平均値で約10倍、最頻値で約６倍の差が出た。各々の担当者の有効エネルギーの増減は大きな作用があると言える。

考　察

本稿のモデルは、各々の担当者の有効エネルギーについて単純ゴミ箱モデルを参考にしたが、案件それぞれについてはエネルギーを考慮していない。従って単純ゴミ箱モデル・プログラムにおいて分類される、飛ばしやり過ごしのような現象について取り組んでいる訳ではない。おそらく、たらい回しに遭っても何度も解決を求めなければならないような案件とは、高いエネルギー状態であると想定される。従って、案件それぞれについてのエネルギー状態をモデルに盛り込むことで複雑にするよりも、他の条件を取り入れる方がいいのであろう。

結　論

役割分担の重複がもたらす問題について、コンピュータ・シミュレーション・モデルを作成し考察した。成員数が３で、役割の種類が３種類を想定し、行列で表現した。また役割分担の重複の状況も行列で表現した。更に単純ゴミ箱モデル・プログラムで使用された、有効エネルギー状態の増減を、各担当者に導入した。

各々の担当者の有効エネルギーを外した場合は、ほぼ予想通りとなった。各々の担当者の有効エネルギーの増減を入れた場合は、前者と比べてたらい回しの状況が大きく増えていた。

（本稿は、高崎経済大学特別研究奨励金の交付を受けた研究成果の一部である。）

（ふじもと　てつ・本学経済学部助教授）

参考文献

Cohen, Michael D., James G. March, and Johan P. Olsen (1972) A garbage can model of organizational choice. ***Administrative Science Quarterly,*** 17(1):1-25.

藤本　哲（1998a）「情報技術をめぐる組織構造の研究：組織構造概念の拡張と成員の役割」神戸大学大学院経営学研究科博士論文。

藤本　哲（1998b）「高度な情報機器を扱う組織における問題対処の階層性：組織成員間相互接触活動の記述による構造の構成」『産業と経済』（奈良産業大学）第13巻第３号、63－80頁。

藤本　哲（2001a）「成員間相互行為の記録データを用いて組織構造の公式化次元を記述する方法に関する一考察」『産業と経済』（奈良産業大学）第16巻第１号、43－51頁。

藤本　哲（2001b）「成員間相互行為の記録データを用いて組織構造の複雑性次元を記述する試み」『産業と経済』（奈良産業大学経済経営学会）第16巻第３・４号、257－271頁。

藤本　哲（2002）「成員間相互行為の記録データを用いて組織構造の公式化次元を記述する方法」『産業と経済』（奈良産業大学経済経営学会）第17巻第４号、343－352頁。

藤本　哲（2006）「単純ごみ箱モデル・プログラムのPerlへの移植」『高崎経済大学論集』第48巻第４号、105－113頁。

Hage, Jerald (1965) An axiomatic theory of organization. ***Administrative Science Quarterly,*** 10: 289-320.

Hall, Richard H. (1996) ***Organizations: structures, processes, and outcomes.*** 6th ed. Englewood Cliffs, New Jersey: Prentice-Hall.

ニューハム、キャメロン、ビル・ローゼンブラット（著）、株式会社クイープ（訳）（2005）『入門bash』第３版、オライリー・ジャパン発行、オーム社発売。

NHKプロジェクトX制作班編（2003）「救命救急　ER誕生：日本初　衝撃の最前線」『プロジェクトX挑戦者たち（第16巻）：開拓者精神、市場を制す』日本放送出版協会、11－58頁。

野中郁次郎、加護野忠男、小松陽一、奥村昭博、坂下昭宣（1978）『組織現象の理論と測定』千倉書房。

シュワルツ、ランダル・L、トム・フェニックス（著）、近藤嘉雪（訳）（2003）『初めてのPerl』第３版、オライリー・ジャパン発行、オーム社発売。

高橋伸夫（1993）『組織の中の決定理論』朝倉書店。

武井純孝（2004）『はじめてのPerl』セレンディップ発行、小学館発売。