順序づけの平均：アルゴリズムの研究

順序づけの平均 —— アルゴリズムの研究

沢　田　善太郎

はじめに

沢田（1990）では、Kemeny と Snell の提唱した《順序づけの平均》を用いて、大阪府立大学の学生の政党支持順序を分析した。その際にも述べたことだが、Kemeny と Snell がこの概念を提唱した1962年の著作は、数理社会学の教科書の古典である。順序づけの平均は、その第２章の主題であり、多くの読者の眼に触れたはずの概念である。にもかかわらず、順序づけの平均がこれまで実際の分析でほとんど用いられなかったのは、この概念が不毛であるからではなく、順序づけの平均を求めるためのアルゴリズムが整備されていなかったことによるものであろう。この小論では、沢田（1990）では紙面の関係から触れることのできなかった順序づけの平均のアルゴリズムについて改めて検討し、探索（search）によるアルゴリズムと、これにもとづくコンピューター・プログラムを提示する。試行錯誤を繰り返しながら目的状態に接近する探索は、一面では原始的な方法だが、人工知能研究の発達につれて再評価されつつある。

1　順序づけの平均——基礎的説明

1-1　順序づけの表記法

順序づけされる対象の集合をSであらわす。Sの諸要素に対する個人κの順序づけをA_κであらわし、Aをその全体集合とする。

$S=\{a, b, c\}$ に対して，A_1ではａがｂよりも選好され、ｂがｃよりも選好されているとする。A_2ではａがｂ，ｃより選好され、ｂ，ｃは同順位であるとする。本稿ではこれを、

$A_1=[[a],[b],[c]]$,　　$A_1=[[a],[b,c]]$　　1-1-1

のように表記する。

このように、[　] の中に文字や数字などの要素を並べたものを《リスト》と呼ぶ。1-1-1の表記方法では、リストの要素は、いずれもそれ自体がリストである（以下ではサブリストと呼ぶ)。サブリストの前後関係は、それぞれのサブリストの要素の選好関係を示す（前にあるサブリストの要素は、後にあるサブリストの要素よりも選好されている)。また、同じサブリストの中の要素はそれらが同順位であることをあらわす。

リストは、FORTRAN や BASIC のデータ構造の基本である《配列》のように事前に配列要素の大きさを指定しなくてもよいので、同順位を認める順序データのような不定型データの扱いに便利である。LISP や Prolog など、リスト処理を得意とする言語を用いると、対象間の順位の入れ替えなど、本稿の目的のために不可欠の作業を柔軟におこなうことができる。

1-2　順序づけの平均

順序づけ間の距離

Kemeny と Snell（1962）の順序づけの平均は、《順序づけ間の距離》の概念を基礎にしている。

対象 i と対象 j とに対する個人 κ の《選好関係》を、

$$a_{ij\kappa}=\begin{cases}1：\kappa\text{は i を j よりも選好する}\\ -1：\kappa\text{は j を i よりも選好する}\\ 0：\kappa\text{は i と j を同程度に選好する}\end{cases}\qquad 1\text{-}2\text{-}1$$

であらわす。Kemeny と Snell は、同じ n 個の対象に対する順序づけ A_ι と A_κ との《距離》を、両者の選好関係の不一致の程度に注目して、

$$d(A_\iota, A_\kappa)=\frac{1}{2}\sum_{i,j}^{n}|a_{ij\iota}-a_{ij\kappa}| \qquad 1\text{-}2\text{-}2$$

で定義し、この意味での距離が、

反射律：　$d(A_\iota, A_\kappa)=0$

対象律：　$d(A_\iota, A_\kappa)=d(A_\kappa, A_\iota)$

3角不等式：　$d(A_\iota, A_\kappa)\leqq d(A_\iota, A_\kappa)+d(A_\kappa, A_\iota)$

という、距離の基本性質を満足することを示した。

1-2-4式で等号が成り立つのは、すべての i とすべての j に対する A_κ の選好が A_ι と A_λ との中間であるとき、すなわち、すべての i とすべての j に対して、

$$a_{ij\iota}\leqq a_{ij\kappa}\leqq a_{ij\lambda}\quad\text{または}\quad a_{ij\iota}\geqq a_{ij\kappa}\geqq a_{ij\lambda}$$

のときである。このとき、A_ι、A_κ、A_λ は、《この順で一直線上にある》といい、A_κ を、A_ι と A_λ との《間にある》という。

参考までに、図1に $S=\{a, b, c\}$ の順序づけ間の距離を示す。この図には、直接に線分で結ばれた《隣合う》順序づけ間の距離を記している（A_ι と A_κ とが《隣合う》順序づけであるとは A_ι と A_κ との《間にある》順序づけが存在しないことを意味する）。その他の順序づけ間の距離は、両者を結ぶ最短経路となる——すなわち一直線上にある——順序づけ間の距離の総和である。

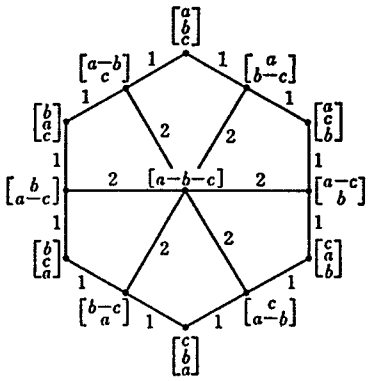

図1　対象3の順序づけ

出典：Kemeny & Snell (1962)

順序づけの平均

Kemeny と Snell は、集合Sの諸要素に対してA $\{A_1, \cdots\cdots A_n\}$ が与えられたとき、

$$v(X, A) = \sum_{i=1}^{n} d(A_i, X)^2 \qquad 1\text{-}2\text{-}3$$

を最小にする順序づけXを、順序づけの平均（以下では、Mであらわす）とよぶ。

算術平均とのアナロジーでいえば、$v(M, A)$をケース数で割った、

$$s^2 = \frac{1}{n} \sum_{i=1}^{n} d(A_i, M)^2$$

は、MのまわりのAの2次の積率であり、算術平均の分散に相当する。算術平均ではAは実数の集合であり、A_iの平均と分散も実数である。これに対して、順序づけの平均では、Aは順序づけの集合であり、平均もまた順序づけである。一方、順序づけの平均の場合にも、s^2は算術平均の場合と同じく非負の実数である。つまり、順序づけの平均は、分散を最小にするという算術平均の特性をより一般的な「平均」の定義として採用することによって成り立つ概念である。（ただし、通常の平均はただ1つの値であるが、この場合には、複数の順序づけが順序づけの平均となる場合がある）。

1-3 順序づけの平均の応用例

Kemeny と Snellによると、順序づけの平均は、Berger, Cohen, Snell and Zeidlich (1962) の小集団研究で利用されたらしいが、筆者は未見である。沢田 (1990) 以外の最近の応用例は見当たらない。沢田は、同年5月におこなった大阪府立大学の学生の政党支持順序の調査データの分析に順序づけの平均を利用するとともに、この概念の延長線上で、順序データに距離行列を用いたクラスター分析風のアプローチが可能であることを示した。

2 順序づけの平均——アルゴリズムの研究

以下では、順序づけの平均を求めるためのアルゴリズムについて論じる。順序づけの平均を定義した1-2-3式は見た目には簡単だが、実際にこの式を展開し、数値計算で順序づけの平均を求めるのは、むつかしい。そこで、本稿では、探索によるアルゴリズムを検討する。

2-1 効率的な探索のための一般的条件

一般的に言うと、探索とは、ある《状態集合》の中からさまざまな《状態》を順々に調べていくことによって、《目的状態》を見いだす施行錯誤の方法である。これを順序づけの平均を求める作業にあてはめると、状態集合はSに対して論理的に可能なすべての順序づけの集合（以下では、Uと呼ぶ）である。たとえば、図1に示したように、$S = \{a, b, c\}$ に対するUの要素は13である。Uの要素のうち、Aに対する分散が最小である順序づけ（すなわち、順序づけの平均）が目的状態である。

探索が試行錯誤の方法であるといっても、非常に多数の状態を文字どおりしらみつぶしに調べる《盲目的探索》と効率的な探索とでは、探索に要する労力に大きな開きが生じる。最初に、言葉の定義をかねて、効率的に探索をおこなうための基本的な条件をあげておく。

i　探索には出発点がある。私たちは状態集合の何らかの要素から探索を開始する。この探索を開始する最初の状態を《初期状態》という。効率的な探索をおこなうためには、私たちが事前に有しているその問題に関する知識をできるだけ活用し、最小限の試行錯誤で目的状態に達するよう、目的状態にできるだけ近い初期状態を選択することが望ましい。

ii　探索の過程で、私たちはいろいろな状態を何らかの基準で評価し、それが目的状態であるかどうか判断する（この判断基準を《評価関数》という）。探索の過程で出会った状態が目的状態でない場合や、あるいは、その状態を調べただけではまだそれが目的状態であるかどうか分からない場合には、引き続き別の状態を探索する。初期状態から出発し，さまざまな状態を調べていくことを《状態遷移》という。状態遷移は系統的かつ効率的におこなう必要がある。つまり、一方では、樹状図等を用いることによって、目的状態になる可能性のある状態を見落とさないように注意する必要があるし、他方では、探索の効率を高めるために、状態集合の中から目的状態になりえない無用な《探索肢の刈り込み》をおこなう必要がある。時間のかかる念入りな評価に先立って、できるだけ早く、できるだけ多くの探索肢を刈り込めるよう、よく工夫された「篩」を備えた評価関数を作るためには、初期状態の選択の場合と同様、事前の知識を最大限に活用する必要がある。

2-2　極値を求める探索

順序づけの平均を求める探索では、2-1で述べた探索の一般的問題とあわせて、それが極値を求める探索であることから生じる困難がある。たしかに、順序づける対象の数が有限であれば、Uも有限集合であるから、Uのすべての要素について、Aに対する分散を計算し、その中から分散の最小のものを選ぶことによって、順序づけの平均は、必ず求めることができる。しかし、このやり方では、順序づけされる対象の数が増加するにともなって、探索肢が爆発的に増加する。極値を求める探索を効率的におこなうには、すべての要素を調べあげることなしに、最適の要素を見つける、という虫のよい注文のかなう探索のやり方を考える必要がある。

《山登り法》と呼ばれる探索がこのような注文に応えてくれる場合がある。一般に、山登り法とは、

i　状態遷移の規則にしたがって、初期状態から１ステップで移行できる諸状態に対して、評価関数による評価をおこなう。

ii　もし、ｉで調べた状態集合の中に、初期状態よりも高い評価を受ける状態があれば、これを出発点として再びｉの探索をおこなう。

iii　もし、ｉの探索で、ある状態の評価関数が、この状態から遷移可能なすべての状態に対して極大（あるいは極小）になれば、この状態を目的状態と判断する。

という手順による探索である（長尾 1988)。

山登り法は、極値を求める探索のなかで最も効率のよいものだが、欠点は、ある状態（Xと

呼ぶ）から１ステップで移行できる状態のなかにＸよりも高い評価を受ける状態がなかったとすると（すなわち、Ｘが局所的に極値であるとすると）、探索はその時点で打ち切られ、Ｘよりも高い評価を受ける状態が別のところにあったとしても、見落とされることである。山登り法によって確実に目的状態を見いだせると言えるのは、目的状態以外に局所的に極値を取る状態が存在しないことが確かめられるときである。

残念なことに、順序づけの平均の探索ではこの保証が得られない。順序づけの平均を求める探索では、状態遷移の規則として1-2で定義した《隣合う》順序づけの概念を用いるのが自然であろう。順序づけＸに隣合う順序づけを、Ｘから１ステップで移行できる順序づけとする。この規則にしたがって状態遷移をおこなうとき、「真の」順序づけの平均以外のところで、Aに対する分散が局所的に極小値を取る順序づけが存在する場合がある。

例)　A_1=[[a],[b],[c]]，A_2=[[b],[c],[a]]，A_3=[[c],[a],[b]]とする。このとき、A= $\{A_1, A_2, A_3\}$ の順序づけの「真の」平均は [[a,b,c]] であり、その分散は、

$$v([[a,b,c]],A)=\sum_{i=1}^{3} d([[a,b,c]],A_i)=3^2+3^2+3^2=27.$$

となる。

他方、A_1=[[a],[b],[c]]の分散は、

$$v([[a],[b],[c]],A)=0^2+4^2+4^2=32.$$

ところが、A_1に隣合う[[a,b],[c]]と[[a],[b,c]]の分散は、

$$v([[a,b],[c]],A)=1^2+3^2+5^2=35.$$

$$v([[a],[b,c]],A)=1^2+5^2+3^2=35.$$

A_1=[[a],[b],[c]]の分散は、「真の」平均である[[a,b,c]]の分散よりは大きいが、すべての隣合う順序づけの分散よりも小さい。すなわち、A_1は局所的な極値である（この例では、A_2、A_3の分散も局所的な極値である)。順序づけの平均を求めるために「山登り法」を採用すると、このような場合には誤った解を見いだす可能性がある。

2-3　順序づけの平均のアルゴリズム

以下では、山登り法ほどには効率的ではないが、盲目的探索よりは効率的な順序づけの平均を求めるための２つのアルゴリズムを述べる。アルゴリズムＩは、2-2の最後に述べた局所的極値の問題を解決しているので、確実に「真の」順序づけの平均を求めることができる。ただし、このアルゴリズムは、回答者の順序づけが多様であり、相互にくい違っていることが多い場合には、探索の効率が低下する。アルゴリズムIIは，局所的極値の問題を完全に解消したわけではないが、ほとんどの場合に真の順序づけの平均を見いだすことができ、かつ、かなり効率的なアルゴリズムである。

基本的な見通し

諸個人のおこなう順序づけにUのすべての要素（論理的に可能なすべての順序づけ）が出現するとは限らない。諸個人の順序づけの中で実際に観測されたω通りの順序づけの集合を

$O=\{O_1, O_2, \cdots\cdots O_t, \cdots\cdots O_\omega\}$ であらわし、その度数分布を $f(O_i, n_i)$ であらわす。$f(O_i, n_i)$ とは、集合 A には O_i である順序づけをおこなった個人が n_i 人存在することを意味する。さらに、$F_t=\{f(O_1, n_1), f(O_2, n_2), \cdots\cdots, f(O_t, n_t)\}$ とし、F_t の順序づけの平均を M_t であらわす。

ここで M_t と $F_{t+1}=\{f(O_1, n_1), \cdots\cdots f(O_t, n_t), f(O_{t+1}, n_{t+1})\}$ の順序づけの平均である M_{t+1} とを比較する。順序づけの平均の定義により、$v(M_t, F_t) \leqq v(M_{t+1}, F_t)$ である。すなわち、

$$\sum_{i=1}^{t} d(O_i, M_t)^2 n_i \leqq \sum_{i=1}^{t} d(O_i, M_{t+1})^2 n_i . \qquad 2\text{-}3\text{-}1$$

また、$v(M_{t+1}, F_{t+1}) \leqq v(M_t, F_{t+1})$ より、

$$\sum_{i=1}^{t} d(O_i, M_{t+1})^2 n_i + d(O_{t+1}, M_{t+1})^2 n_{t+1} \leqq \sum_{i=1}^{t} d(O_i, M_t)^2 n_i + d(O_{t+1}, M_t)^2 n_{t+1} \qquad 2\text{-}3\text{-}2$$

2-3-1式、2-3-2式で等号が成り立つのは、$M_t = M_{t+1}$ のときである。それゆえ、$M_t \neq M_{t+1}$ とすると、両式より、

$$d(O_{t+1}, M_{t+1}) < d(O_{t+1}, M_t). \qquad 2\text{-}3\text{-}3$$

となる。つまり、$M_t \neq M_{t+1}$ なら、M_{t+1} と O_{t+1} の距離は M_t と O_{t+1} との距離より小さい。

このことは、M_t が既知の場合には、M_{t+1} を求めるために U のすべての要素を調べる必要はなく、$U'=\{X : X \in U,\ d(O_{t+1}, X) \leqq d(O_{t+1}, M_t)\}$ の要素についてだけ、F_{t+1} に対する分散を計算すれば、その中で最小の分散をもつXが M_{t+1} であることを意味する。一方、$M_1 = O_1$, $v(O_1, F_1) = 0$ は既知である。それゆえ、$M_1, \cdots\cdots M_t, M_{t+1}, \cdots\cdots, M$ を、このやり方で逐次計算することができる。

これを利用すれば、山登り法ほどには効率的ではないが、盲目的探索よりは、効率的な順序づけの平均を求めるための2つのアルゴリズムが考えられる。

アルゴリズムI

$d(O_{t+1}, X) \leqq d(O_{t+1}, M_t)$ であるすべての順序づけについて F_{t+1} に対する分散を計算し、最小の分散をもつ順序づけ（M_{t+1}）を選ぶ。これを繰り返して、$M=M_\omega$ を見いだす。この方式は、各人の順序づけが比較的狭い（相互に距離の短い）範囲に集中している場合にはある程度の効率を保てる。また、大半の順序づけが狭い範囲に集中し、若干の例外ケースが特異な順序づけをおこなっている場合には、特異なケースの分散の計算に手間取らぬよう、これらのケースについては F の添字のできるだけ若い番号を割り当てるのが賢明である。しかし、このような計算順序の工夫をおこなっても、回答者の順序づけが非常に多様である場合には、計算速度は低下する。

アルゴリズムII

M_tからM_{t+1}を求める際に、M_tを初期状態とする山登り法による探索をおこなうなら、探索の効率はかなり改善される。アルゴリズムIIは、M_tからM_{t+1}を求めるにあたって、

i　$v(M_t, F_{t+1})$ を求める。

ii　M_tと隣合う順序づけ（Xと呼ぶ）の中に、

条件1：$d(O_{t+1}, X) < d(O_{t+1}, M_t)$,

を満足するものが存在するかどうか調べる。

iii　もしM_tと隣合う順序づけの中に、条件1を満たす順序づけがあれば、それが、

条件2：$v(X, F_{t+1}) \leqq v(M_t, F_{t+1})$

を満足するかどうか調べる。

iv　もし、M_tと隣合う順序づけの中に、条件1と条件2を満たす順序づけがなければ、$M_t = M_{t+1}$である。

v　もし、M_tと隣合う順序づけの中に、条件1と条件2を満たす順序づけが1つあるいはそれ以上存在すれば、これらを新たな出発点として探索を続ける。すなわち、Xと隣合う順序づけ（X′と呼ぶ）のうちに、まだ調べていない順序づけがあれば、その中に、

条件1：$d(O_{t+1}, X') \leqq d(O_{t+1}, M_t)$,

条件2：$v(X', F_{t+1}) \leqq v(X, F_{t+1})$

を満足するものがあるかどうかを調べる。

vi　もし、ivの条件1と条件2を満足する順序づけがなければ、Xの中で最小の分散をもつ順序づけがM_{t+1}である。

vii　もし、ivの条件1と条件2を満足する順序づけが1つあるいはそれ以上あれば、これらを出発点としてv～viiの探索を続ける。

という、手順を用いる。

上の手順では、出発点であるM_tから徐々に遠ざかり、次第にO_{t+1}に近づく方向で探索が進む。この過程で調べた順序づけXでは、

$$v(X, F_{t+1}) = v(X, F_t) + d(O_{t+1}, X)^2$$

である。

XがO_{t+1}に近づくと、$d(O_{t+1}, X)^2$の値は減少する。この減少効果は、$x = d(O_{t+1}, X)$とすると、XからさらにO_{t+1}との距離が1減少した順序づけX′と比較して、

$$d(O_{t+1}, X)^2 - d(O_{t+1}, X')^2 = x^2 - (x-1)^2 = 1 - 2x$$

であるから、$d(O_{t+1}, X)$が大きいほど大きく、XがO_{t+1}に近づくほど減少する。これに対して、$v(X, F_t)$は、多くの場合、$d(X, M_t)$が大きくなるほど増大し、かつ、その増大効果は$d(X, M_t)$が大きくなるほど大きいと考えられる。このような$d(O_{t+1}, X)^2$の減少効果と$v(X, F_t)$の増大効果との均衡点がF_{t+1}における順序づけの平均となる。

このアルゴリズムでは、$O = \{O_1, O_2, \cdots\cdots, O_\omega\}$の添字を、出現度数の多いものほど若い番号になるようにつけておくと、M_tとM_{t+1}との距離は比較的短くなり、最小限の試行錯誤でM_{t+1}にいたるので、探索の効率化がはかれる。アルゴリズムIIは局所的極値の問題を

完全に解決はしていないが、筆者が仮想データを含め20～30回ほど試した限りでは誤った解が生じたことはない。

3　コンピューター・プログラム

アルゴリズムIIのプログラムを紹介する。プログラムに用いた Prolog は、人工知能研究等の分野で注目されているコンピューター言語であり、上のアルゴリズムを実現するために必要なリスト処理や探索を得意とする。最初に、プログラムの説明をおこなうが、残念ながら、Prolog を未経験の読者には分かりにくいであろう。興味のある方は、他の参考書（柴山・桜川・萩野1986、後藤1984、小谷1986等）を参照されたい。

データ

プログラムの冒頭部分は、沢田（1990）の政党支持順序の調査結果を３つの述語を用いて表現している。引数１の述語 object_set は順序づけられる対象の集合(S)をあらわす。引数３の述語 frequency は、2-3の$f(O_i, n_i)$に相当する（第１引数は識別番号、第２引数は政党の選好順序、第３引数は観測度数である)。n_of_patterns(64) は、この調査では64通りの順序づけのパターンが観測されたことを述べている。

順序づけ間の距離

1-2-2式で定義した順序づけA、B間の距離Dは、述語 distance（A，B．D）で与えられる。距離の計算に必要な1-2-1式の選好関係は、述語 preference で定義される。preference は、述語 forward を用いて政党支持順序のリストの前後関係を調べ、それを選好関係に翻訳する。(なお、distance の条件節に出てくる known_distance は、同じ計算を何回もおこなう無駄を避けるため、距離計算の結果を記憶させているものだが、パソコンで大量データを扱う場合にはプログラム領域をオーバーする危険がある)。

分散の計算

$F_t = \{f(O_1, n_1), f(O_2, n_2), f(O_t, n_t)\}$ に対する順序づけXの分散Vが、variance（V，X，T）で与えられる。ここでは、F_tまでの分散の計算結果を known_variance に記憶させることによって、その後の計算の効率化をはかっている。

平均の探索

① 述語 mean は実行プログラムである。このプログラムを学んだ Prolog 処理系に mean と入力すると、処理系は、n_of_patterns(ω) を読み取り、for ループによってω回、search_mean を呼び出して、$M_1, M_2, \cdots\cdots, M$を求め、$M$とその分散を出力する。

② 述語 search_mean は次の２つの作業をおこなう。

i　述語 init によって、M_tを探索するにあたっての初期状態を設定する。(init は、複数

のM_{t-1}の中からO_tとの距離が最小となる順序づけを選び、“ini”を第1引数にするスタックに挿入する)。

ii 述語 climb_hill を参照し、山登り法によってM_tを探索する。

③ climb_hill は、与えられたスタックに含まれた順序づけ(State と呼ぶ)を先頭から順に取り出し、述語 evalue_neighbor を参照して、State と隣合う順序づけの評価をおこなった後、State より分散が小の順序づけがあれば、⑤で述べるように、《再帰》をおこなう。

④ 述語 evalue_neighbor は次の作業をおこなう。

i State を第1引数にしたスタックを作る。スタックは空にしておく。

ii 述語 neighbor と put_forward を用いて、State と隣合う順序づけを作る。put_forward(X,Y,State,New_state) とは、「State における任意の項Xの空でない部分集合Yを State にける位置から前に1つ進め、New_state を作る」という操作を意味する。この操作は、

・Xが State の先頭の項であり、X＝Yの場合には不可能である。

・Xが State の先頭の項でなく、かつX＝Yの場合は、Yをその直前の項と併合する。

例) put_forward([b],[b],[[a],[b],[c]],[a,b],[c]]).
put_forward([b,c],[b,c],[a],[b,c]],[[a,b,c]]).

・YがXの真部分集合である場合には、YをXから除き、その直前の項とする。

例) put_forward([a,b],[a],[[a,b],c]],[[a],[b],[c]]).
put_forward([a,b,c],[b,c],[[a,b,c]],[[b,c],[a]]).

State からこの操作で作られる順序づけ(New_state)が State に隣合う順序づけである。

iii New_state を次の篩を用いて評価する。

・test 1 は、New_State がまだ評価を済ませていない順序づけである場合にのみ成功する。

・test 2 は、New_State が、2-3-3式を満たす場合、すなわち、

$$d(\text{New\_State}, f_t) < d(\text{State}, f_t)$$

の場合にのみ成功する。

・test 3 は、New_State のF_tに対する分散が State のそれよりも小さい場合にのみ成功する。

iv もし、New_State が上の3つのテストに合格すると、State を第1引数とするスタックに挿入する。また、探索の過程で得られた最小の分散をもつ順序づけとその分散を記録する述語 min を参照し、New_State の分散がリストのそれよりも小の場合や同等の場合には、min の内容を書き換える。

v State と隣合うすべての順序づけについてii～ivの作業を繰り返す。

⑤ evalue_neighbor の作業が終了した時点で、State と隣合う順序づけの中に State より分散が小であるものがあれば、それは、State を第1引数にするスタックに記録されている。その場合には、これらの状態を出発点として climb_hill による探索を継続する(⇒③に戻る)。

⑥ evalue_neighbor の作業が終了した時点で、State を第1引数にしたスタックが空であれば、State は極値である。それゆえ、第1引数 “ini” のスタックに代替肢がない限り、探索は終了する。この時点で述語 min に記録されている1つあるいは複数の順序づけがM_tである。

以下に掲載するプログラムは、以上のようなプログラムと、それを実行するために必要なリスト処理等のための汎用述語で構成されている。

```
**********************************************
*          順序づけの平均                    *
**********************************************

****************** データ ******************

object_set([自,社,公,共,民]).
n_of_patterns(64).

frequency(1,[[自],[社],[公,共,民]],11).
frequency(2,[[自],[社,公,共,民]],8).
frequency(3,[[自,社,公,共,民]],8).
.....
                中略
.....
frequency(64,[[公,共,民][自,社]],1).

****************** 距離の計算 ******************

distance(A,A,0) :- !.
distance(A,B,D) :-known_distance(A,B,D), !.
distance(A,B,D) :-known_distance(B,A,D), !.
distance(A,B,_) :-object_set(U), init_sigma(distance), forward(I,J,U),
        calc_distance(A,B,I,J), fail.
distance(A,B,D) :-sigma(distance,D), assert(known_distance(A,B,D)),!.

calc_distance(A,B,I,J) :-belong(I,Ia,A), belong(J,Ja,A), belong(I,Ib,b),
        belong(J,Jb,B), preference(A,Ia,Ja,Aij), preference(B,Ib,Jb,Bij),
        Dij is Aij-Bij, abs(Dij,X), add_sigma(distance,X), !.

preference(X,I,I,0) :- !.
preference(X,I,J,1) :-forward(I,J,X), !.
preference(X,I,I,-1) :- !.
```

```
forward(_,_,[A}) : - !, fail.
forward(A,B,[A | R]) : -member(B,R).
forward(A,B,[F | R]) : -forward(A,B,R).

belong(A,A0,L) : -member(A0,L), member(A,A0), !.

                    ********** 分散の計算 **********
variance(V,X,T) : -known_variance(V,X,T), !.
variance(V,X,T) : -viewed(known_variance(V0,X,T0)), !,
        calc_variance(X,T0,T,V0,V).
variance(V,X,T) : -calc_variance(X,0,T,0,V), !.

calc_variance(X,T0,T,V0,_) : -T1 isT0+1, init_sigma(var,V0), for(K,T1,T),
        frequency(K,0,F), distance(X,0,D), DD is D*D,
        DDF is DD*F, and_sigma(var,DDF), fail.
calc_variance(X,_,T,_,V) : -sigma(var,V), assert(known_variance(V,X,T)), !.

                    ********** 平均の探索 **********
mean : -n_of_patterns(W), for(T,1,W), search_mean(T,Mean,Variance),
        assert(mean(T,Mean,Variance)), fail.
mean : -mean(_,Mlist,Variance), report(Mlist,Variance).

report(Mlist,_) : -write(平均 :  ), member(M,Mlist), write(M), nl, tab(8), fail.
report(_,Variance) : -nl, write(分散 :  ), write(Variance).

search_mean(1,[O1],0) : -frequency(1,O1,_), !.
search_mean(T,_,_) : -T0 is T-1, viewed(mean(T0,List,_)), init(T,List,[F | R]),
        init_stack(ini,[F | R]), variance(V0,F,T), init_min(mean,[F | R,V0),
        climb_hill(ini,T).
search_mean(T,Mean,Variance) : -min(mean,Mean,Variance), del(tested(_)), !.

init(T,[A],[A]) : -assert(tested(A)), !.
init(_,List,_) : -init_stack(mlist,List), init_min(org,[],9999), fail.
init(T,_,_) : -pop_stack(mlist,State), assert(tested(State)),
        frequency(T,Observed,_), distance(Pbserved,State,D),
        find_min(org,State,D), fail.
```

```
init(_,_,L2) : -min(org,L2,D),  ! .

climb_hill(Title,T) : -pop_stack(Title,State),  evalue_neighbor(State,T),
        climb_hill(State,T).
evalue_neighbor(State,_) : -init_stack(State,[]),  fail.
evalue_neighbor(State,T) : -frequency(T,Observed,_),
        distance(State,Observed,D0),  variance(V0,State,T),
        neighbor(State,New_state),  test1(New_state),
        test2(New_state,D0,Observed),  test3(New_state,T,V0,V),
        replace(stack(State,Y),stack(State,[New_state | Y])),
        find_min(mean,New_state,V),  fail.
evalue_neighbor(_,_) : - ! .

neighbor(State,New_state) : -member(X,State),  subset(Y,X),  ¥+A=[],
        put_forward(X,Y,State,New_state).

put_forward(X,X,[A | _],_) : - ! ,  fail.
put_forward(X,X,State,New_state) : -append(Head,[A,X | R],State),
        append(A,X,B),  append(Head,[B | R],New_state),  ! .
put_forward(X,Y,State,New_state) : -append(Head,[X | Tale],State),  dif(X,Y,Rest),
        append(Head,[Y,Rest | Tale],New_state),  ! .

test1(New_state) : -tested(X),  equal(New_state,X),  ! ,  fail.
test1(New_state) : -assert(tested(New_state)),  ! .

test2(New_state,D0,Observed) : -distance(Observed,New_state,D),  D<D0,  ! .
test3(New_state,T,V0,V) : -variance(V,New_state,T),  V=<V0,  ! .

equal(X,X) : - ! .
equal(X,Y) : -length(X,N),  ¥+length(Y,N),  ! ,  fail.
equal([F1 | R1],[F2 | R2]) : -permutation(F1,F2),  equal(R1,R2).

init_min(N,X,Y) : -del(min(N,_,_)),  assert(min(N,X,Y)).

find_min(N,S,V) : -viewed(min(N,L0,V)),  ! ,  assert(min(N,[S | L0],V)).
find_min(N,_,V) : -min(N,L0,V0),  V0<V,  ! .
find_min(N,S,V) : -replace(min(N,_,_),min(N,[S],V)),  ! .
```

```
**********汎 用 述 語**********
member(A,[A | L]).
member(A,[F | L]) :-member(A,L).

append([],X,X).
append([F | X],Y,[F | Z]) :-append(X,Y,Z).

dif(A,[],A).
dif([],_,[]).
dif([F | X],Y,Z) :-member(F,Y), !, dif(X,Y,Z).
dif(F | X],Y,[F | Z]) :-dif(X,Y,Z).

subset([],[]) :-!.
subset(F | X],[F | Y]) :-subset(X,Y).
subset(X,[F | Y]) :-subset(X,Y).

permutation(X,X).
permutation(Org,[F | X]) :-delete(Org,F,Y), permutation(Y,X).

delete(A | X],A,X).
delete(F | R],A,[F | X]) :-delete(R,A,X).

init_sigma(X) :-del(sigma(X,_)), assert(sigma(X,0)), !.

init_sigma(X,S) :-del(sigma(X,_)), assert(sigma(X,S)), !.

add_sigma(N,X) :-viewed(sigma(N,S0)), S is S0+X, assert(sigma(N,S)), !.

abs(X,X) :-X>=0, !.
abs(X,Y) :-Y is -X.

viewed(Data) :-Data, retract(Data), !.

del(Data) :-repeat, ¥+retract(Data), !.

replace(R1,R2) :-retract(R1), assert(R2), !.
```

```
init_stack(N,L) : -del(stack(N,_)), assert(stack(N,L)), !.

pop_stack(N,_) : -viewed(stack(N,[])), !, fail.
pop_stack(N,S) : -viewed(stack(N,[S | R])), assert(stack(N,R)).

for(N,N,J) : -N<=J.
for(N,I,J) : -I<J, I1 is I+1, for(N,I1,J).
```

****

参 考 文 献

Berger, Cohen, Snell, and Zeidlih, 1962, *Types of Formalization in Small Group Research Research*.

後藤滋樹, 1984,『PROLOG 入門』, サイエンス社.

Kemeny J. G. & J. L. Snell, 1962, *Mathematical Models in the Social Sciences*, New York : Ginn. 甲田和衛, 山本国雄, 中島 一(訳)『社会科学における数学的モデル』, 培風館, 1966年.

小谷善行, 1986,『知識指向言語 PRPLOG』, 技術評論社.

長尾 真, 1988,『知識と推論』, 岩波書店 (講座ソフトウェア科学14).

沢田善太郎, 1990,「順序づけの平均」,『人間科学論集』, 第22号, pp.23-46, 大阪府立大学人間科学研究会

柴山悦哉・桜川貴司・萩野達也, 1986,『Prolog-KABA 入門』, 岩波書店.