遺伝的アルゴリズムの改良とその応用

福井大学
工学部　研究報告
第40巻　第1号
1992年3月

遺伝的アルゴリズムの改良とその応用

渡辺勝正*　池田宜明*　松尾　聡*　都司達夫*

Improvement of the Genetic Algorithm and its Application

Katsumasa WATANABE, Yoshiaki IKEDA, Satoshi MATSUO, and Tatsuo TSUJI

(Received Feb. 8, 1992)

The Genetic Algorithm (GA) is known as a method to find near optimum solutions of optimization problems in short time. In this paper, we describe how to apply the GA to some problems and how to modify the GA to be conformable to the problems.

Firstly, we introduce the GA, and compare the time of computation and the solution for traveling salesman problem (TSP) in the uses of one by one method with the GA. Secondly, on the knapsack problem, we discuss the ways of generating new genes by cross-over and mutation. Thirdly, we apply the GA to find the minimum value of continuous functions, and propose a method of narrowing the range of new genes for increasing the precision of the solutions. Lastly, we implement the GA on a parallel computer system and evaluate the processing time in some parallel processing modes for the knapsack problem.

As a result of this research, we could conclude that the GA is a basic universal algorithm applicable to many types of problems, and that it is valuable to extend the application of the GA.

1．はじめに

遺伝的アルゴリズム(Genetic Algorithm，以下，GAと略す)は，最適化問題の近似解を，素早く求める方法のひとつとして知られている(文献1，2等)，それは，必ずしも，最適解を見つけ出すことを保証しているわけではないが，そこそこ使える近似解を，ともかく，短時間で見つけ出すことのできる方法である。John H. Holland(米，ミシガン大学)により，1960年代に考え出された方法で(文献3)，生物の遺伝と進化をモデルにした計算法である。乱数を用いており，決定性の計算方法でないために，計算時間(計算回数)と解の収束性の関係について，形式的な予測をすることが困難である。しかし，反面，アルゴリズム(計算手順)の単純さと一様性により，いろいろの型の問題に適用できる可能性を持っているため，ニューロネットワークによる計算と同じように，計

*情報工学科

算方法の新しい枠組みを与えるものと期待されている(文献１)。

その基本的な計算手順の骨組は次のとおりである。

1) 事象のモデル化(modelling)　問題をモデル化して，解を記号列で表現する。
2) 初期集団の発生(initiation)　第１世代の初期遺伝子(gene)を造り出す。
3) 各個体の評価 (evaluation)　各遺伝子の解としての適合性を評価して，順序付ける。
4) 淘　　汰 (selection)　不適合な遺伝子を削除する。
5) 増　　殖 (reproduction) 削除した分を，適合性の高いものを複製して補う。
6) 交　　差 (cross-over)　２つの遺伝子間で特定部位を入れ替える。
7) 突然変異 (mutation)　新しい遺伝子を生成して，増殖したものとおき替える。
8) 終了の判定 (continuation) 終了か，継続か(３から)を判定する。

手順の中では，次のような実行のパラメータが使われる。カッコ内に，本研究で最も適切であると判断して，一番よく使った値を示しておく，

m(16, 24, 32) ひとつの遺伝子を構成する記号列の長さ，または，ビット数。
p (10)　遺伝子の個数(人口，population)。
rp(0.2)　削除して，増殖する割合い。すなわち，p×rp個をおき替える。
co(0.4)　遺伝子の中で交差させる記号(ビット数)の割合。
r (0.1, 0.2)　突然変異を行う割合。乱数によって操作を行うか否かを決定する。
seed　乱数発生関数rand()に与えられる乱数の出発値(種)。

これらの値は，初期集団の発生の前に，外部から設定される(プログラム１のsetpara)。たとえば，n＝８，p＝10，rp=0.2，co=0.4，r＝0.2のとき，次のようになる。

・p×rp=10×0.2＝２により，適合性の悪い２つを削除して，良い２つを増殖(複製)する。
・n×co＝８×0.4=3.2により，ビット位置の０から３までの入れ替えを行う。たとえば，

A＝１１１１１１１０　と　B＝１００００００１　から
A′＝１１１１０００１　と　B′＝１０００１１１０　が得られる。

なお，交差を行う遺伝子の対をどのように選ぶかは，３節で議論する。

・乱数の発生により，0.2の割合で，突然変異の操作を行うことにする。このとき，p×r＝10×0.2＝２個の適合性の悪い遺伝子を，新たに生成した遺伝子でおき替えることにする。

計算手順の終了は，次のような判断で，endflagを１に設定することにより，決定される。

e１　最適値が判っている場合には，それを見つけたとき。ただし，これは，一般には使えない。
e２　(評価値の平均－評価値の最小)<(評価値の最大－評価値の最小)×JUDGE
となったとき。JUDGEは，0.1，0.01等に設定される。これは，その世代の遺伝子全体が，ほぼ同じ評価値をもつことを示している。最適解に近づいていると考えている。
e３　世代数(gcount)を限定して(MAX)，それをオーバーしたとき。
e４　対話型で，一世代毎の状況を見て(outstep)，ユーザーの判断にまかせる(qflag)。以上の計算手順の流れは，プログラム１のように構成される。

プログラム１　遺伝的アルゴリズムのメインフロー

```
int     p;              /* population                  */
float   rp, co, r;      /* reproduction, cross-over, new generation */
int     gcount;         /* generation count            */
int     smallest, largest;
float   aver;

main()
{
        int     endflag;/* flag of loop repetition */
        char    qflag;  /* from terminal key       */

/*1*/   takefunc();     /* function definition from        */
        outfunc();      /* output function definition      */
        setpara();      /* set parameters of execution     */
/*2*/   makefirst();    /* create first generation         */
/*3*/   sortandp();     /* sort on gene's value and output */
        gcount=1;       /* count of generation             */
        scanf("%c",&qflag);     /* take out last Return    */
    for(endflag=0;endflag==0;gcount++){   /* repeat until endflag==1 */
        outstep();      /* output each step status         */
        if(smallest==min){
/*e1*/          printf("You reach at the optimum value!¥n");
                endflag=1;
        }
   else if((aver-smallest)<(largest-smallest)*JUDGE){
/*e2*/          printf("Your smallest value is near optimum!!¥n");
                endflag=1;
        }
   else if(gcount>MAX){
/*e3*/          printf("Too many repetition.....¥n");
                endflag=1;
        }
   else {                       /* ask to you trying more or not */
/*e4*/      if(flag<=1) scanf("%c",&qflag);
            if(qflag=='q'){     /* quit ? */
                printf("Stop of trying according to your suggestion¥n");
                endflag=1;
            }
        else{               /* repeate more                              */
/*4, 5*/      reproduc();   /* reproduce : copy better one to worse one */
/*6*/         crossover();  /* cross over with pair on low bits         */
/*7*/         mutate();     /* change with new creation                 */
/*3*/         sortandp();   /* sort on gene's value and output          */
            }
        }  /* end of ask */
    }      /* end of for */
}
```

２．１次元不連続関数の最小値

最初に，１変数の不連続関数$f(x)$の最小値を見つけ出す問題をとりあげて，遺伝子の世代の変化を追跡した。関数は，図１に示すように，〔０，256)の定義域で，不連続な整数値をもつものとする。この例では，最小値は，xが100から102の間での値６となる。

遺伝子は，xの値を２進数で表現するものとして，８ビットで構成される。最下位の桁をビット０とする。rand()により生成された乱数を〔0..255〕に変換して，遺伝子として採用する。

プログラム１に従って実行した結果から，次のことが言える。表１に，最小値を見つけるまでの世代数を，図１に実行状況の変化の一例を示す。

R１　乱数発生の種の値seedが，最適値を見つけるまでの世代数(計算時間)に影響をおよぼしているが，その関係を形式化することは困難である。

R２　遺伝子の個数ｐは，必ずしも，大きいのが良いとは言えない。突然変異により，最適解に当たる可能性は大きくなるが，逆に，不適合な遺伝子を造り出す可能性も増える。

R３　突然変異(手順７)は，全く独立に行うと，ある世代の遺伝子が全体として良い方向にまとまりかけているのに，それを悪くすることがある(図１のgc＝７，８)。しかし，逆に，不適切な所に止っている状態から抜け出す機会を与えてくれる。

R４　淘汰と増殖(手順４と５)により，適合性の良いものを複製してゆくと，同じ遺伝子が増えてくる。同じものの間で交差しても(同じものになり)何の効果もない。そのため，交差においては，次のような操作を行うことにする。

R4.1　ある世代の遺伝子を，適合性の良い順に，id〔０〕..id〔ｐ－１〕並べて，最も良いid〔０〕は交差をせずにそのまま残しておく。

R4.2　１≦ｊ≦ｐ×coの範囲で，遺伝子id〔ｊ〕とid〔ｊ＋１〕を対にして，もし，それらの遺伝子が同じである(id〔j〕＝id〔ｊ＋１〕)か，その関数値が同じである(f(id〔j〕)＝ｆ(id〔ｊ＋１〕))とき，遺伝子のビット位置Ｋ＝0..ｎ－１をランダムに決めて，その値を反転する。

ここでKの位置を乱数で定めるのは，遺伝子のビットを規則的に反転させると，全体として同じ動作の反復に陥ることがあるが，それを防ぐためである。これは突然変異の一種である。

なお，この問題では，$f(x)$の最小値を求めるのに，すべてのxについての関数値を調べると，256の値を参照することになるが，GAで行った場合には，表１のｐ＝10に対して示されるように30個前後の参照で最小値を見つけることができる。ただし，乱数の出発値(seed)が違うと，遺伝子の数ｐを増したからといって，必ずしも，早く最適値に当るとは限らない。

また，表１の最小値を求めるまでの世代数は，最小値(min)が判っている場合の終了判定(e1)によるものである。前節にも述べたように，一般には，minの値が判らない問題を解くのである。minにある閾値を設定して，それを満すひとつの近似値をe1の判定で求めるか，e2～e4の判定で終了したときの最も適合性の良い遺伝子id〔０〕に対応する値を解として採用することになる。

表１および次節以下の表の数値は，GAによって，総当りによる方法よりも，早く最適解またはその近似解が得られる可能性をもっていることを示している。

３．ナップザック問題

次に，交差や突然変異の仕方を検討する例題として，ナップザック問題(knapsack problem Kpと略す)をとりあげる。Kpは与えられたｎ個の荷物から，重量(w_j)の和の制限のもとで，価値(v_j)の和を最大にするように，ｍ個のものを選び出す問題である。これは次のように表わせる。

重量制限　$g=\sum_{j=0}^{n-1} w_j \cdot x_j \leqq G$　ここに，$x_j=0$または１，と選んで，

価値の総和　$f=\sum_{j=0}^{n-1} v_j \cdot x_j$　を最大にする。

ｎ個のものから，任意のｍ個(ｍ＝0..ｎ)を選ぶ方法は，ｎビットの２進数に対応させることができるので，遺伝子をｎビットにして，ビットｊで荷物ｊを選ぶか否かを表現することができる。

```
/* 1次元不連続関数の最小値の検索 */

fmin.d2

source data f[]
f[  0]= 35 :"""""""""""""""""""",
f[  8]= 39 :"""""""""""""""""""""",
f[ 17]= 27 :"""""""""""""",
f[ 22]= 18 :""""""""""
f[ 29]= 10 :"""""
f[ 31]= 20 :""""""""""
f[ 35]=  8 :""""
f[ 37]= 39 :"""""""""""""""""""""",
f[ 39]= 50 :""""""""""""""""""""""""""
f[ 55]= 23 :"""""""""""",
f[ 70]= 12 :"""""""
f[ 79]= 45 :""""""""""""""""""""""""",
f[ 82]= 56 :""""""""""""""""""""""""""""""
f[ 88]= 32 :""""""""""""""""
f[100]=  6 :"""
f[103]= 13 :"""""""",
f[120]=  7 :""""",
f[125]= 75 :"""""""""""""""""""""""""""""""""""""""""""""",
f[142]= 36 :""""""""""""""""""
f[169]= 54 :""""""""""""""""""""""""""""
f[199]= 20 :""""""""""
f[221]= 13 :"""""""",
f[240]=  8 :""""
f[246]= 31 :"""""""""""""""",
f[255]= 15 :""""""""",
f[256]= 17 :"""""""""",

we execute optimization process with -----
p=10  rp=0.20  co=0.40  r=0.20  flag=2
and seed of random number next=876543
now, minimum value is       min =6
        (average) and minimum of this generation--->
generation!
gc=  1: ( 27.500) f[ 30]= 10 :""""""
gc=  2: ( 22.500) f[ 30]= 10 :""""""
gc=  3: ( 24.500) f[ 30]= 10 :""""""
gc=  4: ( 22.100) f[ 30]= 10 :""""""
gc=  5: ( 19.900) f[ 30]= 10 :""""""
gc=  6: ( 19.600) f[ 30]= 10 :""""""
gc=  7: ( 20.400) f[ 30]= 10 :""""""
gc=  8: ( 27.200) f[ 30]= 10 :""""""
gc=  9: ( 19.700) f[102]=  6 :"""
You reach at the optimum value!
```

図1　不連続関数の例と実行状況の一例

表1　1変数不連続関数の最小値を見つけるまでの世代数

最小値の幅	p=10, seed=876543	p=12, seed=293	p=13, seed=12
d1 : 2/256	12　(10+22)	55　(12+108)	15　(13+28)
d2 : 3/256	9　(10+16)	10　(12+ 18)	48　(13+94)

たとえば，図２に，16個の荷物の重量(weight)と価値(val)，および，それらに対して，制限G＝91.12のもとでの最適解と，総当り法での計算時間を示している。また，これをGAで解いた場合の結果の一例をも示している。今度は，最適解がわからないものとして，終了判定は，世代数を256に限定して(e3で)行われている。計算は，257世代まで反復しているが，すでに153世代で最適解を見つけていることが示されている。id〔０〕＝8183というのは，16進で(1ff7)のことである。ｎ＝24(荷物が24個)の場合も含めて，実行結果の一例が表２のようにまとめられる。

総当たり法は，2^nの組合わせを検査するので，計算時間はｎの指数関数で増えてゆく。それは，別のワークステーション〔plum〕で行った総当たり法の計算時間からも確かめられる。

ｎ＝16　のとき　約１分　　(0.99秒)
ｎ＝24　のとき　約６分　　(359秒)　　約360倍
ｎ＝32　のとき　約35時間　(2100分)　　約350倍

しかし，GAでは，対象の数(ビット数ｎ)よりも，次に見るように，交差や突然変異の仕方，遺伝子の数ｐ，および初期集団の設定の仕方(あるいは，乱数の種seed)により，最適解に到達するまでの時間が左右される。次に，これらの点について議論するが，総当たり法で得られた最適解を終了判定(e1)に用いて，計算時間の検討を行っている。

3.1　交差(cross-over)の対の組み方

交差を行う対の選び方については，最良のもの(id〔０〕)を残しておいて，順に２つずつ組み合わせる方法を定めた(Ｒ４)が，それは，次のような比較の結果から適切であると言える。

Pair１　まったくランダムに２つの遺伝子を選ぶ。
Pair２　淘汰・増殖(手順４と５)の後で並び換えを行わずに，上位から２つずつ順に選ぶ。
　　増殖しているので最良のものは必ず残っている。
Pair３　一番良いものと，一番悪いものを順に組み合わせる。

ｎが16の場合と24の場合について，最も良い遺伝子の値と最適値との差が，0.1%以内になるまでの実行時間を表３に示す。ｎ＝16は異った乱数の種による1000回の試行，ｎ＝24は100回の試行の平均時間である。Pair２，すなわち，Ｒ4.2の組み方がすすめられる。良い遺伝子同士を組み合わせた方が，最適解に近い遺伝子が得やすいと考えられる。

3.2　突 然 変 異

突然変異を行って，新しい遺伝子を造り出すのは，次の２つの場合がある。

ａ．交差(手順６)において，同じ遺伝子同士で交差するときに，ランダムにビット位置を選んで反転する。
ｂ．突然変異(手順７)において，全く独立に新しい遺伝子を造り出して，最悪のものとおき替える。

前者(ａ)については，両方共変化させてしまうと，全体としての進化の傾向が小さくなるので，一方はそのままに残しておき，ひとつだけを変化させた方が良い結果が得られた。

後者(ｂ)については，突然変異によって得られた遺伝子と，おき替えるべき(現在)の最悪の遺伝子とを比較することが考えられる。良い方を選べば，全体として進化の方向に進む。比較せずに新しいものでおき替えれば，一時的に全体の傾向は悪くなるが，局所的な最適領域から抜け出せる

```
/* 16個の荷物の重さと価値 */
source data n=16 and limit=   91.12
weight[15]=   16.00    val[15]= 110.60
weight[14]=   15.00    val[14]= 110.50
weight[13]=   14.00    val[13]= 110.55
weight[12]=   13.50    val[12]= 110.03
weight[11]=   12.00    val[11]= 109.92
weight[10]=   11.70    val[10]= 109.01
weight[ 9]=   10.10    val[ 9]= 109.02
weight[ 8]=    9.00    val[ 8]= 109.00
weight[ 7]=    8.75    val[ 7]= 108.88
weight[ 6]=    7.01    val[ 6]= 108.70
weight[ 5]=    6.02    val[ 5]= 107.69
weight[ 4]=    5.98    val[ 4]= 107.95
weight[ 3]=    4.00    val[ 3]= 106.44
weight[ 2]=    3.56    val[ 2]= 106.33
weight[ 1]=    2.00    val[ 1]= 105.20
weight[ 0]=    1.23    val[ 0]= 105.10

/* 総当たりによる最適解の探索結果 */
set print mode - flag:0(noprint, only step count )
CPU    Time : 12.437500 sec
id[ 0]= 8183   f[ 0]=1296.830   g[ 0]=  90.850 <0001111111110111>

set print mode - flag:2(print  , only largest    )
CPU    Time : 12.203125 sec

/* 遺伝的アルゴリズムによる実行の結果 */
we execute optimization process with -----
p=10   rp=0.20   co=0.40   r=0.20   flag=0
and seed of random number next=876543
now, limit value is   91.12 of 16 items
Too many repetition.....
CPU    Time : 0.900391 sec
id[ 0]= 8183   f[ 0]=1296.830   g[ 0]=  90.850 <0001111111110111>

we execute optimization process with -----
p=10   rp=0.20   co=0.40   r=0.20   flag=2
and seed of random number next=876543
now, limit value is   91.12 of 16 items
gc=153: rc= 440 ( 84.368) f[  0]=1296.830   g[  0]=  90.850
gc=257: rc= 721 ( 84.059) f[  0]=1296.830   g[  0]=  90.850
CPU    Time : 0.900391 sec
```

図2　ナップザック問題の例と実行の結果

表2　ナップザック問題の実行時間の比較(終了判定e3による)[peach]

	n=16, G1=91.12(図2のデータ)	n=24, G2=124.12
総当たり法	12.2秒 f=1296.830, g=90.850 最適解=(1ff7):16進	4670秒(1時間17分50秒), 約380倍 f=1832.680, g=123.850 最適解=(18bfff):16進
GA(p=10)	0.9秒 f=1296.830, g=90.850 最良解=(1ff7) [153世代]	1.17秒 f=1830.680, g=121.850 近似解=(0abfff)

表3　交差の対の選び方による実行時間
(最適解との差0.1%)[Convex](秒)

	n=16	n=24
Part1(ランダム)	0.0407	0.567
Part2(上から　)	0.0320	0.459
Part3(上と下　)	0.0463	0.754

表4　遺伝子の数(p)を変えたときの実行時間の比較[Convex](秒)

p=	5	10	12	16	20	30
n=16	0.0371	0.0320	0.0343	0.0343	0.0388	0.0384
n=24	0.981	0.459	0.491	0.633	0.783	0.668

きっかけを与えてくれるものと期待される(R3)。結果として，次のような方策を採用する。

R5　突然変異では，現在のものより良い遺伝子が発生したときのみおき替える。せいぜいn回試行して，より良いものが発生されなかったら，おき替えをしない。

3.3　遺伝子の個数(p)の決定

遺伝子の個数(p)は，多い方が最適解に当る可能性が大きいと考えられる。しかし，遺伝子の評価や並び換えの操作を考慮すると，全体の実行時間は，表4に示すように。p=10のときが最も良いようである。表4の値は，表3の場合と同様に，n=16のときは1000回，n=24のときは100回の試行の平均時間である。

ただし，これは，終了判定e1による(最適解が判っている)場合についての傾向を示したものである。逆に考えると，同じ時間実行したときに，良い遺伝子を造り出すチャンスは，p=10のときに最も多いと解釈できる。

3.4　単位価値の考慮

ナップザック問題では，それぞれの荷物の重量と価値が与えられているが，

単位価値=価値(val)/重量(weight)

を考慮すると，より早く最適解が求められることがある。問題に依存することになるが，GAの一部を次のように変更した場合の結果を述べる。

Kp1　これまでに述べたGAの方法

Kp2　初期集団の設定に欲張り法を適用する。すなわち，第一世代を次のように決める。

i)　単位価値の良い順に，制限G内で荷物を選ぶ。

ii)　それをもとに，指定したビット数ランダムに変更して，p個の遺伝子を造り出す。

Kp3　突然変異のルーチンで，時折，単位価値を考慮する。すなわち，突然変異を行うときに，

単位価値の小さいものを，小さいものから，あらかじめ定めた個数捨てて，空いた重量分単位価値の良いものを入れる。この操作を，突然変異の３分の１の割合で行う。

Kp23　Kp２とKp３の両方を適用する。

それぞれの変更を採用したときの実行時間は，表５にまとめられる。初期設定の効果(Kp２)はあまり出ていない。突然変更に制約をつける効果(Kp３)はｎが大きい程大きくなると期待される。ただし，一例だけであるので，決定的な判定はできない。

表５　単位価値を考慮して変更したときの実行時間［plum］㈱

	Kp1	Kp2	Kp3	Kp23
n=16	0.81	0.73	1.04	1.04
n=24	1.51	1.39	0.40	0.33
n=32	3.72	3.85	0.90	0.80

４．巡回セールスマン問題

ＧＡを適用するためには，遺伝子の表現に工夫をしなければならない場合がある。そのような問題のひとつに巡回セールスマン問題(Traveling Salesman Problem TSP)がある。それは，ｎ個の街を，一筆書きの形式で巡回して，そのときのコスト(時間，距離等)を最小にする経路を見つける問題である。すべての経路を調べるには，(ｎ－１)！/２の検査が必要になる。

ｎ記号から成る遺伝子で，巡回する街の順序を表わすことができる。しかし，街の記号(または番号)をそのまま用いたのでは，交差したときに，同じ街が重ったり，ある街が欠落したりすることがある。そのため，次のような表現法をとる(文献２)。

○街に順番(名前のアルファベット順，番号順序)をつけ，その前からの順序で表わす。

○選択したものを除いて，残っているものの中で前からの番号を用いる。

たとえば，５つの街，ＡＢＣＤＥがあるとき，それぞれ次のように表現される。

ＡＢＣＤＥは１１１１１，ＢＣＡＥＤは２２１２１，ＤＡＥＣＢは４１３２１

このため，遺伝子の生成や，遺伝子からコストを計算するのに，Kpに比べて時間がかかる。表６に，総当りで最適経路を求めたときの実行時間と，GAでその最適解に達する(終了判定e1)までの時間を示す。総当りの時間は，上式により，ほぼn!で増大している。GAは，乱数の種seedを合わせて，３通り実行した結果である。seedの値によって，早く最適解に当る場合もあるが，一般的には，その影響をあまり意識する必要はない。

GAは，限られた時間内で，ある程度使える近似解を見つけたり(終了判定e2，e3)，希望する範囲でのひとつの解を見つけたり(minを設定して終了判定e1，または，対話型でe4)して適用するところにひとつの特徴があると考えられる。

表6　巡回セールスマン問題の遺伝的アルゴリズムの傾向[plum]

街数 n	総当たり 時間 (秒)	遺伝的アルゴリズム srand(0) 時間 (秒)	srand(0) 世代数 (回)	srand(1) 時間 (秒)	srand(1) 世代数 (回)	srand(2) 時間 (秒)	srand(2) 世代数 (回)
8	0.17	0.23	496	0.20	433	0.03	70
9	1.61	1.05	1974	3.14	5946	4.30	8115
10	17.07	5.80	9652	55.40	92141	4.42	7354
11	198.12	38.29	56442	5.97	8795	6.49	9564

5．1変数連続関数の最小値の探索

連続関数の最小値等を探索する場合には，遺伝子は連続的な座標点を表わしてほしい。n＝16ビットでもよいのだが，精度を高めておきたいこと，浮動小数点(float)の仮数部に適合しやすいことにより，n＝24ビットの遺伝子(id)を用いる。idを，$0 \leqq x \leqq 10$の浮動小数点数xに変換して，与えられた関数により，関数値$f(x)$を評価する。

GAとしては，これまでに述べた方針(R1～R5)に従ったものを採用している。4つの関数に対する実行結果の一部を図3に示す。ここで，

with DIFFで終了しているのは，与えられた最小値を用いて，終了判定e1によるものである。

near optimumと表示しているのは，終了判定e2によるもので，遺伝子の集団がほぼ同じものにかたまった場合である。必ずしも最小値にはなっていないが，比較的少い計算世代数で，がまんのできる近似解が得られている。

Too many...と表示しているのは，終了判定e3によるもので，MAXを256と設定している場合の結果である。終了判定e2によって，世代数をあまり大きくしなくても，かなり良い近似解が得られることが，関数2の実行例(p＝10と12の場合)から期待できる

なお，関数3では，$x＝1$と$x＝5$で最小値0をとるが，最小値を与える遺伝子id[0]がいずれになるかは，初期設定に左右される。計算状況を追跡していると，世代の若い間は，両方の座標に対応する遺伝子が混在しているが，淘汰と増殖をくり返しているうちに，どちらか一方の値に近い遺伝子だけの集団になってゆくことが見られる。

6．2変数連続関数の最小値の探索

関数の最小値の検索を2変数関数$Z＝f(x, y)$に拡張する。今度は，遺伝子はx座標を表わすものid$x$とy座標を表すものid$y$が対になる。n＝16のときは，実質的には32ビット，n＝24のときは，48ビットの遺伝子となる。探索空間が拡がることにより，前節の方法そのままでは，世代数を増しても，良い近似値は期待できない。そのため，終了判定がe2(遺伝子がかたまった)，または，e3(世代数がMAXを超えた)による場合には，得られた最良値を探索の中心にして，探索の幅(width)を1/10にせばめて，探索を継続することにする。

図4に，5つの関数の最小値を求める場合の結果を示している。各関数の1段目の出力(width＝10の場合)は，前節の方法(探索の幅をせばめない場合)での近似値である。設定された予想最小値に近くなるか，それより小さくなったときに反復(widthの縮小)を止める。

```
/* p=10 and seed=876543 の場合 ---------- */
p=10  rp=0.20  co=0.40  r=0.20  flag=2
and function number fno= 1
minimum of y=(x- 5.8000)^2+ 1.1000
now, minimum value is       min =1.100000
gc=  8: (  1.122248)  min. is f[  5.773315]=  1.100712
gc= 12: (  1.100767)  min. is f[  5.773315]=  1.100712
Your smallest value is near optimum!!

and function number fno= 2
now y=(x-1.0)*(x-2.0)*(x-5.0)*(x-8.0)
now, minimum value is       min =-65.687500
gc= 10: (-60.406212)  min. is f[  6.892700]=-60.424072
gc= 13: (-60.423973)  min. is f[  6.892700]=-60.424072
Your smallest value is near optimum!!

and function number fno= 3
now y=(x-1.0)^2*(x-5.0)^2
now, minimum value is       min =0.000000
gc= 14: (  0.000020)  min. is f[  4.999084]=  0.000013
gc= 18: (  0.000013)  min. is f[  4.999084]=  0.000013
Your smallest value is near optimum!!

and function number fno= 4
else.. y=(x-0.5)*(x-4.2)*(x-8.8)
now, minimum value is       min =-41.740002
gc= 10: (-32.808952)  min. is f[  6.898346]=-32.831963
gc= 14: (-32.831932)  min. is f[  6.898346]=-32.831963
Your smallest value is near optimum!!

/* p=12 and seed=293 の場合 ---------- */
and function number fno= 1
minimum of y=(x- 5.8000)^2+ 1.1000
gc= 27: (  1.100003)  min. is f[  5.801392]=  1.100002
gc= 28: (  1.100002)  min. is f[  5.800934]=  1.100001
You reach near with DIFF=  0.000001

and function number fno= 2
now y=(x-1.0)*(x-2.0)*(x-5.0)*(x-8.0)
gc=256: (-60.424076)  min. is f[  6.892395]=-60.424084
gc=257: (-60.424076)  min. is f[  6.892395]=-60.424084
Too many repetition.........

and function number fno= 3
now y=(x-1.0)^2*(x-5.0)^2
gc= 25: (  0.000005)  min. is f[  1.000366]=  0.000002
gc= 26: (  0.000004)  min. is f[  0.999908]=  0.000000
You reach near with DIFF=  0.000001

/* p=15 and seed=593 の場合 ---------- */
and function number fno= 1
minimum of y=(x- 5.8000)^2+ 1.1000
gc= 13: (  1.100012)  min. is f[  5.799103]=  1.100001
You reach near with DIFF=  0.000001

and function number fno= 3
now y=(x-1.0)^2*(x-5.0)^2
gc=  4: (  0.428072)  min. is f[  1.003876]=  0.000240
Your smallest value is near optimum!!
```

図3　一変数連続関数の最小値の探索の実行例

探索の幅を小さくすることにより，計算時間さえかければ，いくらでも近似の精度を上げることができる。しかし，その時点での局所的な最適値へは収束するが，大域的な最適値を探索するために，外にとび出して，探索の方向を変えることはない。この点，GA本来の思想から逸れてしまうことになるのかも知れない。それを補うには，時折，元の幅で新しい遺伝子を発生することが必要となる。

7．並列処理による実現

GAによる計算では，一世代内での処理(手順３～７)と，世代間でのつながりは，関係が密であるが，別々の乱数により，別方向を探索することは独立性が強い。そのため，最も適合性の良い遺伝子を，時折，交換しながら，並列探索によって最適解を求める方法に適している。

本節では，並列計算機システムAP1000(富士通研究所，並列処理研究センター)の，64個のセルプロセッサを用いて，並列処理をしたGAの実行形式と，その結果について述べる。使用したのは，１つのホストプロセッサと，最大64個のプロセッサである。ホストとセル間の通信によって，時折，動作の同期をとる。同期パターンとして，次の３形態を考える。

〈パターンⅠ〉

ａ．各セルに，異なる乱数の種(seed)を与えて，GAを実行させる。

ｂ．各セルは，決められた世代間隔ごとに，その世代の最良の遺伝子をホストに送る。

c1．ホストはすべてのセルからの遺伝子を集めて，終了判定を行い，続行か，終了かのメッセージを全セルに放送する。

〈パターンII〉

パターンⅠのａ，ｂにより，各セルの最良の遺伝子を集めて，続行のとき，

c2．最も評価値の悪い遺伝子を送ってきたセルに，集めた遺伝子の中から，評価値の良い順にｐ個の遺伝子を送る。すなわち，そのセルの遺伝子を，他のセルで得られた良いものと入れ替える。

〈パターンIII〉

同様に，続行するときには，

c3．全セルからの遺伝子を，評価値の高い順にソートして，その中から，増殖に必要な数ｐ×rpだけ，評価値の高い遺伝子を全セルに放送する。すなわち，各セルに，良い遺伝子が一部注入される。

ｎ＝24のナップザック問題に対して，最適解を求める場合と，最適解との差が0.1%以内になる近似値を求める場合の実行時間を比較する。これらは，GAによる問題解決法の比較(パターンⅠ，II，III)と，並列処理における並列度と同期のオーバヘッドの関係との２つの側面をもっている。なお，それぞれの実行時間は，100回の試行の平均値であり，乱数の種seedの影響を隠している。

まず，１プロセッサ(ホスト)で実行したときの時間は，次のようである。

最適解を見つける	6.98秒
近似解(最適解との差0.1%)を見つける	0.68秒

次に，セル数を2^1～2^6としたときの実行時間と速度向上比を，表7.0と図5.0および，表7.1と図5.1に示す。表の中で，「間隔」は，各セルで連続して計算する世代数で，並列動作の粒度を表わしている。

```
/* p=10 and seed=987 の場合 */
p=10   rp=0.20   co=0.40   r=0.20   flag=0
and seed of random number next=987

function number fno= 1
f1(x, y)=10*(x-y)^2+(2.5-x)^2
now, minimum value is       min =0.000000
You reach near with DIFF=  0.000001
sx= 0.0000000 sy= 0.0000000 width=10.0000000
gc= 46:  ( 0.0000014)   min.  is f[ 2.5000434,  2.5000310]= 0.0000000

function number fno= 2
f2(x, y)=(x-2)^2-(x-2)*(y-7)+(y-7)^2+1.2
now, minimum value is       min =1.200000
Your smallest value is near optimum!!
sx= 0.0000000 sy= 0.0000000 width=10.0000000
gc=186:  ( 1.2049458)   min.  is f[ 2.0703328,  7.0703220]= 1.2049459
Your smallest value is near optimum!!
sx= 1.5703328 sy= 6.5703220 width= 1.0000000
gc= 61:  ( 1.2000610)   min.  is f[ 2.0000238,  6.9921937]= 1.2000611
Your smallest value is near optimum!!
sx= 1.9500239 sy= 6.9421935 width= 0.1000000
gc=  9:  ( 1.2000035)   min.  is f[ 2.0020053,  7.0017214]= 1.2000035
You reach near with DIFF=  0.000001
sx= 1.9970053 sy= 6.9967213 width= 0.0100000
gc=  3:  ( 1.2000072)   min.  is f[ 1.9997824,  6.9993839]= 1.2000003

function number fno= 3
f3(x, y)=((x-5)-(y-3)-0.11)^2+(x-5)*(y-3)
now, minimum value is       min =-0.004033
Too many repetition.........
sx= 0.0000000 sy= 0.0000000 width=10.0000000
gc=258:  ( 0.0003547)   min.  is f[ 5.0390615,  2.9687512]= 0.0003546
Too many repetition.........
sx= 4.5390615 sy= 2.4687512 width= 1.0000000
gc=258:  (-0.0040161)   min.  is f[ 5.0781240,  2.9290442]=-0.0040161
You reach near with DIFF=  0.000001
sx= 5.0281239 sy= 2.8790443 width= 0.1000000
gc= 38:  (-0.0040319)   min.  is f[ 5.0730281,  2.9275377]=-0.0040322

function number fno= 4
f2(x, y)=(x-4)^2+(y-8)^2+4.2
now, minimum value is       min =4.200000
Your smallest value is near optimum!!
sx= 0.0000000 sy= 0.0000000 width=10.0000000
gc= 56:  ( 4.2295413)   min.  is f[ 3.8281245,  7.9995532]= 4.2295413
You reach near with DIFF=  0.000001
sx= 3.3281245 sy= 7.4995532 width= 1.0000000
gc= 34:  ( 4.2000022)   min.  is f[ 4.0006943,  7.9998975]= 4.2000003

function number fno= 5
fe(x, y)=(x-0.5)*(y-4.2)^2*(x-8.8)
now, minimum value is       min =-579.280823
Your smallest value is near optimum!!
sx= 0.0000000 sy= 0.0000000 width=10.0000000
gc= 56:  (-579.1492920)   min.  is f[ 4.5699759,  9.9999990]=-579.1492920
You reach at the optimum value!
sx= 4.0699759 sy= 9.4999990 width= 1.0000000
gc=  2:  (-588.1343994)   min.  is f[ 4.2750540, 10.4712210]=-671.8012085
```

図4　二変数関数の最小値の探索の例(width/=10)

表7.0　AP1000による最適解の探索(時間：秒　間隔：回)

セル数	パターン　I	パターン　II	パターン III
2 ： 時間(間隔)	3. 594(800)	3. 039(800)	2. 244(400)
4 ： 時間(間隔)	1. 924(400)	1. 703(400)	1. 122(100)
8 ： 時間(間隔)	1. 183(200)	0. 896(200)	0. 677(100)
16 ： 時間(間隔)	0. 829(200)	0. 612(50)	0. 491(100)
32 ： 時間(間隔)	0. 598(200)	0. 546(200)	0. 438(50)
64 ： 時間(間隔)	0. 550(200)	0. 542(200)	0. 542(150)

表7.1　AP1000による近似解の探索(時間：秒　間隔：回)

セル数	パターン　I	パターン　II	パターン III
2 ： 時間(間隔)	0. 533(80)	0. 446(40)	0. 446(80)
4 ： 時間(間隔)	0. 309(40)	0. 257(40)	0. 208(40)
8 ： 時間(間隔)	0. 233(80)	0. 181(40)	0. 202(60)
16 ： 時間(間隔)	0. 205(40)	0. 169(60)	0. 173(40)
32 ： 時間(間隔)	0. 201(40)	0. 199(40)	0. 196(40)
64 ： 時間(間隔)	0. 271(40)	0. 265(40)	0. 267(40)

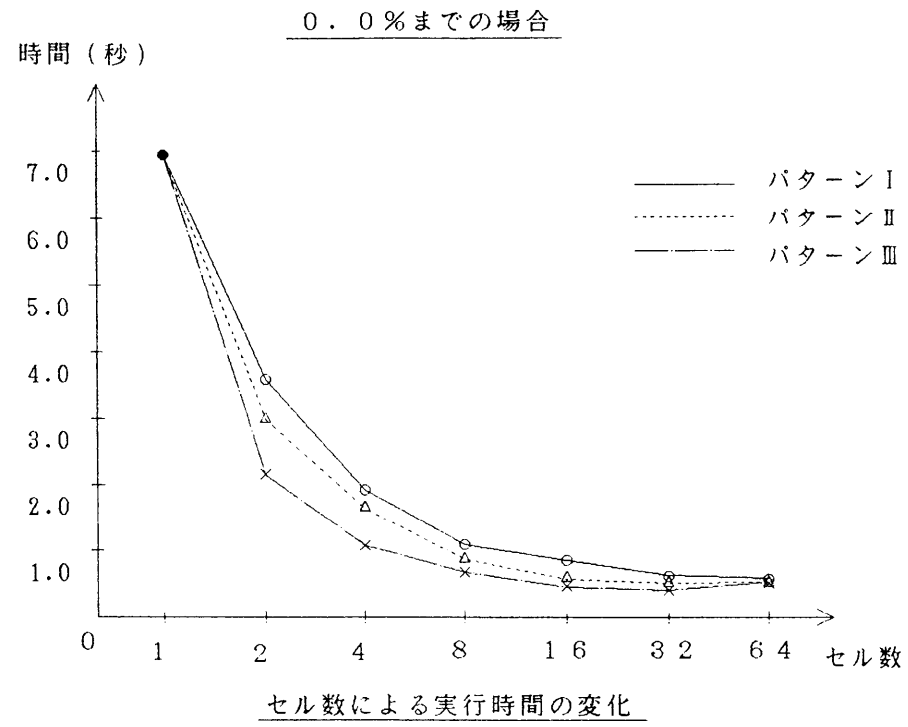
粒度が小さいとき(表7.1と図5.1)，並列度が増す(セル数が大きくなる)と，かえって実行速度が低下している。これは，多くのセルからのメッセージを処理するために，同期と通信のための処理時間の割合が増えることによる。

一方，最適解を求める場合(図5.0)には，パターンIIIで速度向上比が良い結果を示している。これは，良い遺伝子を注入するという方針が効果を表わしているものと考えられる。特に，プロセッサ数が2，4，8では，理想直線よりも良い向上比となっているが，並列処理を行うことの特徴である。並列処理を行うに当っては，この例のように，プロセッサ数以上の効果を発揮するような問題解決法を考えたい。

8．おわりに

最近，一部で，注目されつつある遺伝的アルゴリズム(GA)をいくつかの問題に適応し，それぞれの問題に適した変更・改良を加えた。決定性のアルゴリズムとは違って，必ずしも最適解が求まるという保証はないが，ある限定された範囲の近似解が，素早く(世代数が少くても)見つかる特徴をもつことが確認された。また，さまざまな問題に対して，遺伝子の表現方法を工夫する必要はあるが，全く同じ計算手順で近似解を得ることができる。そのため，はじめにも述べたように，新しい計算の枠組みを与えてくれるものと期待される。それには，今後，もっと違ったタイプの問題(たとえば，連立一次方程式，推論，文字認識等)にGAを応用することと共に，GAを適用するときの高級「指示言語」を設計して，応用しやすくすることが必要となる。

一方，GAは，並列処理におけるひとつの良い型を示唆しているとも考えられる。ある程度独立



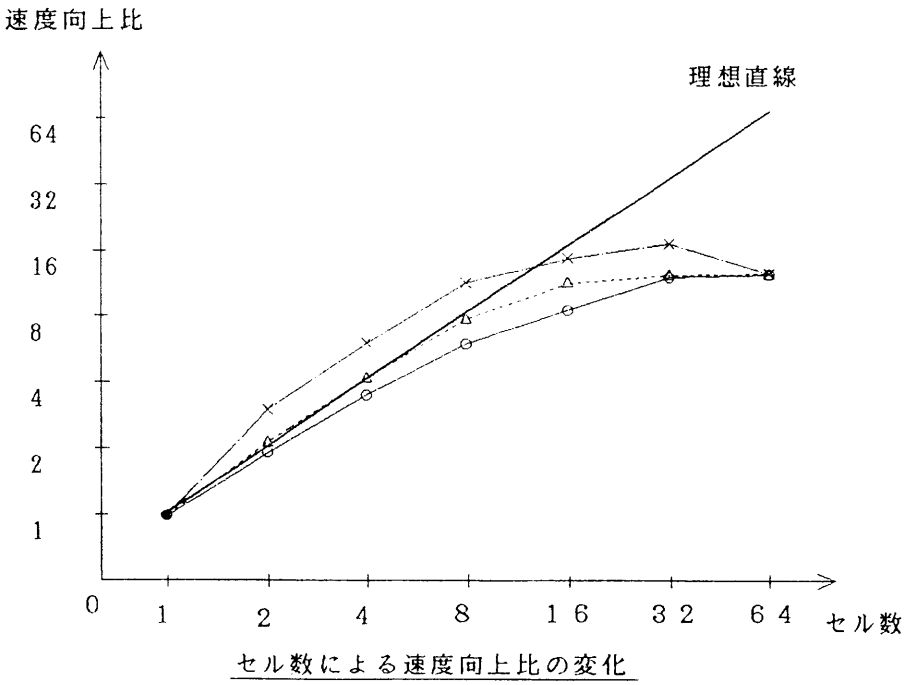
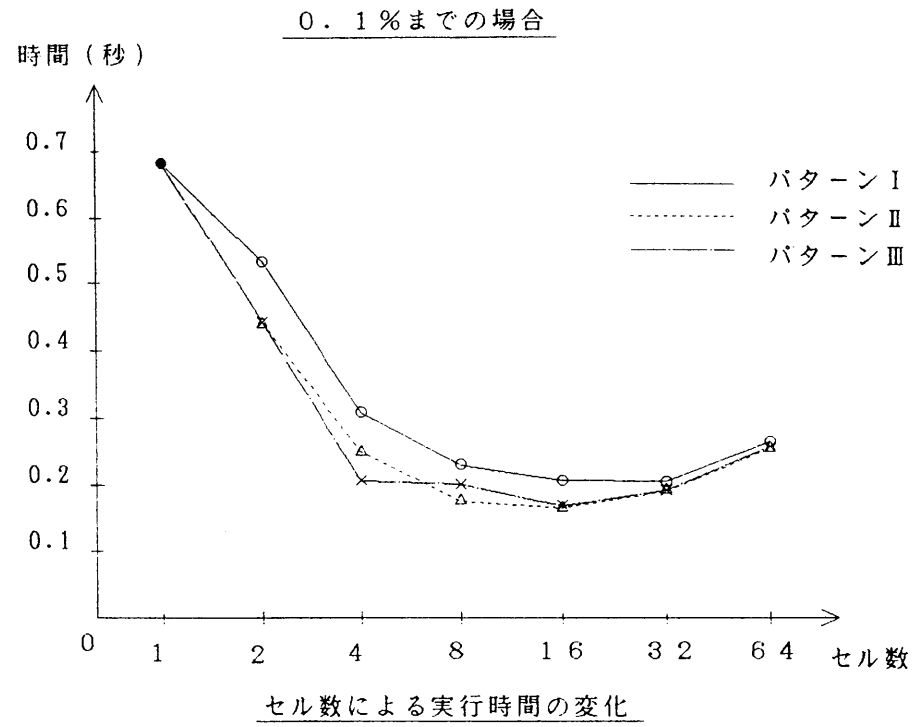


図5.0　AP1000により最適解を求めるまでの実行時間と速度向上比



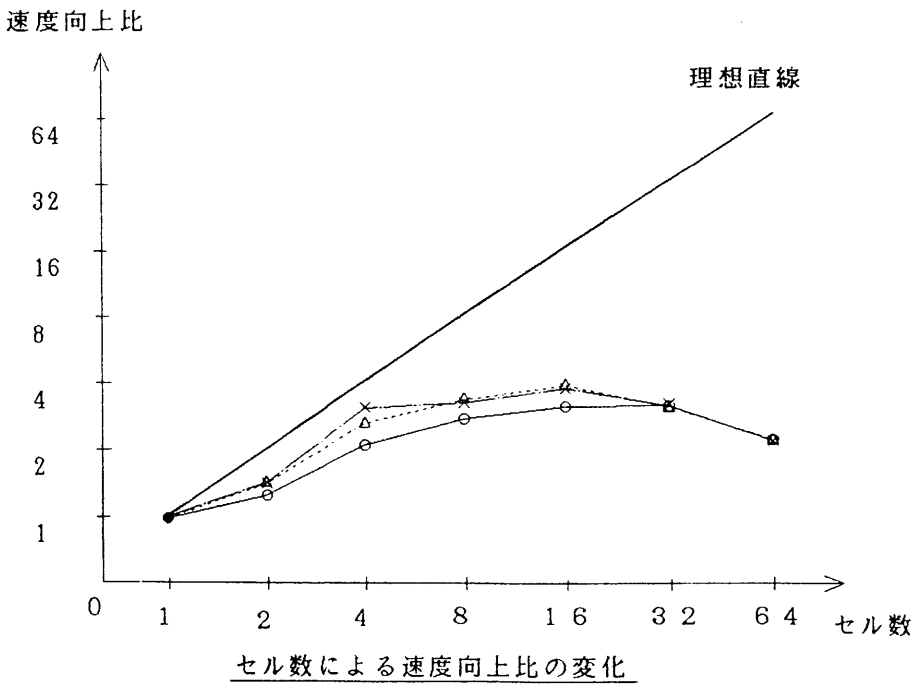


図5.1　AP1000により近似解を求めるまでの実行時間と速度向上比

性をもって計算を進めるセル(プロセッサ)が，互いに自分の知らない良い中間結果を交換しあうことにより，次のステップの計算に，かなり良い効果を及ぼし合うというものである。逐次処理とは違った，並列処理特有の問題解決法のひとつとなることが期待される。

謝　　辞

GAを並列アルゴリズムとして実現するに当り，富士通研究所のAP1000を，長時間にわたり使用させていただいた。すばらしい計算環境を提供くださいました関係各位に感謝いたします。

参考文献

1) 宮沢文夫：遺伝的アルゴリズムと最適化問題，ASCII，Vol.15，No.6 & 7（June&July，1991）．p.301～p.308 & p.325～p.332.

2) 和田健之介：遺伝的アルゴリズムと機械の進化，数理化学，No.328（1991年10月）p.47～51.

3) A.K.デュードニー：flibがすむ原始のコンピュータの海で遺伝的アルゴリズムを探検する。日経サイエンス（1986年1月）p.116～122.