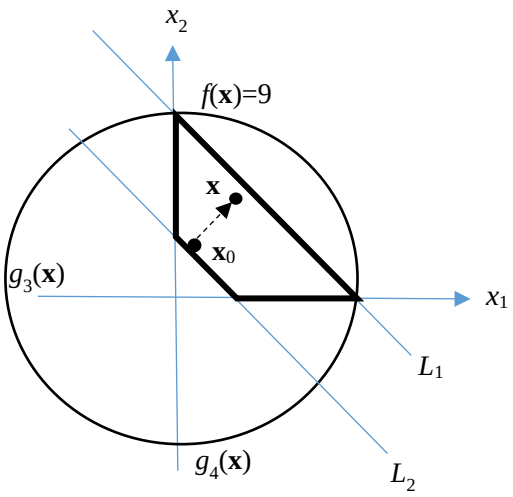
人工知能―AI の基礎から知的探索へ：演習問題解答例

第 8 章　知的探索



演習問題 8.1 の図解

演習問題 8.1　例題 8.1 と同じ目的関数と制約条件の最大化問題を考える。このとき、実行可能領域 D から生成した初期解が大局的最適解ではなければ、最急降下法で大局的最適解を得ることが不可能である。この結論の理由を、図 8.1 を用いて説明せよ。

解答　実行可能領域 D は、太い枠の中の領域である。任意の初期値 $\mathbf{x}_0$ からスタートして、最急降下法で求めた解は、制約 $g_1(\mathbf{x})$に対応する直線 L_1 上の点となる。L_1 上のすべての点 $\mathbf{x}$ に対して、$f(\mathbf{x})\leq 9$ となり、イコールは L_1 と大きい円との交点だけにおいて成立する。したがって、この問題に関しては、$\mathbf{x}_0$ は最適解ではなければ、最急降下法で最適解を求めることができない。

演習問題 8.2　組み合わせ問題の良い例として、ナップザック問題がある。この問題の意味をインターネットで調べ、問題を TS で解決する方法について検討せよ。（ヒント：例題 8.2 のように、状態ベクトルの表現方法、初期状態の生成方法、Candidate List の生成方法などについて説明できれば良い）。

解答　ナップザック問題は、「容量 C のナップザックがあり、n 種類の品物が与えられたとき、容量 C を超えないで $m(\leq n)$個の品物をナップザックに詰め、詰めた品物の価値の和を最大化する」問題である。ただし、各々の品物の価値と大きさ（必要とされる容積）が与えられたとする。

ナップザック問題を解決するためには、まず状態ベクトルを以下のように表現する：

$$\mathbf{x} = [x_1, x_2, ..., x_n]$$

ただし、x_j は 0 か 1 しか取らない整数である。x_j=1 のときだけ j 番目の品物をナップザックに入れる。現在状態は **x** であるとし、Candidate List は **x** の要素を「反転」することによって得られる。例えば、

x=[1 1 **0** 0 1 0 0 1]

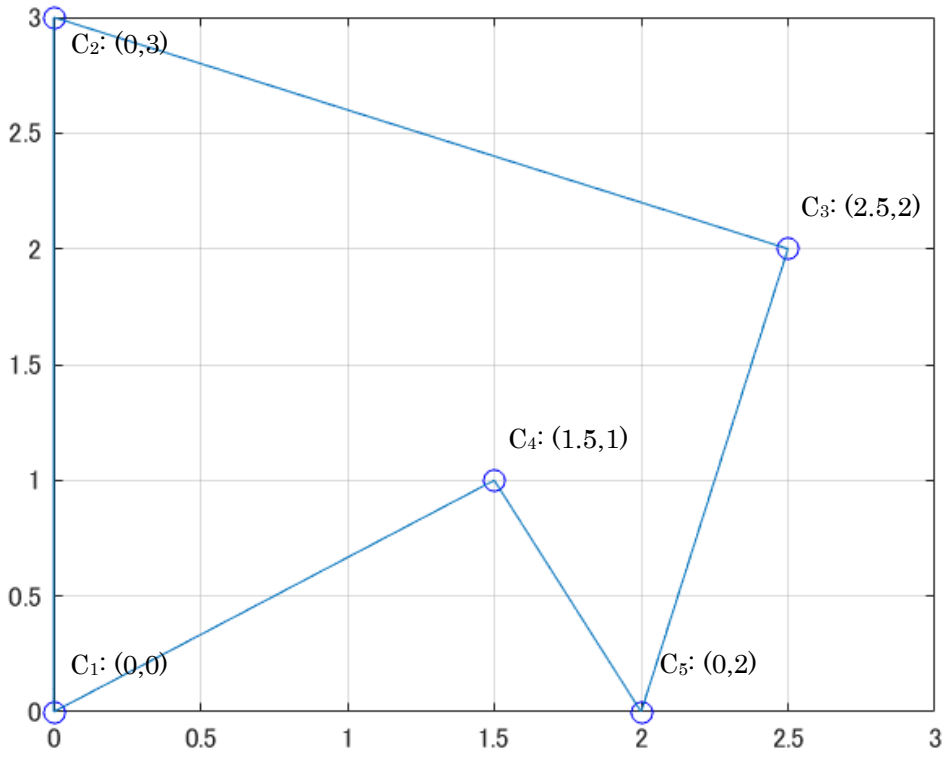
の場合、x_3 を反転すると、

x’=[1 1 **1** 0 1 0 0 1]

が得られる。このように得られた状態は、実行可能解ではないかもしれない。すなわち、品物のトータルサイズが C を超えてしまう可能性がある。通常、複数個の解をランダムに生成してから、実行可能解ではないものを捨て、残りの解だけを Candidate List に入れる。この中からさらに、Tabu List に記憶されている可能解を外す。次の解 **x**’は、Candidate List の中で目的関数値（総価値）を最大にする可能解である。

TS において、**x**’が **x** に比べて、良くなっても悪くなっても次の状態となる。また、**x**’が $\mathbf{x}_{best}$ よりもよければそれを新しい $\mathbf{x}_{best}$ とする。このように繰り返せば、そのうち、良い解が得られる。「十分良い解」を得たら探索が終了できる。十分良い解を知らない場合は、例えば、探索回数を指定することができる。

演習問題 8.3　図 8.3 は、五つの都市の座標を与えている。各都市間のユークリッド距離を計算せよ。また、計算した距離をもとに、巡回セールスマン問題（TSP）を解け。ただし、1 番目の都市はホームであるとする。



5 つ都市 TSP 問題のマップとその最適解

解答　5 つの都市間のユークリッド距離は、以下のマトリックスで与える：

$$D = \begin{bmatrix} 0 & 3.0000 & 3.2016 & 2.0000 & 3.6056 \\ 3.0000 & 0 & 1.8028 & 2.6926 & 1.4142 \\ 3.2016 & 1.8028 & 0 & 2.5000 & 2.0616 \\ 2.0000 & 2.6926 & 2.5000 & 0 & 1.1180 \\ 3.6056 & 1.4142 & 2.0616 & 1.1180 & 0 \end{bmatrix}$$

ただし、D の (i,j) 番目の要素 d_{ij} は Ci 番目の都市と Cj 番目の都市間の距離である。

初期解は、以下のように与えられたとする：

$\mathbf{x}_0=[n(1), n(2), n(5), n(4), n(3)]$

この $\mathbf{x}_0$ の評価値は 11.2338 である。この時点では $\mathbf{x}_{best}=\mathbf{x}$ で、Tabu List=$\{\mathbf{x}_0\}$である。

次に、2 つの可能解を含む Candidate List を作る。ノードの入れ替えを一回行って得られる可能解の集合 $N_1(\mathbf{x})$から、以下の解が得られたとする：

$\mathbf{x}_1=[n(1), n(5), n(2), n(4), n(3)]$
$\mathbf{x}_2=[n(1), n(2), n(3), n(4), n(5)]$

それぞれの評価値は、13.4139 と 12.0264 である。$\mathbf{x}_2$ が良いほうなので、$\mathbf{x}'=\mathbf{x}_2$ とする。この $\mathbf{x}'$は次の状態 $\mathbf{x}$ となる。しかし、$\mathbf{x}'$は $\mathbf{x}_{best}$ より悪いので、$\mathbf{x}_{best}$ が変わらない。この時点では Tabu List=$\{\mathbf{x}_0,\mathbf{x}_2\}$である。

次は、第 2 回の反復探索になる。まず、$\mathbf{x}=[n(1), n(2), n(3), n(4), n(5)]$の近傍 $N_1(\mathbf{x})$から、以下の２つの解を求めたとする：

$\mathbf{x}_3=[n(1), n(3), n(2), n(4), n(5)]$
$\mathbf{x}_4=[n(1), n(2), n(3), n(5), n(4)]$

評価値はそれぞれ、12.4205 と 9.9824 である。したがって、$\mathbf{x}'=\mathbf{x}_4$ となり、これは次の状態 $\mathbf{x}$ となる。また、評価値が現在のベストより良いので、$\mathbf{x}_{best}=\mathbf{x}_4$ と更新する。このように繰り返せば、最適解を求めることができる。実際、$\mathbf{x}_4$ はすでに最適な解である。

表 8.5　SA アルゴリズム

Step 1:	初期化：
	• ランダムに $\mathbf{x}_0$ を生成し、$\mathbf{x}$ に代入する。
	• 初期温度τ_0 をτに代入する。
Step 2:	終了条件：
	• $f(\mathbf{x})$が終了条件を満たせば、$\mathbf{x}$ を返し、終了する。
Step 3:	新しい状態の生成：
	• $N(\mathbf{x})$からランダムに $\mathbf{x}'$を生成する。
Step 4:	状態遷移：
	• $f(\mathbf{x}') < f(\mathbf{x})$ ならば $\mathbf{x}$ に $\mathbf{x}'$を代入する;
	• そうではなければ、確率 $p(\mathbf{x}'\|\tau)$で $\mathbf{x}$ に $\mathbf{x}'$を代入する;
Step 5:	温度の更新：
	• 平衡状態になったら、τを更新する。
Step 6:	Step 2 へ戻る;

演習問題 8.4　ナップザック問題を考える。ナップザックの容積は C=50、品物の数は 5、対応する価値とサイズは、下の表で与えられたとする：

	1	2	3	4	5
価値 p	20	10	50	25	15
サイズ c	10	20	10	30	40

この問題を、SA で解決する過程を示せ。

解答　問題 8.2 の回答にしたがって、状態ベクトルを以下のように表現する：

$$\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$$

ここで、x_j は 0 か 1 しか取らない整数である。表 8.5 に基づいて、ナップザック問題を解決するための matlab プログラムを作成し、次のようになる。

```
% Program for solving the knapsack problem using simulated annealing

p=[20   10   50   25   15]; % price of each item
c=[10   20   10   30   40]; % size of each item
```

```
L=50;                          % limitation (capacity) of the knapsack
s = [0 0 1 1 0];               % initial state (solution)
Iteration=2;                   % number of iterations for each T value
desired_value=90;

fprintf('s0 = [%s]¥n',num2str(s));
[P,C] = f(s,p,c); % evaluate the solution
fprintf('P = %d,    C = %d¥n', P,C);

s_best = s; p_best = P;

% Repeat for different temperatures, start from a high value, finish with a
% small enough value

T_start=10; T_end=1; T_step=-1;
for T=T_start:T_step:T_end
      fprintf('Current best solution = [%s]¥n',num2str(s_best));
      fprintf('P_best = %d¥n',p_best);

      s=s_best; % starting search from the currect best solution
      P=p_best;

      if p_best>=desired_value
            disp('Best solution found!'); break
      end

      for i=0:Iteration
            s_new=N(s); % find a neighbor of the current solution
            [P_new,C]=f(s_new,p,c); % evalute the neighbor
            while C>L
                  s_new=N(s); [P_new,C]=f(s_new,p,c);
            end % repeat until a feasible solution is found
            fprintf('new solution = [%s]¥n',num2str(s_new));
            fprintf('P_new = %d; C_new = %d¥n',P_new,C);

            if P_new > p_best
                  s_best=s_new; p_best=P_new;
            end % update the best solution if needed

            % find probability for replacing the old state
            prob=1;
            if P_new < P
                  prob=exp((P_new-P)/T);
            end
            fprintf('Probality for updating the state = %f¥n',prob);

            if rand <= prob
                  disp('state updated!'); s=s_new; P=P_new;
            end
      end
end
```

このプログラムを実行すると、以下の結果が得られる。

```
s0 = [0  0  1  1  0]
P = 75,  C = 40
Current best solution = [0  0  1  1  0]
P_best = 75
new solution = [0  1  1  0  0]
P_new = 60; C_new = 30
Probality for updating the state = 0.223130
new solution = [0  0  1  1  0]
P_new = 75; C_new = 40
Probality for updating the state = 1.000000
state updated!
new solution = [1  1  1  0  0]
P_new = 80; C_new = 40
Probality for updating the state = 1.000000
state updated!
Current best solution = [1  1  1  0  0]
P_best = 80
new solution = [1  0  0  1  0]
P_new = 45; C_new = 40
Probality for updating the state = 0.020468
new solution = [0  1  1  0  0]
P_new = 60; C_new = 30
Probality for updating the state = 0.108368
new solution = [1  0  1  1  0]
P_new = 95; C_new = 50
Probality for updating the state = 1.000000
state updated!
Current best solution = [1  0  1  1  0]
P_best = 95
Best solution found!
```

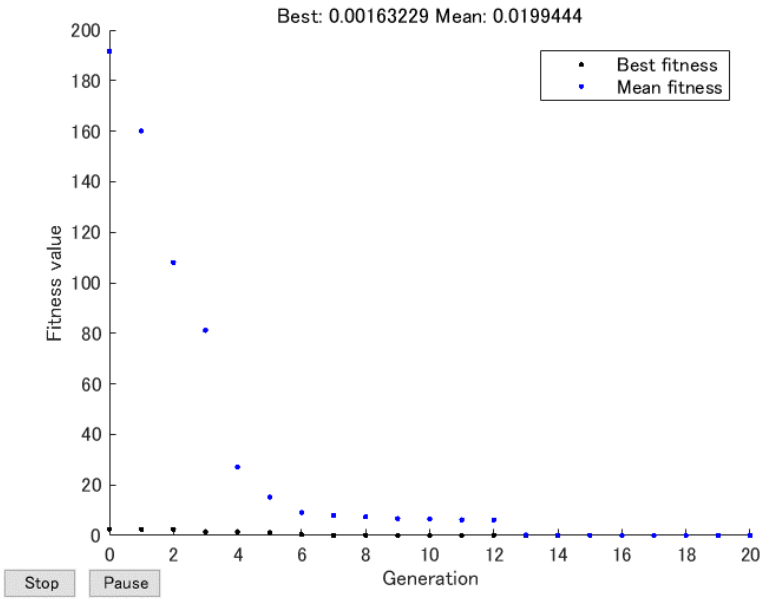
この例は、SA の使い方だけを示しているが、SA の利点を示すためには、より規模の大きい問題を解いてみてください。また、プログラムを理解するために、必要に応じてコメントを入れてみてください。

最小化を GA で行うための Matlab プログラム

```
1.  F=@(x)(x(1)^2+x(2)^2);
2.  options=gaoptimset('Generations',20,'PopulationSize',20,'PlotFcns',@gaplotbestf);
3.  lb=[-20, -20];    % lower bound of (x1,x2)
4.  up=[20, 20];      % upper bound of (x1,x2)
5.  x=ga(F6,2,[],[],[],[],lb,up,[],options)
```

演習問題 8.5　表 8.7 にある目的関数 F6 を、F=@(x)(x(1)^2+x(2)^2)に入れ替え、GA で関数の最小値を求めよ。また、その結果と実の最適値とを比較し、GA の性能について議論せよ。

解答　プログラムの第一行だけを変更すると、上の表となる。一つの実行例は、下の図に示す。実際の最適値は、x=0 なので、20 世代で得られた最適値は、これに近い値となっている。世代数や集団サイズなどを大きくすることによって、より良い解が得られる。興味がある方は、上記のプログラムにあるパラメータを変更して、実行してみると良い。



最小化を PSO で行うための Matlab プログラム

```
1.  F=@(x)(x(1)^2+x(2)^2);
2.  options = optimoptions(@particleswarm,'SwarmSize',20,'PlotFcns',@pswplotbestf);
3.  lb = [-20,-20];
4.  ub = [20,20];
5.  x=particleswarm(F6,2,lb,ub,options)
```

演習問題 8.6　表 8.9 にある目的関数 F6 を、F=@(x)(x(1)^2+x(2)^2)に入れ替え、PSO で関数の最小値を求めよ。また、その結果と実の最適値とを比較し、PSO の性能について議論せよ。

解答　プログラムの第一行だけを変更すると、上の表となる。一つの実行例は、下の図に示す。実際の最適値は、x=0 なので、得られた最適値は、これに非常に近い値となっている。この問題に限って言えば、GA に比べると、PSO の解の質が非常に良いと言える。

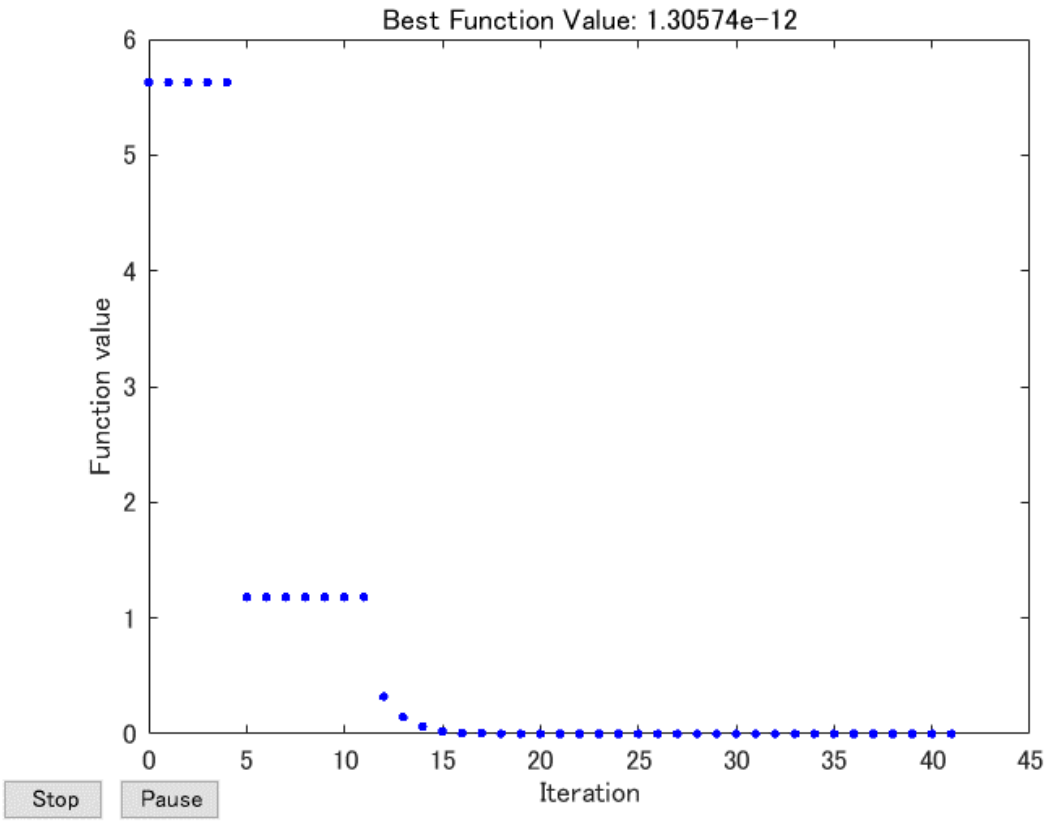


表 8.11　正規化された 10 個の都市の座標

1	0.0833	0.6379	6	0.8045	0.6064
2	0.5111	0.0161	7	0.8169	0.7604
3	0.3668	0.8960	8	0.1895	0.8553
4	0.7395	0.5154	9	0.1237	0.3829
5	0.5247	0.5445	10	0.8210	0.0846

演習問題 8.7　表 8.11 には、１０個の都市の座標（正規化されたもの）を表している。例題 8.6 と同じように、Johannes が開発した Ant System TSP Solver を利用して１０都市 TSP 問題を解け（ヒント： Djibouti.m をテンプレートとして、上記の座標を使って、TenCity.m と言うファイルを作る）。

解答　Ant_system_TSP_Solver のホルダーの中に、以下のように TenCity.m を作成する：

```
function x = TenCity()
% NAME : FiveCity
% COMMENT : Homework8_7
x = [
 1   0.0833    0.6379
 2   0.5111    0.0161
 3   0.3668    0.8960
 4   0.7395    0.5154
 5   0.5247    0.5445
 6   0.8045    0.6064
 7   0.8169    0.7604
 8   0.1895    0.8553
 9   0.1237    0.3829
 10  0.8210    0.0846
];

x = x(:,2:3);
```

そして、matlab の環境で、以下のコマンドを打ち、プログラムを実行する。

```
[popt, fopt] = ant_system_tsp(@TenCity,200,20)
```

ここで、popt は得られたパスで、fopt はそのパスの長さ、200 は探索回数、20 はアリの数である。実行結果は、以下のようになる：

```
popt = [ 6 7 5 3 8 1 9 2 10 4]
fopt = 2.9863
```

下の図の左上の部分は、現在状態の「進化」過程を示し、右上の部分は得られた最適パスである。下の 2 つの図は、それぞれフェロモン行列とヒューリスティック行列を可視化したものである。