Programare evolutiva si algoritmi genetici

Activitate 4 – Problema Comis-Voiajorului (TSP)

2016

Cuprins

- Enunt
- Rezolvare – abordare genetica
- Implementare
- Exemple de rulare
- Exercitii

Enunt

- Fie:
  - n oraşe interconectate două câte două
  - şi D matricea costurilor de deplasare între oraşe: $\forall i, j = 1, \dots, n, D(i, j)$ reprezintă costul trecerii directe de la oraşul i la oraşul j.
- Un comis-voiajor trebuie să facă livrări în toate oraşele date, plecând dintr-un oraş i oarecare şi reîntorcându-se în i.
- Problema este de a găsi o ordine de parcurgere a oraşelor astfel încât costul transportului să fie minim.

Rezolvare – abordare genetica (1a)

- O configuraţie soluţie (fenotip) este reprezentată prin intermediul unei permutări cu n elemente, x, costul parcurgerii oraşelor conform lui x fiind:

$$cost(x) = D(x(n), x(1)) + \sum_{i=1}^{n-1} D(x(i), x(i+1))$$

În scopul obţinerii unei probleme de maxim => funcţia obiectiv definită pe spaţiul genotipurilor este:

$$castig(x) = \frac{1}{cost(x)}$$

Rezolvare – abordare genetica (1b)

Reprezentare posibile solutii prin permutari

- fie matricea distantelor D ->

1	0	8	10	15
2	8	0	4	4
3	10	4	0	7
4	15	4	7	0

- si fie generata o permutare ca si posibila solutie: c=[3 1 2 4].
- se doreste calcularea functiei obiectiv

```
1   function [val] =f_obiectiv(c,D)
2   [~,n]=size(c);
3   val=D(c(1),c(n));
4   disp(val); %eu
5   for i=1:n-1
6       val=val+D(c(i),c(i+1));
7       disp(c(i));
8       disp(
9    end;
10  val=1/val;
11  end
```

```
Pop_curenta =

    3.0000    1.0000    2.0000    4.0000    0.0345
    3.0000    1.0000    2.0000    4.0000    0.0345
    1.0000    4.0000    2.0000    3.0000    0.0303
    1.0000    3.0000    2.0000    4.0000    0.0303
```

Exemplu de populatie

Rezolvare – abordare genetica (1c)

Date de intrare:

- fie matricea distantelor D ->

1	0	8	10	15
2	8	0	4	4
3	10	4	0	7
4	15	4	7	0

- si fie generata o permutare ca si posibila solutie: c=[3 1 2 4].

Calcularea valorii functiei obiectiv pentru permutarea generata:

- n=4 (dimensiunea cromozomului)
- val=D (c(1), c(4))=> val = D(**3**,4) (in permutare) = distanta dintre orasul 3 si 4 din matricea D=> **val = 7**
- pentru i=1:3
- i=1=> val =val +D(c(1),c(2))=7+D(3,1)=7+10 (distanta de la 3 la 1) => **val=17**
- i=2 => val=val +D(c(2),c(3))=17+D(1,2)=17+8=> **val=25**
- i=3 => val=val + D (c(3), c(4)) = 25 + D(2,4) = 25+4 => val=29

val_permutare=1/val=0,0345

Setari aplicatie

Rezolvare - abordare genetica (2)

Setari

1. spaţiul genotipurilor: **permutări**

2. populaţia iniţială este generată aleator, dimensiunea populaţiei este ***dim***;

3. selecţia părinţilor:

- pe baza algoritmului SUS (**S**tochastic **U**niversal **S**ampling), cu probabilitate de selecţie din modelul FPS cu sigma–scalarea = 2;

4. operatorul de **recombinare**:

- PMX (Partially Mapped Crossover) cu pc data ca parametru de intrare (de ex. 0.8)

5. operatorul **mutaţie prin inversiune**: inversiunea unui subşir aleator într-un cromozom, cu probabilitatea pm data ca parametru de intrare $(pm \geq \frac{1}{dim})$;

6. condiţii de oprire:

- a fost depăşit un număr maxim de iteraţii in simularea evolutiei
- a fost depasit numarul maxim de generatii consecutive cu proprietatea ca valoarea functiei obiectiv nu este modificata;

7. schimbul de generaţii:

- pe baza calităţii indivizilor, prin aplicarea **unui algoritm de tip elitist**, care asigură „propagarea” celui mai bun individ al populaţiei curente, dacă acesta este mai bun decât oricare dintre progeniturile generate.

Selectarea parintilor

3.Selectia parintilor – algoritm SUS* (Pb Comis-Voiajorului) (1a)

- ***S**tochastic **U**niversal **S**ampling aplica principiul FPS - asigura construcţia unui bazin de recombinare cât mai apropiat de o instanţă a distribuţiei de probabilitate a selecţiei

- Principiu FPS: **F**itness **P**roportional **S**election:
  - pentru fiecare cromozom $y_i, 1 \leq i \leq \mu$, este calculată performanţa sa, prin funcţia de evaluare $f_i = f(y_i)$. Probabilitatea de selectare a individului y_i pentru încrucişare este

$$p_i = \frac{f_i}{\sum_{i=1}^{\mu} f(y_i)}$$

- Probabilitatea de selectare a individului depinde de:
  - valoarea absolută a calităţii individuale comparativ cu valoarea absolută a calităţii populaţiei curente.

```
Pop_curenta =

    3.0000    1.0000    2.0000    4.0000    0.0345
    3.0000    1.0000    2.0000    4.0000    0.0345
    1.0000    4.0000    2.0000    3.0000    0.0303
    1.0000    3.0000    2.0000    4.0000    0.0303
```

3.Selectia parintilor – algoritm SUS* (Pb Comis-Voiajorului) (1b)

Inconveniente FPS:

- indivizii mult superiori restului populaţiei (din punct de vedere al valorii funcţiei de evaluare) vor domina întreaga populaţie într-un timp relativ scurt (*convergenţă prematură*);

- **dacă indivizii din populaţie sunt a.î. valorile funcţiei de evaluare sunt comparabile**
  - atunci nu există aproape nici o *constrângere de selecţie*, iar selecţia urmează legea de repartiţie uniformă;
  - indivizii cu valori ale funcţiei de evaluare uşor superioare restului populaţiei nu vor fi favorizaţi => pe parcursul evoluţiei GA, când indivizii mai slabi dispar şi începe să se instaleze procesul de convergenţă, performanţa indivizilor generaţiei curente nu este semnificativ îmbunătăţită;

3.Selectia parintilor – algoritm SUS (Pb Comis-Voiajorului) (1c)

- variantă **sigma-scalarea** care încorporează informaţiile **medie de selecţie** şi **respectiv deviaţie standard de selecţie** calculate pentru exemplele populaţiei curente :

$$\bar{f}_t = \frac{1}{|\mathcal{P}^t|} \sum_{y \in \mathcal{P}^t} f(y)$$

$$\sigma^2_{f,t} = \frac{1}{|\mathcal{P}^t| - 1} \sum_{y \in \mathcal{P}^t} \left(f(y) - \bar{f}_t\right)^2$$

$$g(y) = max\left(f(y) - \left(\bar{f}_t - c \cdot \sigma_{f,t}\right), 0\right)$$

unde: c este un parametru dat (de obicei $c = 2$)

Pt – populatia totala (nr de indivizi)

3.Selectia parintilor – algoritm SUS (Pb Comis-Voiajorului) (1e)

- Algoritm SUS:

Pas 1. Pentru fiecare cromozom $y_i, 1 \leq i \leq \mu$:

1.1 evaluează performanţa sa, prin funcţia de evaluare $f_i = f(y_i)$;

1.2 calculează probabilitatea de selecţie din mecanismul de tip FPS sau cea bazată pe ranguri, p_i şi probabilitatea cumulată $q_i = \sum_{j=1}^{i} p_j$

Pas 2. generează aleator un număr $r \in [0, 1/\mu]$; i=1; k=1;

Pas 3. Cât timp $k \leq \mu$ execută 3.1 şi 3.2

3.1 cât timp $r \leq q_i$ execută 3.1.1, 3.1.2 şi 3.1.3

3.1.1 parinte(k)=y_i;

3.1.2 $r = r + 1/\mu$;

3.1.3 $k = k + 1$;

3.2 $i = i + 1$

r este generat aleator în intervalul $[0, 1/\mu]$ şi după fiecare selecţie efectuată este mărit cu $1/\mu$

=> nr. de copii al fiecărui cromozom y_i în bazinul de recombinare este cel puţin $[\mu \cdot p_i]$ şi cel mult $[\mu \cdot p_i] + 1$, unde $[x]$ este partea întreagă a numărului real *x*.

3.Selectia parintilor – algoritm SUS (Pb Comis-Voiajorului) (1)

Populatie initiala

4.0000	3.0000	1.0000	2.0000	0.0345
3.0000	1.0000	2.0000	4.0000	0.0345
4.0000	1.0000	2.0000	3.0000	0.0294
1.0000	2.0000	4.0000	3.0000	0.0345
2.0000	3.0000	1.0000	4.0000	0.0303

Rezultat algoritm SUS

medie= 0.0326
sigma= 0.0026 (dev. std)
val=medie-c*sigma
val= 0.0275
//calcul sigma-scalarea
g[1]= 0.0070
g[2]= 0.0070
g[3] = 0.0019
g[4] = 0.0070
g[5] = 0.0028

Calcul suma elemente vector g = 0.0255

Distrib. de probab pentru generarea de parinti (p=g/s):
p[1] = 0.2725
p[2] = 0.2725
p[3] = 0.0738
p[4] = 0.2725
p[5] = 0.1087

Calcul vector q (q(i)=sum(p(1:i))):
q[1]= 0.2725
q[2]= 0.5449
q[3] = 0.6188
q[4] = 0.8913
q[5]= 1.0000

r generat aleator =
0.1699

3.Selectia parintilor – algoritm SUS (Pb Comis-Voiajorului) (2)

```
while (i<=dim)
    while(r<=q(j))
        Parinti(i,:)=Pop(j,:);
        i=i+1;
        r=r+1/dim;
    end;
    j=j+1;
end;
```

Calcul vector q (
q(i)=sum(p(1:i)))=
q[1]= 0.2725
q[2]= 0.5449
q[3] = 0.6188
q[4] = 0.8913
q[5]= 1.0000

r generat aleator =
0.1699

Din populatia initiala => populatia de parinti

Operatorul de recombinarea

4. Operatorul de **recombinare (1)**

- Recombinarea (încrucişarea)
  - procesul creării de noi soluţii candidat pe baza informaţiei conţinute în două (sau mai multe) soluţii candidat părinţi
  - PMX (**P**artially **M**apped Crossover)

4. Operatorul de **recombinare (2)**

- Exemplu

fie $m = 10, p_1 = 4, p_2 = 7,$

x_1	1	2	4	6	5	7	3	9	10	8
y_1	10	5	3	7	9	2	1	8	6	4

După aplicarea primului pas, obţinem

x_2				6	5	7	3			

La pasul 2 rezultă $A = \{(9,5), (2,6), (1,7)\}$

4. Operatorul de **recombinare (3)**

La pasul 3: $a = 9, p = 5, b = x_2(5) = 5$

Deoarece $y_1(2) = 5$ şi gena 2 din x_2 nu are încă valoare, setează $x_2(2) = 9$. Obţinem

x_2		9		6	5	7	3			

$a = 2, p = 6, b = x_2(6) = 7$

Deoarece $y_1(4) = 7$ şi gena 4 din x_2 are valoarea $x_2(4) = 6$, rezultă $c = 6, y_1(9) = 6,$ deci $j = 9$ şi setează $x_2(9) = 2$. Obţinem

x_2		9		6	5	7	3		2	

$a = 1, p = 7, b = x_2(7) = 3$

Deoarece $y_1(3) = 3$ şi gena 3 din x_2 nu are încă valoare, setează $x_2(3) = 1$. Obţinem

x_2		9	1	6	5	7	3		2	

La pasul 4 vor fi plasate în x_2, în ordinea genelor rămase fără valori, alelele 10, 8 şi 4 (alelele din y_1 ce nu au fost copiate încă în prima progenitură). Rezultă

x_2	10	9	1	6	5	7	3	8	2	4

Operatorul de mutatie

5. Operatorul de mutatie (1)

- Operatorul mutatie (OM):
  - operatorul genetic prin care, dintr-un singur parinte este obtinut un singur copil prin aplicarea unei modificari aleatoare a reprezentarii (genotipului)

  - Exemplu mutatie prin inversiune
    - parinte x1= 4 3 **1 7 8 5** 2 6
    - pozitii 3 si 6

    - => copil x2=4 3 **5 8 7 1** 2 6

Schimbul de generatii

7. Schimbul de generaţii (1)

- Elitism - evitarea pierderii celor mai bine adaptaţi indivizi la schimbul de generaţii.

- Din Populatia curenta si din Poluatia de copii => Populatia urmatoare

7. Schimbul de generaţii (2)

Aplicarea strategiei de elitism:

- fie populatia curenta si populatia de copii (fiecare cu valoarea functiei obiectiv)

```
Pop_curenta =

    3.0000    1.0000    2.0000    4.0000    0.0345
    3.0000    1.0000    2.0000    4.0000    0.0345
    1.0000    4.0000    2.0000    3.0000    0.0303
    1.0000    3.0000    2.0000    4.0000    0.0303
```

```
Copii =

    3.0000    1.0000    2.0000    4.0000    0.0345
    3.0000    1.0000    2.0000    4.0000    0.0345
    1.0000    4.0000    2.0000    3.0000    0.0303
    1.0000    3.0000    2.0000    4.0000    0.0303
```

- identificam maximul pentru Pop_curenta (retinem maximul si linia):
  - val1=0.0345 pe pozitia i1=1
- identificam maximul din Copii (retinem valoarea):
  - v2=0.0345
- daca val1>val2 (nu e cazul aici) => atunci linia din Pop_curenta trece in Pop_urm inlocuind o pozitie aleator

```
best= Pop_curenta(i1,:)
ind=unidrnd (dim) %inlocuiesc un copil aleator
Pop_urm(ind,: )=best
```

Implementare

Implementare

- Functii:
  - **function [val] =f_obiectiv(c,D)**
  - **function [y]=gen_perm(m);**
  - **function [Pop,D] = gen_pop_permutari(dim,nume)** // *generarea populatiei initiale*
  - **function [Parinti] =SUS(Pop)** //*generare parinti*
  - **function [c1,c2] = PMX(x1,y1)**
  - **function [Copii] = crossover(Pop,pc,D)** //*generare copii prin recombinare parinti*
  - **function [y] = inversiune(x)**
  - **function [CopiiM] = mutatie(Pop,pm,D)** //*generare copil dintr-un parinte*
  - **function [Pop_urm] =elitism(Pop_curenta,Copii)** //*calcul generatie urmatoare*
  - **function [cost,solutie] =GA_TSP(dim,numefis,pc,pm,MAX,NRG)//calcul problema TSP**
  - + *fisiere cu distante*

Implementare (1)

```
function [val] =f_obiectiv(c,D)
[~,n]=size(c);
val=D(c(1),c(n));
for i=1:n-1
   val=val+D(c(i),c(i+1));
end;
val=1/val;
end
```

```
function [y]=gen_perm(m);
y=zeros(1,m);
for i=1:m
   gata=0;
   while(~gata)
      v=unidrnd(m);
      if(~ismember(v,y))
         y(i)=v;
         gata=1;
      end;
   end;
end;
end
```

```
function [ Pop,D ] =
gen_pop_permutari(dim,nume)
D=load(nume);
[~,n]=size(D);
Pop=zeros(dim,n+1);
for i=1:dim
   Pop(i,1:n)=gen_perm(n);
   Pop(i,n+1)=f_obiectiv(Pop(i,1:n),D);

end;
end
```

Implementare (2)

```
function [Parinti] =SUS(Pop)
[dim,m]=size(Pop);
n=m-1;
%SUS implementat pe distributia de probabilitate FPS cu
sigma-scalare, c=2
c=2;
medie=mean(Pop(:,n+1));
sigma=std(Pop(:,n+1));
val=medie-c*sigma;
g=zeros(1,dim);
for i=1:dim
   g(i)=max([Pop(i,n+1)-val 0]);
end;
s=sum(g);
%p este distributia de probabilitate dupa care se face
generarea populatiei
%de parinti
p=g/s;
q=zeros(1,dim);
for i=1:dim
   q(i)=sum(p(1:i));
end;
```

```
%generarea populatiei de parinti conform SUS
Parinti=zeros(dim,m);
r=unifrnd(0,1/dim);
i=1;j=1;
while (i<=dim)
   while(r<=q(j))
      Parinti(i,:)=Pop(j,:);
      i=i+1;
      r=r+1/dim;
   end;
   j=j+1;
end;
end
```

Implementare (3-1)

function [c1,c2] = PMX(x1,y1)

```
%implementarea operatorului PMX pentru doua permutari parinte (x1,y1)


n=length(x1);


gen=0;
while(~gen)
   poz=unidrnd(n,[1 2]);
   p1=min(poz);p2=max(poz);
   if(p1<p2)
      gen=1;
   end;
end;
c1=c_PMX(x1,y1,p1,p2);
c2=c_PMX(y1,x1,p1,p2);
end
```

```
function [c]=c_PMX(x1,y1,p1,p2)
n=length(x1);
%la momentul initial, toate genele cromozomului copil sunt "libere"
c=zeros(1,n);

%copiaza genele p1…p2 din primul parinte in primul copil
c(p1:p2)=x1(p1:p2);

%determina si copiaza in primul copil valorile inca necopiate din genele
%p1..p2 din al doilea parinte
A=[];
for i=p1:p2
   if(~ismember(y1(i),c))
      A=[A; [y1(i) i]];
   end;
end;
```

Implementare (3-2)

```
[p,~]=size(A);
for i=1:p
    %plaseaza alela A(i,1) din pozitia A(i,2)
    va=A(i,1);pa=A(i,2);
    b=c(pa);
    [~,j]=ismember(b,y1);
    while(c(j))
            b=c(j);
            [~,j]=ismember(b,y1);
    end;
    c(j)=va;
end;
```

```
 %copiaza in genele libere din permutarea
copil alelele inca necopiate din
%al doilea parinte, in ordinea in care apar
for i=n:-1:1
    if(~ismember(y1(i),c))
        %calculeaza ultima pozitie libera in c
        [~,j]=ismember(0,c);
        c(j)=y1(i);
    end;
end;
end
```

Implementare (4)

```
function [Copii] = crossover(Pop,pc,D)
[dim,m]=size(Pop);
Copii=Pop;
n=m-1;
% deoarece in populatia de parinti nu exista nici o ordine, pot fi imperechiati parinti succesivi
for t=1:dim/2
   r=unifrnd(0,1);
   % daca r<pc, are loc schimbul genetic
   % altfel, parintii sunt copiati in populatia de copii
   if(r<pc)
      [x,y]=PMX(Pop(2*t-1,1:n),Pop(2*t,1:n));
      Copii(2*t-1,1:n)=x;
      Copii(2*t-1,n+1)=f_obiectiv(x,D);
      Copii(2*t,1:n)=y;
      Copii(2*t,n+1)=f_obiectiv(y,D);
   end;
end;
end
```

Implementare (5)

```
function [y] = inversiune(x)
% mutatia prin inversiune
n=length(x);
gata=0;
while ~gata
   poz=unidrnd(n, [1 2]);
   p1=min(poz); p2=max(poz);
   if(p1<p2)
      gata=1;
   end;
end;
y=x;
y(p1:p2)=x(p2:-1:p1);
end
```

```
function [CopiiM] = mutatie(Pop,pm,D)
CopiiM=Pop;
[dim,m]=size(Pop);
n=m-1;
for t=1:dim
   r=unifrnd(0,1);
   if(r<pm)
      CopiiM(t,1:n)=inversiune(Pop(t,1:n));

CopiiM(t,n+1)=f_obiectiv(CopiiM(t,1:n),D);
   end;
end;
end
```

Implementare (6)

```
function [Pop_urm] =elitism(Pop_curenta,Copii )
%presupunem ca dimensiunea populatiei de copii=dimensiunea populatiei la
%momentul curent
[dim,m]=size(Pop_curenta);
Pop_urm=Copii;
[val1,i1]=max(Pop_curenta(:,m));
[val2,~]=max(Copii(:,m));
if(val1>val2)
    best=Pop_curenta(i1,:);
    %este inlocuit un copil aleator
    ind=unidrnd(dim);

    %disp(['Este inlocuit individul cu indice ' num2str(ind)]);
    Pop_urm(ind,:)=best;
end;
end
```

Implementare (9-1)

```
function [cost,solutie] =GA_TSP(dim,numefis,pc,pm,MAX,NRG)

% dim - numarul de indivizi din populatie
% numefis - numele fisierului in care sunt pastrate costurile tranzitiei de la un oras la altul
% pc - probabilitatea de crossover
% pm - probabilitatea de mutatie
% MAX - numarul maxim de iteratii in simularea evolutiei
% NRG - numarul maxim de generatii consecutive cu proprietatea  ca valoarea
% functiei obiectiv nu este modificata

% cost - costul celui mai bun drum calculat
% solutie - configuratia cu costul cel mai bun

%generarea populatiei initiale
[Pop,D]=gen_pop_permutari(dim,numefis);
[n,~]=size(D);
% n - numarul de orase
t=0;nrit=0;
```

Implementare (9-2)

```
% V - multimea celor mai bune valori (1/cost) calculate la fiecare generatie
[val,poz]=max(Pop(:,n+1));
V=[val];
% Sol - multimea celor mai buni candidati la solutie, pentru fiecare generatie
Sol=[Pop(poz,1:n)];
while t<MAX && nrit<NRG
   Parinti=SUS(Pop);
   Copii=crossover(Parinti,pc,D);
   CopiiM=mutatie(Copii,pm,D);
   t=t+1;
   PopU=elitism(Pop,CopiiM);
   [valU,pozU]=max(PopU(:,n+1));
   if(valU==val)
      nrit=nrit+1;
   else
      nrit=0;
   end;
   V=[V;valU];
   Sol=[Sol;PopU(pozU,1:n)];
   val=valU;
   Pop=PopU;
end;
```

```
%datorita elitismului, cel mai bun
individ calculat de algoritm este
%regasit in ultima generatie

cost=1/val;
solutie=Sol(t,:);
disp(['Costul cel mai bun:'
num2str(cost)]);
disp('Solutia cea mai buna');
disp(solutie);



figure
i=1:t;
plot(i,1./V(i));
axis([0,t+5,0,50]);

end
```

Implementare 10

- Fisiere cu distante

distante.txt

```
0 4 5 3 4 2 2 5 2 4
4 0 4 4 9 6 4 3 4 6
5 4 0 7 9 1 6 2 5 6
3 4 7 0 1 5 6 5 8 3
4 9 9 1 0 7 8 2 5 3
2 6 1 5 7 0 2 6 1 2
2 4 6 6 8 2 0 8 8 1
5 3 2 5 2 6 8 0 4 6
2 4 5 8 5 1 8 4 0 3
4 6 6 3 3 2 1 6 3 0
```

distante1.txt

```
0 4 5 1 4 2 5 5 2
4 0 4 3 1 6 4 8 4
5 4 0 7 9 9 1 3 5
1 3 7 0 5 5 6 7 8
4 1 9 5 0 7 8 2 5
2 6 9 5 7 0 2 6 1
5 4 1 6 8 2 0 8 8
5 8 3 7 2 6 8 0 4
2 4 5 8 5 1 8 4 0
```

distante2.txt

```
0 4 5 1 4 2 5 5 2 4 7 3
4 0 4 3 1 6 4 8 4 5 6 7
5 4 0 7 9 9 1 3 5 2 8 7
1 3 7 0 5 5 6 7 8 3 5 3
4 1 9 5 0 7 8 2 5 2 8 3
2 6 9 5 7 0 2 6 1 2 8 4
5 4 1 6 8 2 0 8 8 5 5 5
5 8 3 7 2 6 8 0 4 5 8 6
2 4 5 8 5 1 8 4 0 3 3 4
4 5 2 3 2 2 5 5 3 0 5 1
7 6 8 5 8 8 5 8 3 5 0 4
3 7 7 3 3 4 5 6 4 1 4 0
```

distante3.txt

```
0 4 5 1 4 2 5 5 2 4 7 3 4 2 3 5 1
4 0 4 3 1 6 4 8 4 5 6 7 3 8 8 2 3
5 4 0 7 9 9 1 3 5 2 8 7 3 2 3 2 3
1 3 7 0 5 5 6 7 8 3 5 3 4 3 4 5 6
4 1 9 5 0 7 8 2 5 2 8 3 2 3 4 3 3
2 6 9 5 7 0 2 6 1 2 8 4 3 5 3 5 5
5 4 1 6 8 2 0 8 8 5 5 5 3 2 3 3 2
5 8 3 7 2 6 8 0 4 5 8 6 4 5 4 6 7
2 4 5 8 5 1 8 4 0 3 3 4 2 1 4 5 4
4 5 2 3 2 2 5 5 3 0 5 1 3 2 2 3 3
7 6 8 5 8 8 5 8 3 5 0 3 4 3 1 4 3
3 7 7 3 3 4 5 6 4 1 3 0 4 3 4 7 8
4 3 3 4 2 3 3 4 2 3 4 4 0 4 3 2 3
2 8 2 3 3 5 2 5 1 2 3 3 4 0 3 4 2
3 8 3 4 4 3 3 4 4 2 1 4 3 3 0 1 7
5 2 2 5 3 5 3 6 5 3 4 7 2 4 1 0 2
1 3 3 6 3 5 2 7 4 3 3 8 3 2 7 2 0
```

Implementare – exemple de apel si rezultate (script)

- GA_TSP(100,'distante1.txt',0.6,0.1,700,70);

% Costul cel mai bun:16 - este chiar costul optim

% Solutia cea mai buna

% 9 1 4 2 5 8 3 7 6

- % GA_TSP(100,'distante2.txt',0.6,0.1,700,70);

% Costul cel mai bun:25 - este chiar costul optim

% Solutia cea mai buna

% 6 7 3 8 5 2 4 1 9 11 12 10

- % GA_TSP(500,'distante3.txt',0.8,0.1,1000,80);

% Costul cel mai bun:29- este chiar costul optim

% Solutia cea mai buna

% 8 5 2 4 1 17 14 9 6 10 12 11 15 16 13 7 3

% GA_TSP(500,'distante3.txt',0.8,0.15,1000,80);

% Costul cel mai bun:29

% Solutia cea mai buna

% 6 9 14 17 1 4 10 12 11 15 16 13 2 5 8 3 7

% GA_TSP(2000,'distante3.txt',0.8,0.15,1500,100);

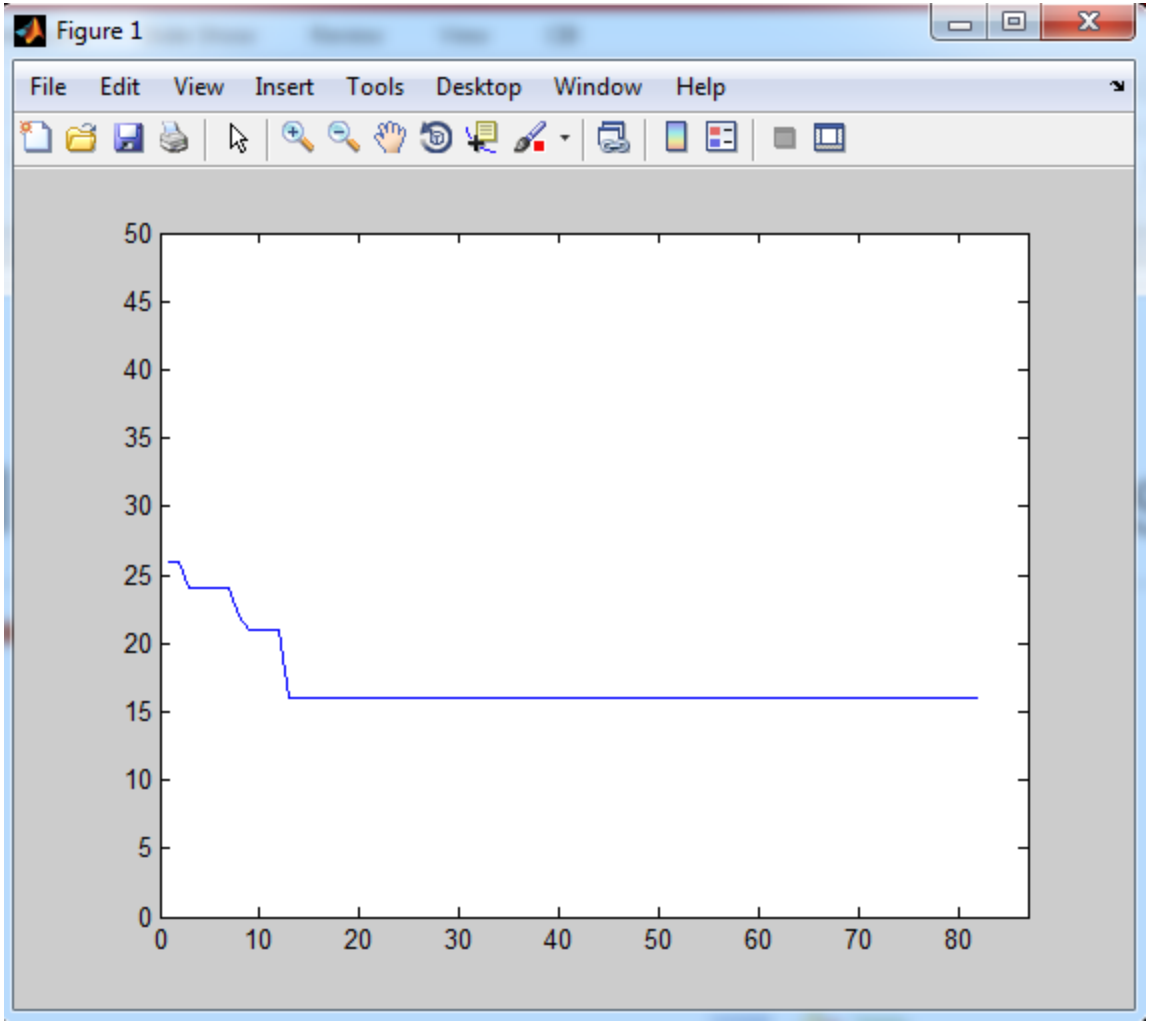
% Costul cel mai bun:29

% Solutia cea mai buna

% 1 17 13 16 15 11 12 10 14 9 6 7 3 8 5 2 4

Rezultate

Implementare – rezultat (distante1.txt)



Exercitiu

- Creati si alte fisiere pentru distante.
- Rulati modificand dimensiunea populatiei si variind parametrii pm si pc.