ALGORITMI GENETICI

Dr. George Daniel Mateescu[*]
Corina Sâman[*]
Mihai Buneci[*]

Noţiuni introductive

Un algoritm genetic reprezintă o transpunere in programare a unor principii biologice. Structura unui astfel de algoritm este descrisă prin următorii paşi:

Pasul 1. generează populaţia iniţială
Pasul 2. generează urmaşii (*off-sping*)
Pasul 3. aplică mutaţii genetice
Pasul 4. selectează o parte a populaţiei care părăseşte sistemul
Pasul 5. repetă pasii 2, 3, 4, până la satisfacerea unui criteriu de optimalitate (sau de un anumit număr de ori)

Formularea algoritmului este foarte generală, iar programatorul trebuie sa-şi adapteze propriul model, astfel:

Populaţia

Aceasta este, din punct de vedere matematic, mulţimea soluţiilor posibile ale problemei. De exemplu, pentru cazul în care se modelează o problemă de minim (sau de maxim) pentru o funcţie $f:\mathrm{R}^m \to \mathrm{R}$, atunci orice obiect $(x_1, x_2, ...x_m) \in \mathrm{R}^m$ reprezintă o soluţie posibilă. În consecinţă, în acest caz, orice submulţime $P \subseteq \mathrm{R}$ reprezintă, din punctul nostru de vedere, ***populaţia*** unui algoritm genetic.

Indivizii

Populaţia este formată din indivizi; din punct de vedere matematic, indivizii sunt elementele mulţimii Π. La rândul său, fiecare individ este caracterizat de trăsăturile sale, de zestrea sa cromozomială. Astfel, revenind la exemplul problemei de optimizare, este uşor de observat că un individ (adică un punct din R^m) este complet determinat de coordonatele sale, respectiv de valorile $x_1, x_2, ...x_m$. Desigur, fiecare coordonată va influenţa valoarea funcţiei care se minimizează (maximizează).

Urmaşii (generaţia nouă, *off-spring*)

Această noţiune modelează un procedeu de modificare (îmbogăţire) a mulţimii Π. Putem înţelege mai bine această situaţie daca o comparăm cu alte metode de calcul. De exemplu, anumite metode numerice determină soluţia prin intermediul unui şir de valori (definite, eventual, iterativ) ceea ce reprezintă, de asemenea, o situaţie de îmbogăţire a unei mulţimi, prin construcţia, succesivă, de noi valori. De asemenea, metodele de simulare cunosc modalităţi de creare a de noi entităţi, adecvate unei anumite probleme.

Modul de generare a urmaşilor depinde, intrinsec, de problema modelată, fără să existe un procedeu general. Astfel, pentru problema de optimizare a funcţiei *f* descrise mai sus, putem

[*] Institutul de Prognoză Economică, Academia Română, e-mail: ipe@ipe.ro

considera ca un urmaş provine din alăturarea componentelor (trăsăturilor cromozomiale) preluate de la doi "părinți". Matematic, dacă vom considera doi indivizi:

$$(x_1, x_2, ..., x_k, x_{k+1}, ..., x_m)$$

$$(y_1, y_2, ..., y_k, y_{k+1}, ..., y_m)$$

atunci, printr-un procedeu inspirat din mecanismul *cross-over*, putem imagina doi urmaşi, de forma:

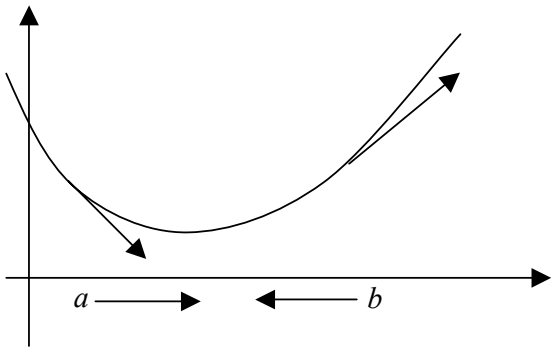
$$(x_1, x_2, ..., x_k, y_{k+1}, ..., y_m)$$

$$(y_1, y_2, ..., y_k, x_{k+1}, ..., x_m)$$

obținuți prin "*divizarea lanțului cromozomial*" în poziția *k*.

Mutații genetice

Orice modificare "*suficient de mică*" a unui "*cromozom*" reprezintă o mutație genetică. Astfel de mutații pot fi haotice (întâmplătoare) sau pot fi urmarea unor aşa-numite achiziții genetice. Desigur, modelul cel mai adecvat de mutație genetică va putea fi ales numai în funcție de problema modelată. În cazul particular al unei probleme de optimizare, cea mai potrivită cale de reprezentare a mutațiilor genetice este constituită de aşa numita deplasare pe direcția gradientului. Justificarea unui astfel de procedeu se regăseşte în metodele numerice cunoscute, de tip *Newton*, în care determinarea soluției se face printr-o deplasare pe o ***direcție*** care ține cont de gradientul funcției.



Imaginea de mai sus este sugestivă, astfel, dacă vrem să determinăm minimul funcției şi ne aflăm în punctul *a*, atunci va trebui să „*ne deplasăm*" spre dreapta, ceea ce corespunde unui sens crescător, invers față de semnul derivatei (gradientului) funcției în punctul *a*. Dacă ne aflăm în punctul *b*, atunci va trebui să realizăm o deplasare spre stânga, ceea ce reprezintă un sens descrescător, invers semnului derivatei (gradientului) funcției în punctul *b*.

Strategia de deplasare care ține seama de direcția gradientului poate fi extinsă pentru cazul funcțiilor de mai multe variabile, prin deplasări în raport cu fiecare coordonată, corespunzător semnului (direcției) derivatei parțiale a funcției obiectiv, în raport cu respectiva coordonată.

Mutațiile genetice sunt deosebit de importante pentru convergența algoritmului, pentru probleme concrete, aşa cum am demonstrat în [2].

Selecția

Acest pas reprezintă modul în care anumiți indivizi ies din sistem. Alegerea unui procedeu de ieşire din sistem ține de problema modelată. Astfel, este cu totul natural ca pentru o problemă de optimizare, clasificarea populației să se facă în funcție de valoarea corespunzătoare

a funcției obiectiv. Această funcție devine, necesar, o funcție de performanță, prin intermediul căreia se va decide ieşirea din sistem a indivizilor.

Concret, pentru o problemă de minimizare a unei funcții *f*, vom clasifica indivizii ***crescător***, în funcție de valoarea corespunzătoare a funcției *f* şi vom elimina surplusul de populație, rezultat ca urmare a generării de „*urmaşi*". Pentru o problemă de maximizare vom clasifica indivizii în ordinea descrescătoare a valorilor funcției obiectiv.

Finalitatea

Evident, ca în orice algoritm, trebuie să existe un test de terminare a acestuia. Finalitatea este determinată de scopul urmărit, deci nu se poate indica o strategie generală. Pentru probleme de optimizare, aşa cum am arătat în lucrarea [2] se poate folosi un test similar cu estimarea *aposteriori* a erorii, cunoscută în probleme de calcul numeric. Astfel, dacă notăm prin x_n individul cu cea mai bună performanță al generației a *n*-a, atunci un test de oprire a algoritmului poate fi de forma:

$$\left| f\left(x_{n-1}\right) - f\left(x_n\right) \right| < \varepsilon$$

Programarea unui algoritm de optimizare, prin algoritm genetic

Considerăm următoarea problema de optimizare:

$$\min f\left(x,z,y\right)$$
$$f\left(x,z,y\right) = \left(x-1\right)^2 + \left(y-2\right)^2 + \left(z-4\right)^2$$

Pentru a determina soluția printr-un algoritm de tip genetic, vom modela conceptele expuse privind populația, urmaşii, mutațiile genetice şi selecția.

Am ales acest exemplu pentru că cititorul poate înțelege şi poate verifica programele, comparând soluția obținută prin algoritm, cu soluția exactă, în cazul dat $x = 1, y = 2, z = 4$.

Având în vedere că un algoritm genetic este un procedeu de determinare aproximativă, utilizatorul va accepta, ca soluție, orice aproximare suficient de fină a soluției exacte.

Populația. Orice submulțime din R^3 poate juca rolul de populație, orice punct de coordonate $\left(x,y,z\right)$ putând fi o soluție posibilă. Inițial, vom genera populația în mod aleator. Pe parcursul algoritmului, vom considera că populația are un număr fix de indivizi (1000 în exemplul dat) şi vom menține acest număr, prin eliminarea excesului de populație, în baza criteriului de optimalitate.

Drept suport de programare am ales limbajul ***Pascal***, astfel încât definiția populației revine la o lista de forma:

```
var
pop:array[1..1200,1..4] of real;
```

Fiecare linie a matricei **pop** reprezintă un individ. Pentru fiecare individ, primele trei componente reprezintă coordonatele *x*, *y*, *z*, în accepțiunea "genetică" a algoritmului aceasta fiind zestrea sa cromozomială. Cea de-a patra componentă din fiecare line a matricei, corespunzătoare unui individ, va conține valoarea funcției de performanță, respectiv $f\left(x,y,z\right)$.

Schematic, matricea populației se prezintă în forma:

x_1	y_1	z_1	$f(x_1, y_1, z_1)$
x_2	y_2	z_2	$f(x_2, y_2, z_2)$
…	…	…	…
x_n	y_n	z_n	$f(x_n, y_n, z_n)$

Funcția de performanță. Având în vedere specificul problemei, funcția de performanță este chiar $f(x,z,y)=(x-1)^2+(y-2)^2+(z-4)^2$, iar programarea acesteia se face ținând cont de sintaxa specifică a limbajului ***Pascal***:

```
function f(x,y,z:real):real;
begin
   f:=sqr(x-1)+sqr(y-2)+sqr(z-4);
end;
```

Selecția. Aceasta este necesară pentru a păstra numărul de indivizi la nivelul inițial de 1000. La fiecare generație vom adăuga 200 de urmași, după care vom elimina surplusul având drept criteriu valoarea funcției de performanță. Deoarece această valoare este plasată în a patra poziție a fiecărei linii, urmează să aplicăm un procedeu de ordonare descrescătoare a liniilor matricei **pop**, în funcție de valoarea aflată pe ultima coloană.

Algoritmul folosit este o transpunerea cunoscutei metode a bulelor, in care, prin parcurgeri succesive ale liniilor matricei, elementul de valoare minimă avansează în jos. Sortarea este implementată în procedura care urmează.

```
procedure ordonare;
var
   i,j,h:integer;
   test:boolean;
   x:real;
begin
for h:=1199 downto 1 do
   for i:=1 to h do
    if pop[i,4]>pop[i+1,4] then
      for j:=1 to 4 do
         begin
            x:=pop[i,j]; pop[i,j]:=pop[i+1,j];pop[i+1,j]:=x;
         end;
end;
```

Inițializarea. Populația inițială este generată aleator, folosind generatorul de numere aleatoare al limbajului ***Pascal***. Pentru fiecare individ, după generarea aleatoare a componentelor sale (coordonate) se calculează valoarea funcției de performanță, care este plasată în ultima poziție.

```
procedure initializare;
var i,j:integer;
begin
  randomize;
  for i:=1 to 1200 do for j:=1 to 3 do pop[i,j]:=10*random;
  for i:=1 to 1200 do pop[i,4]:=f(pop[i,1],pop[i,2],pop[i,3]);
end;
```

Mutaţiile genetice. După cum am menţionat, mutaţiile genetice joacă un rol cheie în algoritm, deoarece, aşa cum am văzut în [2] de îndată ce iniţiem mutaţii genetice pe direcţia gradientului funcţiei obiectiv, algoritmul este convergent. La rândul său, gradientul poate fi calculat folosind o formulă de calcul aproximativ, pentru fiecare derivată parţială:

$$\frac{\partial f}{\partial x} \approx \frac{f(x+h,y,z)-f(x,y,z)}{h}$$
$$\frac{\partial f}{\partial y} \approx \frac{f(x,y+h,z)-f(x,y,z)}{h}$$
$$\frac{\partial f}{\partial z} \approx \frac{f(x,y,z+h)-f(x,y,z)}{h}$$

După cum se ştie, precizia aproximării în formulele de mai sus este de acelaşi ordin de mărime cu *h*. Pe de altă parte, mutaţiile genetice trebuie să fie variaţii mici ale coordonatelor, astfel încât am folosit expresiile: $-0.001\cdot\operatorname{sgn}\frac{\partial f}{\partial x}$, $-0.001\cdot\operatorname{sgn}\frac{\partial f}{\partial y}$ respectiv $-0.001\cdot\operatorname{sgn}\frac{\partial f}{\partial z}$ pentru coordonatele corespunzătoare.

```
procedure mutatie;
var i,n,m:integer;
begin
  for i:=1 to 20 do begin
    n:=1+random(1000);
    if
      f(pop[n,1]+0.001,pop[n,2],pop[n,3])>f(pop[n,1],pop[n,2],pop[n,3])
      then pop[n,1]:=pop[n,1]-0.001 else pop[n,1]:=pop[n,1]+0.001;
    if
      f(pop[n,1],pop[n,2]+0.001,pop[n,3])>f(pop[n,1],pop[n,2],pop[n,3])
      then pop[n,2]:=pop[n,2]-0.001 else pop[n,2]:=pop[n,2]+0.001;
    if
      f(pop[n,1],pop[n,2],pop[n,3]+0.001)>f(pop[n,1],pop[n,2],pop[n,3])
      then pop[n,3]:=pop[n,3]-0.001 else pop[n,3]:=pop[n,3]+0.001;
    pop[n,4]:=f(pop[n,1],pop[n,2],pop[n,3]);
  end;
end;
```

Urmaşii (*off spring*). Pentru doi indivizi, *x*, *y*, identificaţi prin liniile *m* şi *n* ale matricei populaţiei, am generat doi „urmaşi" folosind mecanismul *cross-over*, astfel:

$z_1 = x_1, z_2 = y_2, z_3 = y_3$, respectiv $z_1 = x_1, z_2 = x_2, z_3 = y_3$

Indivizii obţinuţi sunt plasaţi pe liniile de la 1001 la 1200 ale matricei astfel încât, după sortare în funcţie de performanţă, numărul de indivizi rămâne constant, corespunzător liniilor 1..1000 ale matricei **pop**.

```
procedure generatie;
var i,j,k,m,n:integer;
begin
for i:=1001 to 1100 do
 begin
  m:=1+random(20);n:=1+random(20);
  pop[i,1]:=pop[m,1];
  pop[i,2]:=pop[n,2];
  pop[i,3]:=pop[n,3];
  pop[i,4]:=f(pop[i,1],pop[i,2],pop[i,3]);
  pop[i+100,1]:=pop[n,1];
```

```
  pop[i+100,2]:=pop[n,2];
  pop[i+100,3]:=pop[m,3];
  pop[i+100,4]:=f(pop[i+100,1],pop[i+100,2],pop[i+100,3]);
 end;
end;
```

Programul propriu-zis, care rezolvă problema, constă în execuția repetată a procedurilor corespunzătoare algoritmului genetic:

```
var i:integer;
begin
  initializare;
  for i:=1 to 300 do begin
    generatie;
    mutatie;
    ordonare;
  end;
writeln(pop[1,1],'  ',pop[1,2],'  ',pop[1,3],'  ',pop[1,4]);
readln;
end.
```

Calculul parametrilor funcției de regresie

Vom prezenta în continuare o aplicație a programării genetice pentru determinarea parametrilor unei funcții de regresie. Problema model utilizează următorul cadru:

$x_i, i = 1..n$ serie de valori (date) independente

$y_i, i = 1..n$ serie de valori (date) dependente

$y = ax + b$ relația de dependență prezumată

Problema constă în determinarea parametrilor *a* şi *b*, care minimizează suma pătratelor erorilor (diferențelor) dintre valorile statistice (observate) y_i şi cele rezultate din relația de dependență prezumată, respectiv, $ax_i + b$.

Considerăm funcția: $\varphi(a,b) = \sum_{i=1..n} (y_i - ax_i - b)^2$ iar problema devine:

$$\min \varphi(a,b)$$

adică o problemă de optimizare pentru care se poate aplica algoritmul (şi programul) anterior.

Desigur, pentru regresii de tip liniar, ca cea de mai sus, există formule de determinare a parametrilor *a* şi *b*. Pe de altă parte, se observă că prin aplicarea unui algoritm de tip genetic, putem determina parametrii unei funcții arbitrare, indiferent dacă există sau nu există formule de determinare.

Ne propunem să utilizăm o dependență de tipul:

$$y = a + bx + cx^2$$

şi observăm cu uşurință că determinarea parametrilor *a*, *b*, *c* se face prin minimizarea funcției:

$$f(a,b,c) = \sum_{i=1..n} (y_i - a - bx_i - cx_i^2)^2$$

Adaptarea algoritmului genetic deja prezentat se reflectă în elementele următoare.

În plus față de variabilele globale ale programului anterior, am introdus variabilele **vx** şi **vy**, care corespund, respectiv, variabilei independente *x* şi variabilei dependente *y*. Aceste variabile vor fi citite din fişierul **c:\date.dat**, identificat prin descriptorul **date**.

```
var
pop:array[1..1200,1..4] of real;
date : text;
nr : integer;
vx,vy:array[1..100] of real;
```

Adaptarea funcției de performanță se face în conformitate cu problema de minimizare a sumei abaterilor pătratice, respectiv metoda celor mai mici pătrate.

```
function f(x,y,z:real):real;
var
s:real;
i:integer;
begin
   s:=0; for i:=1 to nr do
         s:=s+sqr(vy[i]-x-y*vx[i]-z*sqr(vx[i]));
   f:=s;
end;
```

Inițializarea datelor problemei şi a populației conține, în plus față de programul anterior, procedura de inițializare (citire) a valorilor seriilor de date.

```
procedure initializare;
var i,j:integer;
begin
  assign(date,'c:\date.dat');
  reset(date);
  readln(date,nr);
  for i:=1 to nr do readln(date,vx[i],vy[i]);
  close(date);
  randomize;
  for i:=1 to 1200 do for j:=1 to 3 do pop[i,j]:=random;
  for i:=1 to 1200 do pop[i,4]:=f(pop[i,1],pop[i,2],pop[i,3]);
end;
```

Sisteme de ecuații neliniare

Tehnica de programare genetică poate fi aplicată pentru determinarea soluțiilor sistemelor de ecuații neliniare. Pentru un astfel de sistem, scris formal:

$$\begin{cases} F_1(x_1, x_2, ..., x_m) = 0 \\ F_2(x_1, x_2, ..., x_m) = 0 \\ \quad ... \\ F_m(x_1, x_2, ..., x_m) = 0 \end{cases}$$

vom considera $f(x_1, x_2, ..., x_m) = \sum_{i=1..m} \left(F(x_1, x_2, ..., x_m)\right)^2$, iar sistemul de ecuații dat este echivalent cu problema de optimizare:

$$\min f(x_1, x_2, ..., x_m)$$

De exemplu, pentru determinarea soluției sistemului de ecuații neliniare:

$$\begin{cases} -x+y^2-z=-1 \\ x+2yz-z=2 \\ xy-y-9z=-9 \end{cases}$$

vom avem $f(x,y,z)=\left(-x+y^2-z+1\right)^2+\left(x+2yz-z-2\right)^2+\left(xy-y-9z+9\right)^2$.

Singura deosebire față de primul program constă în descrierea funcției de performanță, în limbajul ***Pascal***:

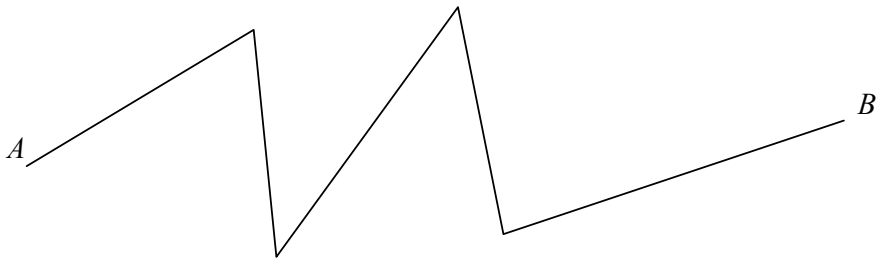
```
function f(x,y,z:real):real;
begin
   f:=sqr(-x+sqr(y)-z+1)+sqr(x+2*y*z-z-2)+sqr(x*y-y-9*z+9);
end;
```

O problemă de optimizare discretă: problema comis-voiajorului

Sub acest nume este cunoscută o problemă, semnificativă, prin care ne propunem să determinăm un drum de cost minim într-un graf. Formularea acesteia se face prin referire la următoarea situație practică:

> Se presupune că un comis-voiajor trebuie să viziteze un număr de *n* oraşe, pornind din oraşul *A* şi având ca destinație finală oraşul *B*. Deplasarea între oricare două oraşe este condiționată de un anumit cost. Se cere să se determine traseul optim, adică acel traseu care parcurge toate oraşele, costul total al deplasării fiind minim.

Pentru început, vom observa că numărul total de trasee este egal cu *n*! ceea ce face ca determinarea traseului optim, printr-o metodă exhaustivă, să fie imposibilă, chiar pentru valori modeste ale lui *n*.



Problema poate fi modelată în conformitate cu principiile genetice, deja expuse.

Vom începe cu noțiunea de individ. Aşa cum am observat, un individ reprezintă o soluție posibilă a problemei date. În consecință, vom considera că un traseu complet, de la *A* la *B*, care trece prin toate oraşele, reprezintă un individ. Aşadar, codificarea unui individ se prezintă sub forma:

$$A\ O_{i1}\ O_{i2}\ \ldots\ O_{in}\ B$$

Costul deplasării între două oraşe va fi reținut în matricea de adiacență **a**, în care elementul **a[i,j]** conține o valoare numerică ce reprezintă costul deplasării de la oraşul „**i**" la oraşul „**j**". Nu este neapărat necesar ca **a[i,j]=a[j,i]**.

În exemplul considerat (în program) am folosit un model care are 14 oraşe, primul şi ultimul fiind fixate (*A* şi *B*). Populația este reprezentată prin matricea **pop**, fiecare linie reprezentând un individ (traseu) iar pe ultima poziție a fiecărei linii se află, calculat, costul traseului respectiv.

```
var
pop:array[1..1200,1..15] of integer;
a:array[1..14,1..14] of integer;
date:text;
```

Funcția de performanță calculează costul total al deplasării, corepunzătoare unui individ (traseu posibil).

```
function f(i:integer):integer;
var s,j:integer;
begin
   s:=0;
   for
   j:=2 to 14 do
        s:=s+a[pop[i,j-1],pop[i,j]];
   f:=s;
end;
```

Clasificarea indivizilor se face în funcție de valorile crescătoare ale funcției de performanță (costul traseului).

```
procedure ordonare;
var
   i,j,h:integer;
   test:boolean;
   x:integer;
begin
for h:=1199 downto 1 do
   for i:=1 to h do
    if pop[i,15]>pop[i+1,15] then
      for j:=1 to 15 do
         begin
            x:=pop[i,j]; pop[i,j]:=pop[i+1,j];pop[i+1,j]:=x;
         end;
end;
```

Datele privind costurile deplasărilor sunt citite din fişier şi depuse în matricea **a** (matricea de adiacență a grafului).

Populația inițială este generată aleator, fiecare individ fiind "depus" pe o linie a matricii **pop**. Pe ultima poziție din fiecare linie se plasează costul traseului, calculat cu ajutorul funcției *f*.

```
procedure initializare;
var i,j,k,x:integer;
test:boolean;
begin
  assign(date,'c:\work\a.txt');
  reset(date);
  for i:=1 to 14 do begin
```

```
     for j:=1 to 14 do read(date,a[i,j]);
     readln(date);
  end;
  close(date);
  randomize;
  for i:=1 to 1000 do begin
    for j:=2 to 13 do begin
      repeat
        x:=2+random(12);
        test:=true; for k:=2 to j-1 do if x=pop[i,k] then test:=false;
      until test;
      pop[i,j]:=x;
    end;
    pop[i,1]:=1;pop[i,14]:=14;
  end;
  for i:=1 to 1000 do
        pop[i,15]:=f(i);
end;
```

Mutațiile genetice au fost modelate prin alegerea, în mod aleator, a unui traseu şi inversarea pozițiilor unor oraşe, selectate de asemenea aleator.

```
procedure mutatie;
var i,n,m,p,x:integer;
begin
  for i:=1 to 50 do begin
    n:=1+random(1200);
    m:=2+random(12);p:=2+random(12);
    x:=pop[n,m];pop[n,m]:=pop[n,p];pop[n,p]:=x;
    m:=2+random(12);p:=2+random(12);
    x:=pop[n,m];pop[n,m]:=pop[n,p];pop[n,p]:=x;
    pop[n,15]:=f(n);
  end;
end;
```

Generația nouă (*off spring*) este obținută printr-un procedeu *cross-over*. Spre deosebire de problemele (modelele) anterioare, pentru problema comis-voiajorului, nu orice doi indivizi sunt eligibili. Concret, fiind date două trasee care se intersectează, vom crea două noi trasee, astfel:

- primul traseu continuă, de la intersecție, pe cel de-al doilea traseu
- al doilea traseu continuă, de la intersecție, pe primul traseu

Cele două noi trasee sunt permise dacă se parcurg toate oraşele; procedura care urmează verifică această condiție.

```
procedure generatie;
var i,j,k,m,n:integer;
test:boolean;
begin
for i:=1001 to 1100 do
  begin
    repeat test:=true;
       m:=1+random(20);n:=1+random(20);
       k:=1;
       repeat
         k:=k+1;
       until pop[m,k]=pop[n,k];
       for j:=1 to k do pop[i,j]:=pop[m,j];
```

```
        for j:=k+1 to 14 do pop[i,j]:=pop[n,j];
        for j:=2 to 13 do
          for k:=2 to 13 do
                  if (pop[i,j]=pop[i,k]) and (j<>k) then test:=false;
        for j:=1 to k do pop[i+100,j]:=pop[n,j];
        for j:=k+1 to 14 do pop[i+100,j]:=pop[m,j];
        for j:=2 to 13 do
          for k:=2 to 13 do
             if (pop[i+100,j]=pop[i+100,k]) and (j<>k) then test:=false;
      until test;
      pop[i,15]:=f(i);
      pop[i+100,15]:=f(i+100);
    end;
  end;
```

Programul propriu-zis conține etapele care caracterizează orice algoritm genetic:

```
  var i,j:integer;
  begin
    initializare;
    for i:=1 to 100 do begin
      generatie;
      mutatie;
      ordonare;
    end;
  for j:=1 to 14 do write(pop[1,j],'  ');
  writeln('cost:',pop[1,15]);
  readln;
  end.
```

Bibliografie

[1]Banzhaf W., Nordon P., Keller R.E., Francone F.D., *Genetic Programming – An introduction*, Morgan Kaufmann Publishiers, San Francisco 1998

[2]Mateescu, G. D.: *Optimization by Using Evolutionary Algorithms with Genetic Acquisitions*, Romanian Journal of Economic Forecasting, 2, 2005

[3]Stoer J., Bulirsch R., *Introduction to Numerical Analysis*, Springer-Verlag, 1992