1. **Algoritmi genetici**
   1. **Introducere. Prezentare generală**

Algoritmii genetici (GA) au fost dezvoltaţi iniţial de Holland (Holland, 1975) ca un instrument de studiu al comportamentului de tip adaptiv. Chiar dacă nu întotdeauna corect (De Jong), GA sunt în general consideraţi ca aparţinând clasei metodelor de optimizare.

Un GA „clasic” (sau canonic) este caracterizat astfel: *reprezentarea* populaţiei este realizată prin intermediul *şirurilor binare*, operatorul de *selecţie* a unui individ în multisetul părinţilor este *proporţional cu valoarea funcţiei de evaluare* calculată pentru acel individ, *probabilitatea efectuării unei mutaţii este scăzută*, accentul fiind pus pe procesul de *recombinare*, de inspiraţie biologică, *ca mijloc de generare a noi candidaţi la soluţie*.

Un exemplu simplu relativ la evoluţia unui GA este prezentat în (Goldberg, 1989) şi se referă la maximizarea funcţiei , cu . Un fenotip (număr din cuprins între 0 şi 31) este reprezentat printr-un şir binar de lungime 5 (pe 5 biţi pot fi reprezentate numerele din cuprins între 0 şi ). Dimensiunea populaţiei, *dim*, este 4. Populaţia iniţială este generată aleator, din distribuţia uniformă pe . Prin apelul funcţiei MATLAB *genereaza\_ini(dim)* este generată *pop*, populaţia iniţială. Reprezentarea cromozomială a fiecărui fenotip *x* pe *m* biţi (aici ) este *y*, obţinută prin apelul funcţiei *repr\_bin(x,m)*. Pentru un cromozom dat, *y*, reprezentat pe *m* biţi, fenotipul corespunzător lui este calculat prin funcţia MATLAB *repr\_int(y,m)*.

function [pop]=genereaza\_ini(dim);

pop=zeros(dim,6);

%fiecare membru al populatiei este un numar natural, cuprins inte 0 si

%31,reprezentat printr-un sir binar de dimensiune 5

%la care este adaugata valoarea functiei obiectiv

for i=1:dim

%genereaza dim numere intre 0 si 31, distribuite uniform - functia

%unidrnd cu primul parametru N genereaza uniform numere de la 1 la N

%genereaza numere de la 1 la

x=unidrnd(32)-1;

pop(i,1:5)=repr\_bin(x,5);

%evaluarea fiecarui membru al populatiei prin intermediul functiei

%obiectiv

pop(i,6)=f\_obiectiv(x);

end;

disp(pop);

end

function [y]=repr\_bin(x,m);

y=bitget(x,m:-1:1);

end

function [x]=repr\_int(y,m);

x=0;

% fiecare bit al lui x, initializat cu 0, este setat conform

% reprezentarii y

for i=1:m

x=bitset(x,m-i+1,y(i));

end;

end

function [val]=f\_obiectiv(x);

val=x^2;

end

**Observaţii**

1. Trecerea unui şir de cifre dintr-o bază dată în baza 10 poate fi realizată prin apelul funcţiei MATLAB

numar=base2dec(sir\_cifre,baza\_data)

Exemple:

numar = base2dec('0011110',2), rezultă numar =30

numar = base2dec('0002113',4), rezultă numar = 151

1. Trecerea unui număr scris în baza 10 într-o altă bază, cu rezultat un şir de cifre în acea bază, poate fi realizată prin apelul funcţiei

sir\_baza = dec2base(numar,baza)

Exemple:

sir\_baza = dec2base(20,2), rezultă sir\_baza =10100

sir\_baza = dec2base(151,4), rezultă sir\_baza = 2113

Probabilitatea de selecţie a părinţilor este calculată astfel:

1. Pentru fiecare cromozom , corespunzător fenotipului , se calculează performanţa sa prin funcţia de evaluare ;
2. Se calculează performanţa întregii populaţii ca sumă a performanțelor membrilor:
3. Pentru fiecare cromozom , se calculează probabilitatea de selecţie  
   şi probabilitatea cumulată

**Observaţie**. Evident, o parte a cromozomilor populaţiei curente poate fi selectată de mai multe ori. Acest lucru corespunde intuitiv ideii conform căreia cei mai buni cromozomi generează mai mulţi copii, iar cei mai slabi dispar.

Pentru o populaţie dată, *pop*, prin apelul funcţiei MATLAB *selectie\_parinti(pop)* este realizat calculul probabilităţii de selecţie, al probabilităţii cumulate, al numărului de apariţii a fiecărui individ în bazinul de împerechere şi este implementată o modalitate de alegere a acestuia, pe baza unei proceduri de tip ruletă (prezentată mai amplu în §3.6).

Bazinul de împerechere (multisetul părinţilor) este creat prin selecţii aleatoare pe baza distribuţiei de probabilitate cumulată astfel. Procesul de selecţie constă în rotirea ruletei de *dim* ori. De fiecare dată este selectat un câte un singur cromozom astfel:

1. Se generează aleator un număr ;
2. Dacă este selectat ; altfel, este selectat cromozomul , cu proprietatea

function [parinti,p,q]=selectie\_parinti(pop);

[dim,m]=size(pop);

% aici m este 6, adica numarul de biti+1(valoarea fitness)

p=zeros(dim,1);

p(1:dim)=pop(1:dim,6);

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=zeros(dim,m);

for k=1:dim

r=unifrnd(0,1);

pozitie=1;

for i=1:dim

if(r<=q(i))

pozitie=i;

break;

end;

end;

parinti(k,1:m)=pop(pozitie,1:m);

end;

end

Indivizii selectaţi în populaţia de părinţi interschimbă material genetic prin aplicarea operatorului de încrucişare. Părinţii sunt împerecheaţi aleator, din fiecare pereche rezultând doi copii astfel: fie , perechea de puncte selectate; se generează *pozitie*, un număr aleator în intervalul şi sunt efectuate operaţiile (*m*=5 dimensiunea reprezentării binare):

* copiază primele *pozitie* elemente din în , respectiv din în
* copiază în ultimele *m- pozitie +1* poziţii din , respectiv , ultimele *m-pozitie +1* elementele din , respectiv .

Funcţia MATLAB *crossover(pop)* furnizează copiii populaţiei *pop*, *popN*, obţinuţi pe baza mecanismului explicat mai sus.

function [popN]=crossover(pop);

[dim,n]=size(pop);

popN=zeros(dim,n);

m=n-1;V=[];

for k=1:dim/2

% determinarea a doi indivizi distincti si care nu au fost selectati

% anterior

poz=zeros(2,1);

for i=1:2

poz(i)=unidrnd(dim);

while(ismember(poz(i),V))

poz(i)=unidrnd(dim);

end;

V=[V poz(i)];

end;

% incrucisarea celor doi indivizi; rezulta y1,y2

disp('Incrucisare intre indivizii din pozitiile');

disp(poz);

x1=pop(poz(1),1:n);y1=x1;

x2=pop(poz(2),1:n);y2=x2;

% genereaza pozitia din care se face interschimbul

pozitie=unidrnd(4);

disp('Punctul de incrucisare');

disp(pozitie);

y1(pozitie+1:m)=x2(pozitie+1:m);

y2(pozitie+1:m)=x1(pozitie+1:m);

y1(m+1)=f\_obiectiv(repr\_int(y1(1:m),m));

y2(m+1)=f\_obiectiv(repr\_int(y2(1:m),m));

popN(2\*(k-1)+1,1:n)=y1(1:n);

popN(2\*k,1:n)=y2(1:n);

end;

end

Multisetul urmaşilor direcţi rezultaţi este supus operaţiei de mutaţie, efectuate cu probabilitatea *pm*. În cadrul unui GA clasic, operatorul mutaţie este aplicat astfel: pentru fiecare genă din fiecare cromozom este generat *r*, un număr aleator în intervalul . Dacă , atunci gena respectivă suferă o mutaţie (valoarea este schimbată din 0 în 1, respectiv din 1 în 0). În general probabilitatea efectuării unei mutaţii este foarte mică (de exemplu 0.001). În exemplul nostru, deoarece numărul de indivizi este foarte mic, am considerat . Apelul funcţiei MATLAB *mutatie(pop,pm)* determină obţinerea unei noi populaţii, *popN*, în urma efectuării mutaţiei asupra populaţiei curente *pop*.

function [popN]=mutatie(pop,pm);

[dim,n]=size(pop);

popN=pop;

m=n-1;

for i=1:dim

efectuat=0;

for k=1:m

r=unifrnd(0,1);

if(r<pm)

disp('Mutatie efectuata in cromozomul'); disp(i);

disp('Gena');disp(k);

popN(i,k)=not(popN(i,k));

efectuat=1;

end;

if(efectuat)

popN(i,n)=f\_obiectiv(repr\_int(popN(i,1:m),m));

end;

end;

end;

end

Într-un GA clasic, selecţia supravieţuitorilor este de tip generaţie, în sensul că, la fiecare moment de timp, noua populaţie rezultă prin înlocuirea populaţiei curente cu urmaşii direcţi şi care au suferit eventuale mutaţii. În următoarea sursă MATLAB este descrisă generarea populaţiei iniţiale şi obţinerea următoarei generaţii, conform mecanismelor descrise mai sus.

function []=o\_iteratie();

dim=4;

[pop]=genereaza\_ini(dim);

disp('Populatia initiala si valorile functiei obiectiv');

disp(pop);

[parinti,p,q,na]=selectie\_parinti(pop);

disp('Probabilitatile de selectie');disp(p);

disp('Probabilitatile cumulate'); disp(q);

disp('Bazinul de imperechere si valorile functiei obiectiv');

disp(parinti);

disp('Operatia de incrucisare');

popN=crossover(parinti);

disp('Copii rezultati');

disp(popN);

disp('Operatia de mutatie');pm=0.01;

[popN]=mutatie(popN,pm);

disp('Populatia rezultata');

disp(popN);

end

La o execuţie a acestei funcţii pot fi obţinute următoarele date.

Populaţia iniţială

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Indice cromozom | Cromozom | Valoarea funcţiei obiectiv | Probabilitatea de selecţie | Probabilitatea cumulată |
| 1 | 0 1 0 0 0 | 64 | 0.0566 | 0.0566 |
| 2 | 1 1 0 0 1 | 625 | 0.5531 | 0.6097 |
| 3 | 1 0 1 0 1 | 441 | 0.3903 | 1.0000 |
| 4 | 0 0 0 0 0 | 0 | 0 | 1.0000 |

Populaţia selectată pentru recombinare şi populaţia rezultată

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bazinul de împerechere selectat şi modul de împerechere | | | | Indivizii rezultaţi (copiii) | |
| Indice cromozom | Cromozom | Valoarea funcţiei obiectiv | Împerecherea | Cromozom | Valoarea funcţiei obiectiv |
| 1 | 1 1 0 0 1 | 625 | Cromozomii 1 şi 2, punctul de încrucişare 4 | 1 1 0 0 1 | 625 |
| 2 | 1 1 0 0 1 | 625 | 1 1 0 0 1 | 625 |
| 3 | 1 0 1 0 1 | 441 | Cromozomii 3 şi 4, punctul de încrucişare 3 | 1 1 0 0 1 | 625 |
| 4 | 1 1 0 0 1 | 625 | 1 0 1 0 1 | 441 |

Efectuarea operaţiei de mutaţie

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Populaţia urmaşilor direcţi şi modul de efectuare a mutaţiei | | | | Indivizii rezultaţi (generaţia următoare) | |
| Indice cromozom | Cromozom | Valoarea funcţiei obiectiv | Împerecherea | Cromozom | Valoarea funcţiei obiectiv |
| 1 | 1 1 0 0 1 | 625 | Mutaţii efectuate în primul cromozom, genele 3 şi 5 | 1 1 1 0 0 | 784 |
| 2 | 1 1 0 0 1 | 625 | 1 1 0 0 1 | 625 |
| 3 | 1 1 0 0 1 | 625 | 1 1 0 0 1 | 625 |
| 4 | 1 0 1 0 1 | 441 | 1 0 1 0 1 | 441 |

* 1. **Reprezentarea cromozomială**

Reprezentarea cromozomială presupune identificarea spaţiului fenotipurilor şi definirea spaţiului genotipurilor, între cele două spaţii existând o corespondenţă biunivocă. Alegerea unei anumite reprezentări depinde de problema particulară de rezolvat. Problema alegerii unei reprezentări adecvate este una din cele mai importante şi dificile în dezvoltarea unui algoritm evolutiv care să furnizeze o soluţie foarte apropiată de cea optimă. În continuare sunt prezentate o serie dintre cele mai utilizate modalităţi de reprezentare cromozomială.

**Reprezentarea binară**

Reprezentarea binară este una dintre primele (ca istoric) dintre cele utilizate în construirea unui GA. În literatura de specialitate au fost raportate o serie de situaţii de utilizare abuzivă a reprezentării binare, aproape independent de problema de rezolvat, fapt care a condus la dezvoltări GA eronate.

Pentru o problemă particulară, prima etapă în construcţia unei reprezentări binare este stabilirea dimensiunii reprezentării şi semnificaţia acesteia în obţinerea unui fenotip. În alegerea corespondenţei fenotip-genotip în reprezentarea binară trebuie ţinut cont de faptul că modul de codificare trebuie să asigure faptul că fiecare şir de biţi din spaţiul genotipurilor corespunde unei soluţii valide (fezabile) şi, reciproc, în spaţiul genotipurilor pot fi reprezentate toate soluţiile posibile. În practică, soluţiile fezabile sunt cele care respectă constrângerile problemei particulare de rezolvat, generarea lor nefiind întotdeauna posibilă. În literatura de specialitate au fost raportate o serie de strategii de tratare a problemelor de optimizare cu restricţii, din punctul de vedere al spaţiului soluţiilor fezabile (Eiben, Smith, 2003; Michalewicz, Schoenauer, 1996; Joines, Houck, 1994; Goldberg, 1989).

În cazul unor clase de probleme care conţin variabile de tip decizie booleană, reprezentarea binară este cea mai naturală (de exemplu, problema de tip 0-1 a rucsacului, prezentată în §2.4). Dacă informaţia codificată nu este de tip boolean, poate fi reprezentată de exemplu similar celor prezentate în §2.3 şi §3.1, dar, în multe situaţii, pot fi obţinuţi GA superiori ca performanţe prin utilizarea reprezentărilor directe (pe mulţimile sau ), similar primelor două probleme rezolvate în §2.4.

Una din problemele utilizării reprezentării binare este aceea că biţi diferiţi au în general semnificaţii diferite. Reprezentarea binară standard a numerelor din are dezavantajul că distanţa Hamming între două numere întregi consecutive nu este întotdeauna 1, unde distanţa Hamming între două şiruri de aceeaşi dimensiune este definită ca numărul de poziţii ale căror valori corespunzătoare diferă. De exemplu, în reprezentarea prezentată în §3.1, numărului îi corespunde reprezentarea . Probabilitatea ca, prin *k* modificări succesive ale valorii unui bit, să fie obţinut numărul „vecin” , reprezentat prin şirul este evident inferioară probabilităţii de a fi obţinut numărul „vecin” , reprezentat prin şirul . Distanţa Hamming dintre reprezentările binare ale fenotipurilor şi este 5 (maximă), în timp ce distanţa Hamming dintre reprezentările binare ale fenotipurilor şi este 1. O variantă de reprezentare binară care să rezolve această problemă (în cazul fenotipurilor numere din mulţimea ) este codificarea Gray.

Codificarea Gray furnizează reprezentări ale numerelor din prin intermediul şirurilor binare astfel încât distanţa Hamming dintre oricare două numere întregi consecutive să fie 1. Fie . Fie şi reprezentarea binară standard, dată de un şir de biţi: . Reprezentarea Gray a lui, , este obţinută astfel

unde este operatorul logic „sau exclusiv”. De exemplu, dacă , atunci şi reprezentările binare standard şi respectiv Gray sunt prezentate în tabelul următor.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Număr** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
| Cod binar std. | 0 0 0 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 |
| Cod Gray | 0 0 0 0 | 0 0 0 1 | 0 0 1 1 | 0 0 1 0 | 0 1 1 0 | 0 1 1 1 | 0 1 0 1 | 0 1 0 0 |
| **Număr** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| Cod binar std. | 1 0 0 0 | 1 0 0 1 | 1 0 1 0 | 1 0 1 1 | 1 1 0 0 | 1 1 0 1 | 1 1 1 0 | 1 1 1 1 |
| Cod Gray | 1 1 0 0 | 1 1 0 1 | 1 1 1 1 | 1 1 1 0 | 1 0 1 0 | 1 0 1 1 | 1 0 0 1 | 1 0 0 0 |

În următoarele funcţii MATLAB sunt prezentate obţinerea codificării Gay a unui număr din , respectiv obţinerea acelui număr din a cu proprietatea că îi corespunde un cod Gray dat.

function [y]=repr\_int\_Gray(x,m);

y=zeros(1,m);

y(1)=bitget(x,m);

for i=2:m

y(i)=xor(bitget(x,m-i+1),bitget(x,m-i+2));

end;

disp(y);

end

function [x]=repr\_Gray\_int(y,m);

z=zeros(m,1);

z(1)=y(1);

val=z(1);

for i=2:m

if(y(i))

val=not(val);

end;

z(i)=val;

end;

x=0;

for i=1:m

x=bitset(x,m-i+1,z(i));

end;

disp(x);

end

**Reprezentarea prin intermediul numerelor întregi**

Aşa cum am precizat în secţiunea de mai sus, reprezentarea binară nu este întotdeauna potrivită în cazul problemelor în care unui fenotip îi corespunde un cromozom în care fiecare genă are mai mult de 2 valori (nu este de tip boolean). O astfel de reprezentare este utilizată, de exemplu, în cadrul problemelor de optimizare a funcţiilor cu variabile luând valori în mulţimea numerelor întregi. Problema de optimizare poate fi fără restricţii (o variabilă poate avea ca valoare orice număr întreg) sau, în cele mai multe cazuri, poate fi astfel încât variabilele să fie restricţionate la un set finit de valori. De exemplu, problema determinării unui drum de la un punct de start *S* într-o reţea de puncte în care conexiunile posibile sunt sus, jos, stânga şi respectiv dreapta (nord, sud, vest, est) revine la determinarea unei secvenţe de deplasări către nord, sud, vest, respectiv est cu câte o unitate (corespunzătoare unei conexiuni), fiecare tip de deplasare putând fi codificată cu o valoare din mulţimea cu elemente (fiecare element corespunde unui punct cardinal).

Alegerea unei codificări şi a operatorilor de variaţie definiţi pe mulţimea genotipurilor trebuie făcută, dacă este posibil, pe baza unor relaţii naturale între valorile pe care le poate lua fiecare atribut (variabilă). Astfel de relaţii pot fi evidenţiate uşor în cazul atributelor ordinale (de exemplu cu valori într-o mulţime de numere întregi: de exemplu 2 este „mai similar “ cu 3 decât cu 312 ), dar pot fi greu de identificat în cazul atributelor cardinale (de exemplu, dacă punctele cardinale sunt reprezentate prin numerele 0,1,2 şi 3, faptul 0 este „mai similar“ cu 1 decât cu 3 pe mulţimea nu poate fi transpus corespunzător pentru elementele din spaţiul punctelor cardinale).

**Reprezentarea prin intermediul numerelor reale**

În multe situaţii practice, reprezentarea unui candidat la soluţie este realizată prin intermediul unui şir de numere reale. În general este preferată această reprezentare atunci când fenotipurile provin dintr-o distribuţie continuă. Genotipul corespunzător unui candidat soluţie cu *p* gene este deci vectorul , unde .

**Reprezentarea prin intermediul permutărilor**

Permutările sunt utilizate pentru reprezentări cromozomiale în probleme în care trebuie stabilită ordinea apariţiei unor secvenţe de evenimente. O consecinţă imediată este că, în timp ce într-un GA clasic este permis ca gene diferite să aibă aceleaşi valori (alele), în cazul reprezentării prin permutări, cromozomii cu proprietatea că două sau mai multe gene posedă aceeaşi valoare nu corespund unor soluţii fezabile. Evident, în acest caz operatorii de variaţie trebuie definiţi astfel încât rezultatul aplicării acestora să corespundă unor soluţii admisibile.

În alegerea operatorilor de variaţie este importantă clasa din care face parte problema particulară de rezolvat. În general, problemele care sunt rezolvate genetic utilizând cromozomi permutări sunt de două tipuri. În prim clasă de probleme, ordinea apariţiei evenimentelor este importantă. Această situaţie apare, de exemplu, când evenimentele utilizează resurse limitate sau se desfăşoară într-o anumită perioadă de timp. Cea mai comună problemă din această clasă este problema planificării activităţilor, a cărei rezolvare prin intermediul GA este prezentată în finalul capitolului. O a doua clasă de probleme este cea în care apare *dependenţa de adiacenţă*, cel mai comun exemplu din această clasă fiind problema comis-voiajorului, în care, pentru *n* oraşe interconectate date, trebuie determinat un drum de lungime (distanţă) minimă care să treacă prin toate cele *n* oraşe, cu revenire în oraşul de start. Diferenţa dintre cele două clase de probleme este dată de faptul că, evident, în cazul celei de-a doua clase, punctul de start (în cazul problemei comis-voiajorului oraşul de plecare) nu este important.

Pentru reprezentarea permutărilor sunt utilizate în general două variante: în prima variantă, cel de-al *i*-lea element al reprezentării semnifică evenimentul care se produce în acea poziţie a secvenţei, iar în varianta alternativă, valoarea celui de-al *i*-lea element reprezintă poziţia din secvenţă în care se produce evenimentul *i*. De exemplu, în prima variantă de reprezentare şirul corespunde, prin permutarea , şirului , în timp ce în varianta alternativă de reprezentare corespunde şirului ( este în poziţia 3, este în poziţia 1, este în poziţia 4 şi este în poziţia 2).

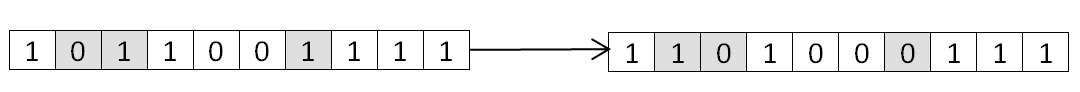
* 1. **Operatorul mutaţie**

Mutaţia este acel operator genetic prin care, dintr-un singur părinte este obţinut un singur copil prin aplicarea unei modificări aleatoare a reprezentării (genotipului). Operatorul mutaţie depinde de modalitatea de reprezentare cromozomială şi are asociat un parametru, *pm*, care reprezintă rată de aplicare. În continuare vom prezenta o serie de modalităţi de alegere a operatorului mutaţie în funcţie de reprezentarea cromozomială. O serie de consideraţii privind setarea ratei de mutaţie sunt prezentate în capitolul 5.

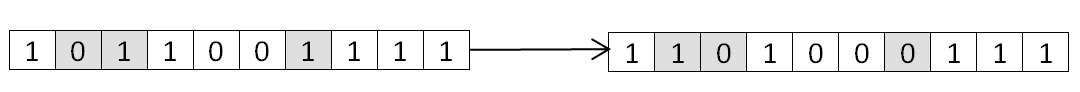
**Operatorul mutaţie în reprezentarea binară**

Cel mai utilizat operator de mutaţie în reprezentarea binară consideră fiecare genă a fiecărui cromozom pentru inversarea valorii asociate (din 0 în 1, respectiv din 1 în 0) cu o probabilitate *pm* în general mică. Numărul valorilor modificate nu este fixat, dar, în medie, dacă populaţia conţine *dim* cromozomi cu câte *n* gene, este egal cu .

De exemplu, dacă în cromozomul părinte



este aplicat operatorul de mutaţie genelor 2, 3 şi 7, rezultă cromozomul copil



Un exemplu de astfel de operator este utilizat în §3.1 şi implementat prin intermediul funcţiei MATLAB *mutatie(pop,pm)*.

Setarea ratei de mutaţie depinde în general de rezultatul dorit, în sensul în care aplicaţia trebuie să conducă fie la obţinerea unei populaţii în care toţi indivizii sunt buni (valoare mare a funcţiei de evaluare), fie la obţinerea unei populaţii cu un singur individ cu adaptare foarte bună la mediu. În general, în cele mai multe dezvoltări GA care folosesc reprezentarea binară pentru construirea spaţiului genotipurilor, parametrul *pm* este setat în următoarele limite: într-o generaţie, în medie, este modificată o singură genă a unui singur cromozom din populaţie, respectiv în populaţia curentă este modificată, în medie, câte o genă din fiecare cromozom.

**Operatorul mutaţie în reprezentarea prin şiruri de numere întregi**

În cazul reprezentării cromozomiale pe mulţimea numerelor întregi sunt două modalităţi uzuale de a defini mutaţia.

Prima dintre ele este *resetarea aleatoare* prin care, cu probabilitatea *pm*, valoarea fiecărei gene este modificată prin generarea aleatoare a unei valori din mulţimea valorilor admisibile pentru gena respectivă. Acest operator este utilizat în situaţia în care genele codifică atribute de tip cardinal (în acest caz valorile admisibile ale unei gene trebuie să poată fi selectate cu aceeaşi probabilitate într-o operaţie de mutaţie).

În continuare este prezentată implementarea unui operator de mutaţie prin resetare aleatoare pentru o populaţie cu *dim* indivizi *n*-dimensionali, fiecare atribut fiind cu valori în mulţimea de numere întregi . Probabilitatea de mutaţie este *pm*.

function [popN]=mutatie\_intregi\_ra(pop,pm,M,N);

[dim,n]=size(pop);

popN=pop;

for i=1:dim

efectuat=0;

for k=1:n

r=unifrnd(0,1);

if(r<pm)

disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));

disp('Gena');disp(k);

% este generat aleator un numar intreg intre M si N:

% genereaza aleator un numar R intre 1 si N-M+1

% aduna la R valoarea M-1

popN(i,k)=M-1+unidrnd(N-M+1);

efectuat=1;

end;

end;

if(efectuat)

disp('Cromozom rezultat');disp(popN(i,:));

end;

end;

end

function []=test\_mutatie\_intregi\_ra(dim,n,pm,M,N);

% generarea populatiei care sufera mutatie

pop=zeros(dim,n);

for i=1:dim

for k=1:n

pop(i,k)=unidrnd(N-M+1)+M-1;

end;

end;

disp('Populatia initiala');disp(pop);

%aplicarea mutatiei

popN=mutatie\_intregi\_ra(pop,pm,M,N);

end

Exemplu de aplicare: la un apel test\_mutatie\_intregi\_ra(6,4,1/6,-100,230), pot fi obţinute următoarele rezultate:

Populatia initiala

118 -40 110 -44

78 106 127 124

190 -96 2 157

1 206 124 -76

-77 -97 -25 70

51 132 92 68

Mutatie efectuata in cromozomul

118 -40 110 -44

Gena 1

Cromozom rezultat

-37 -40 110 -44

Mutatie efectuata in cromozomul

190 -96 2 157

Gena 4

Cromozom rezultat

190 -96 2 168

Mutatie efectuata in cromozomul

1 206 124 -76

Gena 2

Cromozom rezultat

1 -60 124 -76

Mutatie efectuata in cromozomul

51 132 92 68

Gena 4

Cromozom rezultat

51 132 92 -74

O altă variantă de definire este prin intermediul *mutaţiei de tip fluaj*. Operatorul este utilizat în cazul atributelor de tip ordinal şi, cu o probabilitate *p*, adaugă/scade o anumită cantitate (de obicei mică) la/din valoarea fiecărei gene a fiecărui cromozom. În general valorile cu care sunt modificate alelele sunt generate aleator dintr-o distribuţie de medie 0 şi astfel încât probabilitatea apariţiei unei valori mici să fie mult mai mare decât probabilitatea generării unei valori mari. Mutaţia de tip fluaj necesită setarea parametrilor ce caracterizează distribuţia de probabilitate (ce fel de distribuţie este, ce statistici sunt cunoscute, probabilitatea *p*) din care sunt generate valorile ce modifică alelele, deci a numărului de paşi efectuaţi de operatorul mutaţie în spaţiul de căutare. Determinarea unei setări potrivite a acestor parametri este în general un proces dificil, în general fiind preferată utilizarea în tandem a mai multor operatori de mutaţie de acest tip. De aceea, o variantă alternativă este de a utiliza un operator mutaţie resetare aleatoare cu rată foarte mică împreună cu un operator fluaj care să efectueze modificări mici asupra alelelor.

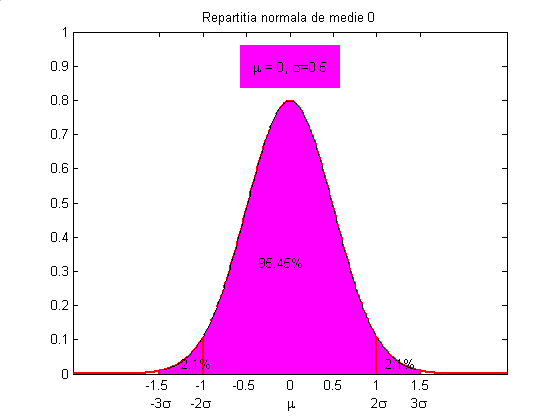
Un exemplu de implementare a unui operator de tip fluaj este prezentat în următoarea secţiune.

**Operatorul mutaţie în reprezentarea prin şiruri de numere reale**

În cazul reprezentării cromozomiale prin şiruri de numere din reprezentate în virgulă mobilă sunt utilizaţi în general următorii operatori. Fie *n* numărul genelor unui cromozom şi presupunem că, pentru , valoarea genei *i* aparţine intervalului şi, prin aplicarea operatorului mutaţie asupra cromozomului este obţinut , unde .

*Mutaţia uniformă* presupune modificarea fiecărei gene din fiecare cromozom, cu probabilitatea *pm*, prin generarea aleatoare uniformă a câte unui număr din . Deci, dacă gena *i,*  a cromozomului *x* a fost selectată pentru mutaţie, este generat aleator, cu distribuţie uniformă, în . Altfel, . Acest tip de operator este corespondentul mutaţiei prin resetare aleatoare în cazul reprezentării cromozomiale prin şiruri de numere întregi.

*Mutaţia neuniformă, cu distribuţie fixată*, este una dintre cele mai des utilizate în reprezentările cromozomiale în virgulă mobilă şi este corespondentul operatorului de mutaţie fluaj. Operatorul este proiectat astfel încât cantitatea de informaţie modificată să fie mică. Aplicarea lui asupra unei gene *i, ,* presupune modificarea valorii curente cu o valoare generată de obicei din distribuţia gaussiană cu medie 0 şi deviaţie standard specificată, , (distribuţie notată ) rezultatul fiind apoi adus pe intervalul . Motivul alegerii unei astfel de distribuţii este următorul. Probabilitatea ca un număr generat aleator să fie în , respectiv în este aproximativ 95.45%, respectiv 99.73%. În figura de mai jos este prezentată grafic această proprietate, pentru distribuţia .



Cu alte cuvinte, deoarece valorile cu care sunt modificate alelele trebuie să fie, de exemplu, inferioare unui prag dat *t*, deci în intervalul , atunci ele pot fi generate din distribuţia , unde .

**Observaţie.** În general, pentru distribuţia , au loc proprietăţile

O variantă de implementare a acestui operator pentru o deviaţie standard dată, , este prezentată în continuare. Pentru uşurinţa înţelegerii operatorului, am considerat .

function [popN]=mutatie\_creep\_fp(pop,pm,a,b,sigma);

[dim,n]=size(pop);

popN=pop;

for i=1:dim

efectuat=0;

for k=1:n

r=unifrnd(0,1);

if(r<pm)

disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));

disp('Gena');disp(k);

% R este generat normal, cu medie 0 si deviatie sigma

R=normrnd(0,sigma);

popN(i,k)=pop(i,k)+R;

if(popN(i,k)<a)

popN(i,k)=a;

else if (popN(i,k)>b)

popN(i,k)=b;

end;

end;

efectuat=1;

end;

end;

if(efectuat)

disp('Cromozom rezultat');disp(popN(i,:));

end;

end;

end

function []=test\_mutatie\_creep\_fp(dim,n,pm,a,b,sigma);

% generarea populatiei care sufera mutatie este realizata

% aleator, uniforma pe [a,b]

pop=zeros(dim,n);

for i=1:dim

for k=1:n

pop(i,k)=unifrnd(a,b);

end;

end;

disp('Populatia initiala');disp(pop);

%aplicarea mutatiei

popN=mutatie\_creep\_fp(pop,pm,a,b,sigma);

end

Exemplu de aplicare: la un apel test\_mutatie\_creep\_fp(8,4,1/8,-2,3.5,0.3) pot fi obţinute următoarele rezultate:

Populatia initiala

-1.9676 1.1593 2.0915 2.4376

1.5066 -0.6180 -1.2062 1.5835

3.2035 2.4874 3.1164 -0.2953

-0.5215 0.9505 -1.1020 -0.8396

-0.8075 1.5848 -1.7097 -0.7389

1.6709 -0.2899 -0.3135 1.9637

3.2492 -1.2787 -1.6245 -1.3113

-1.0862 3.0128 -1.2506 1.3935

Mutatie efectuata in cromozomul

-0.8075 1.5848 -1.7097 -0.7389

Gena 4

Cromozom rezultat

-0.8075 1.5848 -1.7097 -0.6407

Mutatie efectuata in cromozomul

3.2492 -1.2787 -1.6245 -1.3113

Gena 3

Cromozom rezultat

3.2492 -1.2787 -1.5721 -1.3113

Mutatie efectuata in cromozomul

-1.0862 3.0128 -1.2506 1.3935

Gena 4

Cromozom rezultat

-1.0862 3.0128 -1.2506 1.3375

**Operatorul mutaţie în reprezentarea prin permutări**

În cazul reprezentării cromozomiale prin permutări, operatorul mutaţie poate fi aplicat doar la nivel de cromozom (aplicarea lui la nivel de genă determină obţinerea unui şir care nu este permutare), deci rata de mutaţie corespunde probabilităţii modificării unui genotip, nu a unei gene. Cele mai utilizate variante de definire a mutaţiei sunt prezentate în continuare (Syswerda, 1992).

*Mutaţia prin interschimbare* revine la selectarea aleatoare a două gene dintr-un cromozom şi interschimbarea valorilor asociate acestora. Un exemplu de implementare a fost prezentat în §2.1 (rezolvarea problemei celor 8 regine).

*Mutaţia prin inserare* presupune selectarea aleatoare a două gene, , memorarea alelei din gena *j*, *A*, deplasarea cu o poziţie către dreapta a valorilor genelor şi plasarea valorii *A* pe poziţia *i*+1.Evident, dacă , mutaţia nu modifică nimic în cromozom.

Următoarele funcţii MATLAB implementează acest tip de mutaţie.

function [popN]=mutatie\_perm\_inserare(pop,pm);

[dim,n]=size(pop);

popN=pop;

for i=1:dim

efectuat=0;

r=unifrnd(0,1);

if(r<pm)

disp('Mutatie efectuata in cromozomul'); disp(pop(i,:));

p=zeros(1,2);

p(1)=unidrnd(n);

p(2)=unidrnd(n);

while(p(1)==p(2))

p(2)=unidrnd(n);

end;

poz=sort(p);

disp('Pozitiile:');

disp(poz);

popN(i,1:poz(1))=pop(i,1:poz(1));

popN(i,poz(1)+1)=pop(i,poz(2));

popN(i,poz(1)+2:poz(2))=pop(i,poz(1)+1:poz(2)-1);

popN(i,poz(2)+1:n)=pop(i,poz(2)+1:n);

disp('Cromozom rezultat');disp(popN(i,:));

end;

end;

end

function []=test\_mutatie\_perm\_inserare(dim,n,pm);

% presupunem ca un cromozom este o permutarea a mutimii {1,2,...,n}

% dim este dimensiunea populatiei

% generarea populatiei care sufera mutatie

% este similara problemei celor n regine

pop=genereaza\_ini(n,dim);

disp('Populatia initiala');disp(pop);

%aplicarea mutatiei

[popN]=mutatie\_perm\_inserare(pop,pm);

end

La un apel test\_mutatie\_perm\_inserare(10,8,0.2) poate fi obţinut următorul rezultat.

Populatia initiala

5 1 6 4 8 3 7 2

3 4 2 1 7 5 6 8

3 6 1 8 7 5 2 4

3 8 7 4 2 5 6 1

2 8 6 3 1 5 7 4

3 1 7 8 5 6 2 4

7 4 2 6 8 3 1 5

4 7 2 6 8 1 3 5

4 1 7 8 3 5 2 6

5 7 8 1 3 4 2 6

Mutatie efectuata in cromozomul

3 6 1 8 7 5 2 4

Pozitiile:

5 8

Cromozom rezultat

3 6 1 8 7 4 5 2

Mutatie efectuata in cromozomul

4 1 7 8 3 5 2 6

Pozitiile:

1 5

Cromozom rezultat

4 3 1 7 8 5 2 6

Mutatie efectuata in cromozomul

5 7 8 1 3 4 2 6

Pozitiile:

1 2

Cromozom rezultat

5 7 8 1 3 4 2 6

*Mutaţia prin amestec* presupune repoziţionarea fiecărei alele în mod aleatoriu, pentru o secvenţă de gene ale unui cromozom sau pentru întreg cromozomul. De exemplu, pentru cromozomul

4 3 **1 7 8 5** 2 6

prin amestecul aleator al valorilor genelor 3,4,5 şi 6 poate rezulta cromozomul copil

4 3 **7 1 5 8** 2 6.

*Mutaţia prin inversiune* presupune selectarea aleatoare a două gene, , şi inversarea ordinii în secvenţa dintre cele două poziţii. De exemplu, pentru cromozomul

4 3 **1** 7 8 **5**  2 6

şi poziţiile 3 şi 6, rezultă cromozomul copil

4 3 **5 8 7 1** 2 6.

În rezolvarea problemelor bazate pe dependenţa de adiacenţă (din clasa problemei comis-voiajorului), inversiunea unui subşir aleator într-un cromozom este cea mai mică modificare ce poate fi adusă acelui cromozom şi corespunde reprezentării Gray în cazul spaţiului genotipurilor şiruri binare. Această variantă este cea mai des utilizată când datele problemei de rezolvat sunt reprezentate prin permutări.

* 1. **Operatorul de recombinare**

Recombinarea (încrucişarea) este procesul creării de noi soluţii candidat pe baza informaţiei conţinute în două (sau mai multe) soluţii candidat părinţi şi este una dintre cele mai importante caracteristici ale GA.

Operatorul de recombinare este de obicei aplicat în manieră probabilistă, cu o probabilitate (rată) *pc*, setată în general în intervalul . Procesul presupune selectarea a două soluţii candidat părinte, generarea aleatoare a unui număr din , *r*, şi compararea lui cu rata de recombinare, *pc*. Dacă , indivizii noi sunt creaţi prin *recombinarea celor doi părinţi*, în caz contrar sunt creaţi *asexuat*, adică prin *duplicarea părinţilor*.

Spre deosebire de rata de mutaţie, *pm*, prin care este controlată secvenţa de gene modificate, rata de încrucişare, *pc*, corespunde şansei ca perechea de părinţi selectată să fie supusă operaţiei de recombinare.

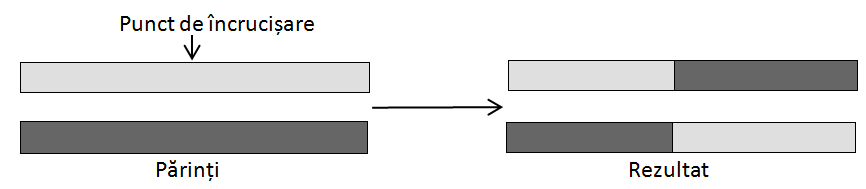
**Recombinarea în reprezentarea prin şiruri binare**

În cazul reprezentării binare există trei modalităţi uzuale de definire a operatorului de recombinare, toate generând o pereche de progenituri pe baza unei perechi de cromozomi părinte. În literatura de specialitate sunt menţionate, de asemenea, extensii ale acestora la cazul utilizării *n*-tuplurilor de părinţi, , respectiv la situaţia obţinerii unei singure progenituri (Eiben, van Kemenade, Kok, 2000).

*Încrucişarea uni-punct* este primul operator propus pentru recombinare, introdus de Holland în 1975. Recombinarea este aplicată câte unei perechi de cromozomi părinte şi presupune efectuarea următoarelor operaţii: selectarea aleatoare a unei gene, *poz*, şi obţinerea progeniturile astfel :

* copiază primele *poz* elemente din , respectiv în , respectiv
* copiază în ultimele *m-poz+1* poziţii din , respectiv , ultimele *m-poz+1* elementele din , respectiv , unde *m* este lungimea reprezentării

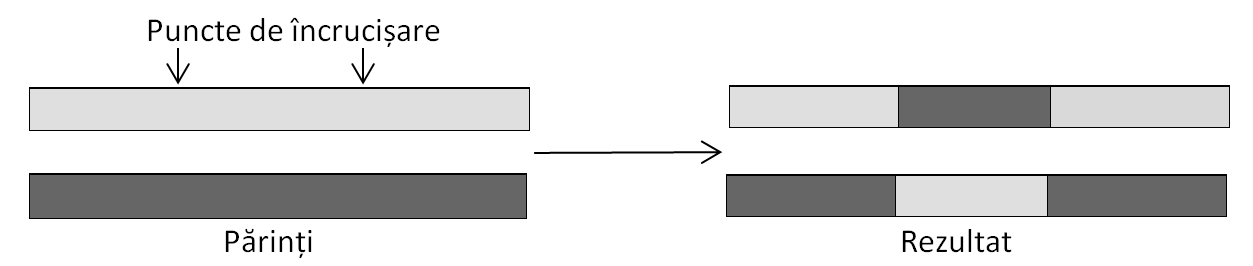
În figura următoare este reprezentată încrucişarea uni punct.



Un exemplu de implementare a acestui tip de operator a fost prezentat în §2.4 (în rezolvarea problemei rucsacului de tip 0-1).

*Încrucişarea multipunct* este o extensie naturală a operatorului de recombinare uni punct. Recombinarea este aplicată câte unei perechi de cromozomi părinte şi presupune selectarea aleatoare a *N* gene şi obţinerea progeniturile prin copierea alternativă a secvenţelor din cromozomii fiecărui părinte.

În figura următoare este reprezentată încrucişarea multipunct pentru situaţia .



*Încrucişarea uniformă* (Syswerda, 1989) este aplicată câte unei perechi de cromozomi părinte şi presupune obţinerea progeniturile prin consultarea fiecărei gene în mod independent şi atribuirea valorii corespunzătoare din gena unuia dintre părinţi, ales aleator. Implementarea este realizată în felul următor. Fie *m* este dimensiunea reprezentării prin şiruri binare şi *p* un parametru dat (în general ). Pentru fiecare genă , este generată aleator, din distribuţia uniformă, o valoarea *v*. Dacă , atunci , cu alte cuvinte primul descendent direct moşteneşte valoarea genei de la primul părinte, iar cel de-al doilea de la al doilea părinte. În caz contrar, copierea este inversă, . Următoarea funcţie MATLAB implementează acest tip de recombinare. Deoarece prin această funcţie dorim să exemplificăm operaţia de încrucişare uniformă şi nu modul de selecţie a părinţilor, vom presupune că populaţia de părinţi este disponibilă (în cazul nostru generată aleator pentru dimensiunea reprezentării *m* şi volumul datelor *dim*) şi împerecherea se face în ordinea apariţiei indivizilor în populaţie.

function [popN]=crossover\_rbinara\_uniform(pc,dim,m,p);

% pop este populatia de parinti

pop=genereaza\_ini\_binar(m,dim);

disp('Populatia de parinti:');

disp(pop);

[dim,m]=size(pop);

popN=pop;

for k=1:2:dim

x1=pop(k,1:m);

y1=pop(k+1,1:m);

r=unifrnd(0,1);

disp('Parintii:');

disp(x1);disp(y1);

if(r<=pc)

disp('Incrucisarea este realizata pe baza valorilor:');

for i=1:m

v=unifrnd(0,1);

disp(v);

if(v<=p)

popN(k,i)=pop(k,i);

popN(k+1,i)=pop(k+1,i);

else

popN(k,i)=pop(k+1,i);

popN(k+1,i)=pop(k,i);

end;

end;

else

disp('Incrucisare asexuata');

end;

disp('Urmasii:');

disp(popN(k,1:m));disp(popN(k+1,1:m));

end;

end

La un apel crossover\_rbinara\_uniform(0.7,8,6,0.5) poate fi obţinut următorul rezultat.

Populatia de parinti:

0 0 1 1 0 1

0 0 1 1 0 1

1 0 0 1 1 1

0 1 0 0 1 0

0 0 1 1 0 1

0 0 0 0 0 1

1 1 0 0 0 0

0 1 1 1 0 1

Parintii:

0 0 1 1 0 1

0 0 1 1 0 1

Incrucisare asexuata

Urmasii:

0 0 1 1 0 1

0 0 1 1 0 1

Parintii:

1 0 0 1 1 1

0 1 0 0 1 0

Incrucisarea este realizata pe baza valorilor:

0.4186 0.8462 0.5252 0.2026 0.6721 0.8381

Urmasii:

1 1 0 1 1 0

0 0 0 0 1 1

Parintii:

0 0 1 1 0 1

0 0 0 0 0 1

Incrucisarea este realizata pe baza valorilor:

0.6813 0.3795 0.8318 0.5028 0.7095 0.4289

Urmasii:

0 0 0 0 0 1

0 0 1 1 0 1

Parintii:

1 1 0 0 0 0

0 1 1 1 0 1

Incrucisarea este realizata pe baza valorilor:

0.1897 0.1934 0.6822 0.3028 0.5417 0.1509

Urmasii:

1 1 1 0 0 0

0 1 0 1 0 1

**Recombinarea în reprezentarea prin şiruri de numere întregi**

În cazul în care spaţiul genotipurilor este format din şiruri de numere întregi, operatorii de încrucişare sunt similari celor definiţi pentru reprezentarea prin şiruri binare (Eiben, Smith, 2003). O altă alegere, de exemplu recombinarea aritmetică (prezentată în secţiunea următoare), poate avea ca rezultat cromozomi care nu aparţin spaţiului soluţiilor fezabile (nu sunt şiruri de numere întregi).

**Recombinarea în reprezentarea prin şiruri de numere reale (virgulă mobilă)**

În acest caz sunt două clase de operatori de încrucişare folosiţi uzual, şi anume recombinarea de tip discret şi recombinarea aritmetică.

*Recombinarea de tip discret* este clasa operatorilor definiţi similar reprezentării prin şiruri binare, cu menţiunea că, în acest caz, o alelă este un număr real, nu 0 sau 1. Acest tip de încrucişare are dezavantajul că nu sunt obţinute valori noi pentru genele cromozomilor urmaşi, singurul operator care poate introduce valori noi ale genelor rămânând cel de mutaţie.

*Recombinarea aritmetică* este clasa operatorilor prin aplicarea cărora sunt obţinuţi cromozomi cu alele noi, rezultate prin combinarea aritmetică (prin medie ponderată) a alelelor cromozomilor părinţi. În literatura de specialitate sunt prezentate o serie de variante ale operatorilor aparţinând acestei clase, cei mai utilizaţi fiind prezentaţi în continuare (Michalewicz, 1996). Vom considera în continuare şi *m* dimensiunea reprezentării (numărul de gene ale unui cromozom). Acest parametru este fie generat aleatoriu, fie predefinit (în general în jurul valorii ).

*Recombinarea aritmetică simplă* este aplicată câte unei perechi de cromozomi părinte şi presupune obţinerea progeniturile prin selectarea aleatoare a unei gene , şi efectuarea operaţiilor

* copiază primele *i*-1 elemente din , respectiv în , respectiv

De exemplu, dacă şi

rezultă

*Recombinarea aritmetică singulară* operează asupra unei perechi de cromozomi părinte şi determină obţinerea cromozomilor în modul următor. Este selectată aleator o genă , şi sunt efectuate următoarele operaţii:

* copiază primele *i*-1 elemente din , respectiv în , respectiv
* copiază în ultimele *m-i* poziţii din , respectiv , ultimele *m-i* elementele din , respectiv

De exemplu, dacă şi

rezultă

Un exemplu de implementare a acestui tip de operator pentru a fost prezentat în §2.4 (în rezolvarea problemei optimizării unei funcţii de o variabilă).

*Recombinarea aritmetică totală* este descrisă astfel. Fie o pereche de cromozomi părinte. Progeniturile sunt obţinute prin:

Evident, pentru , cei doi cromozomi rezultaţi sunt identici.

De exemplu, dacă şi

rezultă

**Recombinarea în reprezentarea prin permutări**

În cazul în care spaţiul genotipurilor este format din permutări, au fost propuşi o serie de operatori de recombinare, care au ca scop transmiterea unei cantităţi cât mai mare de informaţie de la părinţi la copii, cu precădere din categoria informaţiei comune prezente în cromozomii părinţi. În continuare sunt prezentaţi câţiva dintre cei mai utilizaţi operatori de recombinare pe spaţiul genotipurilor permutări.

*Operatorul PMX* (**P**artially **M**apped Crossover) a fost propus în (Goldberg, Lingle, 1985) pentru rezolvarea problemei comis-voiajorului şi este unul dintre cei mai utilizaţi operatori de recombinare pentru rezolvarea problemelor bazate pe dependenţa de adiacenţă. Fie o pereche de cromozomi părinte, progeniturile calculate şi *m* numărul genelor unui cromozom. O variantă de implementare a acestui operator este descrisă astfel (Whitley, 2000).

Pas 1. Alege aleator puncte de încrucişare, efectuează

Pas 2. Calculează mulţimea alelelor din secvenţa de gene din cel de-al doilea părinte, care nu au fost deja copiate în prima progenitură,

Pas 3. Pentru fiecare , scopul este de a copia *a* în prima progenitură. Pentru aceasta putem proceda astfel: calculează (*b* este valoarea genei *p* din prima progenitură, , care a fost copiată din poziţia alelei *a* din cel de-al doilea părinte; din construcţia mulţimii *A*, rezultă că valoarea *a* nu a fost copiată încă în ) şi cu proprietatea că (poziţia acestei gene, *b*, din cel de-al doilea părinte).

3.1 Dacă în poziţia *i* din prima progenitură nu a fost plasată nici o valoare ( nu este „completată” ), setează (*a* poate fi plasată în gena *i*).

3.2 În caz contrar, determină şi cu proprietatea că ( este deja „completată” )

Dacă poziţia *j* în este neocupată, setează .

Altfel i=j şi reia 3.2.

Pas 4. La acest moment au fost copiate în prima progenitură alelele secvenţei de gene din ambii părinţi. Genele rămase fără valori în prima progenitură sunt ocupate, în ordine, de alelele încă necopiate din cel de-al doilea părinte.

Cel de-al doilea copil este construit similar, plecând de la secvenţa de alele din cel de-al doilea părinte şi copiind restul elementelor din primul părinte.

De exemplu, fie ,

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 5 | 7 | 3 | 9 | 10 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 5 | 3 | 7 | 9 | 2 | 1 | 8 | 6 | 4 |

Construcţia primului copil, , este realizată astfel.

După aplicarea primului pas, obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 6 | 5 | 7 | 3 |  |  |  |

La pasul 2 rezultă

La pasul 3:

Deoarece şi gena 2 din nu are încă valoare, setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 9 |  | 6 | 5 | 7 | 3 |  |  |  |

Deoarece şi gena 4 din are valoarea , rezultă deci şi setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 9 |  | 6 | 5 | 7 | 3 |  | 2 |  |

Deoarece şi gena 3 din nu are încă valoare, setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 9 | 1 | 6 | 5 | 7 | 3 |  | 2 |  |

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

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 9 | 1 | 6 | 5 | 7 | 3 | 8 | 2 | 4 |

**Observaţie.** Dintre relaţiile de adiacenţă comune celor doi părinţi, între 1 şi 2, între 4 şi 6 şi între 3 şi 7, nu este regăsită în primul urmaş relaţia de adiacenţă dintre 6 şi 4, deci *operatorul PMX nu are neapărat proprietatea de a păstra relaţiile comune existente între părinţi*. Operatorii de recombinare descrişi pentru reprezentările prin şiruri binare, respectiv prin şiruri de numere întregi, dar şi operatorul de recombinare de tip discret definit pentru reprezentări prin şiruri de numere reale posedă proprietatea de a păstra intacte în urmaşi relaţiile de adiacenţă comune existente în cromozomii părinte.

Construcţia celui de-al doilea copil, , este realizată astfel.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 5 | 7 | 3 | 9 | 10 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 5 | 3 | 7 | 9 | 2 | 1 | 8 | 6 | 4 |

După aplicarea primului pas, obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 7 | 9 | 2 | 1 |  |  |  |

La pasul 2 rezultă

La pasul 3:

Deoarece şi gena 6 din are valoarea , rezultă deci şi setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 6 |  | 7 | 9 | 2 | 1 |  |  |  |

Deoarece şi gena 8 din nu are încă valoare, setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 6 |  | 7 | 9 | 2 | 1 | 5 |  |  |

Deoarece şi gena 1 din nu are încă valoare, setează . Obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 6 |  | 7 | 9 | 2 | 1 | 5 |  |  |

La pasul 4 vor fi plasate în , în ordinea genelor rămase fără valori, alelele 4, 10 şi 8 (alelele din ce nu au fost copiate încă în cea de-a doua progenitură). Rezultă

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 3 | 6 | 4 | 7 | 9 | 2 | 1 | 5 | 10 | 8 |

Funcţie MATLAB PMX(m) generează, pentru două permutări aleatoare de dimensiune *m*, progeniturile rezultate în urma aplicării operatorului PMX.

function [x2,y2]=PMX(m);

x1=gen\_perm(m);

y1=gen\_perm(m);

while(x1==y1)

y1=gen\_perm(m);

end;

disp(x1);

disp(y1);

p(1)=unidrnd(m-1);

p(2)=unidrnd(m);

while(p(2)==p(1))

p(2)=unidrnd(m);

end;

poz=sort(p);disp(poz);

x2=PMX1(x1,y1,poz(1),poz(2));

y2=PMX1(y1,x1,poz(1),poz(2));

disp(x2);

disp(y2);

end

function [x2]=PMX1(x1,y1,p1,p2);

[n,m]=size(x1);

x2=zeros(1,m);

x2(p1:p2)=x1(p1:p2);

disp(x2);

for p=p1:p2

a=y1(p);

if(~ismember(a,x2))

b=x2(p);

for t=1:m

if(y1(t)==b)

i=t;break;

end;

end;

if(x2(i)==0)

x2(i)=a;

else

j=cauta(x2,y1,i,m);

while(x2(j))

i=j;

j=cauta(x2,y1,i,m);

end;

x2(j)=a;

end;

end;

end;

for i=1:m

if(~ismember(y1(i),x2))

for j=1:m

if(x2(j)==0)

x2(j)=y1(i);break;

end;

end

end;

end;

end

function [j]=cauta(x2,y1,i,m);

c=x2(i);

for t=1:m

if(y1(t)==c)

j=t;break;

end;

end;

end

La apelul PMX(8), un rezultat posibil este:

Cromozomii initiali:

2 3 1 7 6 4 8 5

5 4 7 2 1 3 6 8

Pozitiile:

1 4

Urmasii:

2 3 1 7 5 4 6 8

5 4 7 2 6 3 8 1

**Observaţie.** În acest caz, pentru construcţia primului copil algoritmul este aplicat astfel:

,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 1 | 7 | 6 | 4 | 8 | 5 |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 5 | 4 | 7 | 2 | 1 | 3 | 6 | 8 |

Construcţia primului copil, , este realizată astfel.

După aplicarea primului pas, obţinem

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 1 | 7 | 0 | 0 | 0 | 0 |

La pasul 2 rezultă

La pasul 3:

Deoarece şi gena 4 din are valoarea , rezultă deci . Deoarece , şi rezultă . În continuare, , şi setează . Obţinem

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 1 | 7 | 5 | 0 | 0 | 0 |

Deoarece şi gena 6 din nu are încă valoare>0, setează . Obţinem

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 1 | 7 | 5 | 4 | 0 | 0 |

La pasul 4 vor fi plasate în , în ordinea genelor rămase fără valori, alelele 6 şi 8 (alelele din ce nu au fost copiate încă în prima progenitură). Rezultă

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 1 | 7 | 5 | 4 | 6 | 8 |

*Operatorul de recombinare a muchiilor* (Edge Crossover) este dezvoltat pe baza ideii că un cromozom copil trebuie creat utilizând exclusiv perechi de alele (secvenţe de două alele vecine – *muchii*) prezente cel puţin într-unul dintre părinţi. Cea mai utilizată versiune a acestui operator este cel propus în (Whitley, 2000) şi este proiectat astfel încât să asigure păstrarea tuturor muchiilor comune părinţilor în cromozomul urmaş. În acest scop este menţinută o listă a adiacenţelor (tabela de muchii) în care, pentru fiecare valoare *v*, sunt memorate toate alelele *a* vecine cu *v* într-unul dintre genotipurile părinte; dacă *a* şi *v* sunt vecine în ambii părinţi, atunci proprietatea este marcată adăugând după *a* simbolul ‘+’. Pentru o valoare dată, *v*, vom nota cu lista alelelor vecine cu *v* într-unul dintre părinţi şi cu numărul elementelor listei . Procedura de recombinare are ca rezultat crearea unui singur cromozom urmaş în modul următor.

Pas 1. Construieşte tabela de muchii.

Pas 2. Alege aleator o genă, *i*, şi copiază valoarea acesteia, *v*, în cromozomul progenitură. .

REPETĂ Pas 3 – Pas 5 PÂNĂ CÂND TOATE GENELE CROMOZOMULUI URMAŞ AU FOST COMPLETATE (AU PRIMTI CÂTE O VALOAREA)

Pas 3. Determină , lista vecinilor elementului curent .

Pas 4. Examinează elementele din încă necopiate în cromozomul urmaş. Dacă există o muchie comună, adică conţine cel puţin un element *a*+, atunci selectează primul *a* cu această proprietate pentru copiere în cromozomul rezultat. Altfel, alege pentru copiere acea valoarea cu proprietatea că . Setează . Valoarea copiată în cromozomul părinte nu va mai fi consultată ulterior: ea este eliminată din celelalte liste de adiacenţă.

Pas 5. Dacă , este examinat pentru expandare celălalt capăt al cromozomului urmaş, altfel este ales aleator un nou element (o nouă alelă).

De exemplu, fie şi

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 5 | 7 | 3 | 9 | 10 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 5 | 3 | 7 | 9 | 2 | 1 | 8 | 6 | 4 |

cromozomii selectaţi pentru încrucişare.

Tabela muchiilor este prezentată mai jos.

|  |  |  |  |
| --- | --- | --- | --- |
| Element | Lista muchiilor | Element | Lista muchiilor |
| 1 | 2+,8+ | 6 | 4+,5,8 |
| 2 | 1+,4,9 | 7 | 3+,5,9 |
| 3 | 5,7+,9 | 8 | 1+,6,10 |
| 4 | 2,6+,10 | 9 | 2,3,7,10 |
| 5 | 3,6,7,10 | 10 | 4,5,8,9 |

În următoarea figură este prezentată o modalitatea de obţinere a rezultatului recombinării cromozomilor şi (datorită alegerilor aleatoare cromozomul rezultat, nu este, desigur, unica variantă de urmaş).

|  |  |  |  |
| --- | --- | --- | --- |
| Alegeri posibile | ElC | Tipul selectării ElC | Cromozomul urmaş în construcţie, r |
| 1,2,…,10 | 1 | Aleatoriu | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | |
| 2,8 | 2 | Valoare comună | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | |
| 4,9 | 4 | Lista cea mai scurtă | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 9 | 9 | 9 | 9 | 9 | 9 | 9 | |
| 6,10 | 6 | Valoare comună | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 9 | 9 | 9 | 9 | 9 | 9 | |
| 5,8 | 8 | Lista cea mai scurtă | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 9 | 9 | 9 | 9 | 9 | |
| 10 | 10 | Singurul element | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 10 | 9 | 9 | 9 | 9 | |
| 5,9 | 5 | Aleator (listele cu vecinii încă necopiaţi au aceeaşi lungime) | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 10 | 5 | 9 | 9 | 9 | |
| 3,7 | 3 | Aleator (listele cu vecinii încă necopiaţi au aceeaşi lungime) | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 10 | 5 | 3 | 9 | 9 | |
| 7,9 | 7 | Valoare comună | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 10 | 5 | 3 | 7 | 9 | |
| 9 | 9 | Singurul element rămas neselectat | |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 4 | 6 | 8 | 10 | 5 | 3 | 7 | 9 | |

*Operatorul de recombinare de ordine* (Order Crossover) a fost propus în (Davis, 1991) pentru rezolvarea problemelor în care spaţiul genotipurilor este dat printr-un set de permutări semnificând ordinea apariţiei unor evenimente. Fie o pereche de cromozomi părinte, progeniturile calculate şi *m* numărul genelor unui cromozom. Procedura de recombinare revine la parcurgerea următorilor paşi.

Pas 1. Alege aleator puncte de încrucişare, efectuează

Pas 2. Copiază în valorile din care nu au fost deja copiate în la pasul 1, începând cu gena până la gena *m* şi apoi din poziţia 1 în poziţia

Cel de-al doilea copil este construit similar, plecând de la secvenţa de alele din cel de-al doilea părinte şi copiind restul elementelor din primul părinte similar procedeului descris pentru crearea primei progenituri.

De exemplu, fie ,

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 4 | 6 | 5 | 7 | 3 | 9 | 10 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 5 | 3 | 7 | 9 | 2 | 1 | 8 | 6 | 4 |

Construcţia primului copil, , este realizată astfel.

După aplicarea primului pas, obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 6 | 5 | 7 | 3 |  |  |  |

După efectuarea pasului 2 rezultă

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 10 | 9 | 2 | 6 | 5 | 7 | 3 | 1 | 8 | 4 |

Construcţia celui de-al doilea urmaş, , este realizată astfel.

După aplicarea primului pas, obţinem

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | 7 | 9 | 2 | 1 |  |  |  |

În final rezultă

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 4 | 6 | 5 | 7 | 9 | 2 | 1 | 3 | 10 | 8 |

*Operatorul recombinare ciclică* (CX, Cycle Crossover) este dezvoltat pe baza ideii că un cromozom copil trebuie creat păstrând cât mai bine posibil informaţia relativă la poziţia absolută în care apare o valoare în cromozomii părinte. Procedura de generare a progeniturilor grupează elementele în cicluri. Un ciclu este o secvenţă de alele cu proprietatea că fiecare element apare în pereche cu un alt element al aceluiaşi ciclu în perechea de cromozomi părinte. După partajarea părinţilor în cicluri, cromozomii rezultaţi sunt construiţi prin alegerea alternativă a câte unui ciclu din fiecare părinte. Fie o pereche de cromozomi părinte, progeniturile calculate, *m* numărul genelor unui cromozom şi *nrcicluri* numărul ciclurilor calculate. Pentru fiecare (alegerile impare), elementele ciclurilor din (pereche ordonată) se regăsesc în aceleaşi poziţii în (pereche ordonată), în timp ce, pentru fiecare (alegerile pare), elementele ciclurilor din (pereche ordonată) se regăsesc în aceleaşi poziţii în (pereche ordonată).

O variantă de implementare a acestui operator a fost prezentată în (Oliver, Smith, Holland, 1987) pentru rezolvarea problemei comis-voiajorului, modalitatea de partiţionare în cicluri fiind descrisă astfel.

Cât timp nu au fost asignate toate alelele unui ciclu, execută

Pas 1. Iniţializează ciclul curent cu secvenţa vidă,

Pas 2. Fie *i* prima genă din încă neutilizată;

Pas 3. Calculează

Pas 4. Determină

Pas 5.

Pas 6. Repetă Pas 3 - Pas 5 până când .

De exemplu, fie , o pereche de cromozomi părinte prezentaţi mai jos. Ciclurile identificate sunt:

Grafic, procedura de identificare a ciclurilor poate fi figurată astfel.





În continuare este prezentată implementarea acestui operator. Partiţionarea în cicluri este realizată prin atribuirea indicelui ciclului din care face parte fiecărei alele din fiecare cromozom, conform algoritmului descris mai sus.

function [x2,y2]=CX(m);

x1=gen\_perm(m);

y1=gen\_perm(m);

while(x1==y1)

y1=gen\_perm(m);

end;

%x1=[1 2 4 6 5 7 3 9 10 8];

%y1=[10 5 2 7 9 1 3 8 6 4];

disp('Parintii:');

disp(x1);

disp(y1);

[c,nrcicluri]=cicluri(x1,y1,m);

disp('Indecsii ciclurilor:');

disp(c);

x2=x1;y2=y1;

for t=2:2:nrcicluri

for i=1:m

if(c(i)==t)

x2(i)=y1(i);

y2(i)=x1(i);

end;

end;

end;

disp('Progeniturile:');

disp(x2);

disp(y2);

end

function [c1,index]=cicluri(x1,y1,m);

i=1;index=1;continua=1;

c1=zeros(1,m);

while(continua)

a=y1(i);

c1(i)=index;

while(x1(i)~=a)

for k=1:m

if(x1(k)==a)

j=k;break;

end;

end;

c1(j)=index;

a=y1(j);

end;

continua=0;

for k=1:m

if(c1(k)==0)

i=k;index=index+1;

continua=1;

break;

end;

end;

end;

end

Prezentăm în continuare câteva rezultate posibile obţinute la apelul CX(10), CX(12) şi CX(14).

Parintii:

6 1 8 10 5 7 9 3 4 2

9 8 7 3 6 1 5 10 4 2

Indecsii ciclurilor:

1 2 2 3 1 2 1 3 4 5

Progeniturile:

6 8 7 10 5 1 9 3 4 2

9 1 8 3 6 7 5 10 4 2

Parintii:

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

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

Indecsii ciclurilor:

1 2 1 2 1 1 3 1 1 3 2 2

Progeniturile:

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

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

Parintii:

13 4 10 7 14 6 8 3 2 1 11 5 9 12

3 1 11 8 4 6 13 9 5 10 14 12 7 2

Indecsii ciclurilor:

1 2 2 1 2 3 1 1 4 2 2 4 1 4

Progeniturile:

13 1 11 7 4 6 8 3 5 10 14 12 9 2

3 4 10 8 14 6 13 9 2 1 11 5 7 12

**Recombinarea multiplă**

Operatorii de recombinare multiplă presupun obţinerea de progenituri prin utilizarea n-tuplurilor de părinţi, . Astfel de operatori sunt uşor de implementat, dar, spre deosebire de recombinarea a doi cromozomi părinte, nu au echivalent biologic. Din punct de vedere tehnic utilizarea recombinării multiple poate amplifica efectele încrucişării. Operatorii de recombinare multiplă sunt clasificaţi în funcţie de mecanismul care stă la baza combinării informaţiei cromozomilor părinte, astfel:

* bazaţi pe frecvenţa alelelor: de exemplu votul *p*-sexual, care generalizează încrucişarea uniformă;
* bazaţi pe segmentarea şi recombinarea secvenţelor: încrucişarea diagonală, care generalizează recombinarea multi-punct;
* bazaţi pe operaţii asupra alelelor numere reale în virgulă fixă: recombinarea de tip baricentru, care generalizează operatorii de recombinare artitmetică.

În general creşterea arităţii operatorului de încrucişare de la 2 la *n* nu garantează obţinerea unor EA superiori ca performanţă, dar, în multe situaţii raportate în literatura de specialitate (Eiben, Schippers, 1993) utilizarea mai multor părinţi pentru recombinare s-a dovedit avantajoasă.

* 1. **Modele de populaţie**

În literatura de specialitate au fost evidenţiate două modele de GA: modelul generaţional şi modelul stărilor stabile. Exemplul prezentat la începutul capitolului implementează un model generaţional.

La fiecare moment de timp, adică la fiecare generaţie, populaţia este de dimensiune , iar bazinul de încrucişare (format din mulţimea cromozomilor aleşi pentru a se reproduce) este, de asemenea, de dimensiune . În continuare sunt create progenituri prin aplicarea operatorilor de variaţie. Populaţia curentă este înlocuită în întregime cu progeniturile rezultate (generaţia următoare), fiecare copil fiind evaluat prin intermediul funcţiei de fitness asociate.

În cadrul modelelor cu stări stabile, doar o parte din progenituri înlocuiesc o parte din membrii populaţiei curente pentru a forma generaţia următoare. În acest caz indivizi din populaţia curentă sunt înlocuiţi cu indivizi noi. Procentul modificărilor din populaţie este şi se numeşte decalajul dintre generaţii. De exemplu, unul din modelele intens studiate este modelul GENITOR propus în (Whitley, 2000), unde decalajul dintre generaţii este .

* 1. **Mecanismul de selectare a părinţilor**

**Selecţia proporţională cu calitatea indivizilor (valorile funcţiei de evaluare), FPS**

Principiul acestui mecanism de selectare a părinţilor (numit FPS: **F**itness **P**roportional **S**election) este următorul. Pentru fiecare cromozom , este calculată performanţa sa, prin funcţia de evaluare . Probabilitatea de selectare a individului pentru încrucişare este

Cu alte cuvinte, probabilitatea de selectare a individului depinde de valoarea absolută a calităţii individuale comparativ cu valoarea absolută a calităţii populaţiei curente. Numărul părinţilor este egal cu . Pentru fiecare individ , numărul de apariţii în multisetul părinţilor este, în medie,

unde este valoarea medie a funcţiei obiectiv calculată pentru populaţia curentă. Evident, aceste valori nu sunt numere din mulţimea ; ele reprezintă realizări ale unei variabile aleatoare cu o anumită distribuţie de probabilitate.

Mecanismul de selecţie FPS (Holland, 1992) are o serie de inconveniente, din care menţionăm:

* indivizii mult superiori restului populaţiei (din punct de vedere al valorii funcţiei de evaluare) vor domina întreaga populaţie într-un timp relativ scurt, proprietate cunoscută drept *convergenţă prematură*;
* dacă indivizii din populaţie sunt astfel încât 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, în consecinţă, 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;
* mecanismul se comportă diferit pe versiuni translatate, respectiv transpuse, ale unei aceeaşi funcţii de evaluare.

Ultimele două inconveniente pot fi eliminate prin utilizarea unei proceduri de împărţire în ferestre. Aceasta presupune ca, la fiecare moment de timp *t*, pentru fiecare cromozom , al populaţiei curente corespunzător fenotipului , scăderea valorii funcţiei de evaluare cu o valoare , care depinde într-un anume mod de istoria recentă a evoluţiei GA. O alegere frecventă este cea în care semnifică media mobilă a calităţii celui mai slab individ pe parcursul ultimelor *p* generaţii

O altă variantă foarte utilizată este sigma-scalarea (Goldberg, 1989), care încorporează informaţiile medie de selecţie şi respectiv deviaţie standard de selecţie calculate pentru exemplele populaţiei curente ,,

unde c este un parametru dat (de obicei ).

**Selecţia rangurilor**

Selecţia pe baza funcţiei rang este o metodă alternativă inspirată de FPS. Procedura menţine o valoare constantă a constrângerii de selecţie prin sortarea populaţiei în funcţie de valorile funcţiei fitness şi alocarea câte unei probabilităţi de selecţie fiecărui individ, în funcţie de rangul ocupat în şirul sortat şi nu de valoarea curentă a funcţiei de evaluare a acelui individ. Presupunem în continuare că este dimensiunea populaţiei curente sortate, primul individ (cel cu rang 1) este cel mai slab şi individul cu rang este cel mai bun.

Formele uzuale de calcul al probabilităţii de selecţie a unui individ sunt următoarele.

Fie este un parametru dat. Pentru fiecare cromozom , al populaţiei curente sortate , probabilitatea de selecţie este setată liniar,

Această variantă de definire a probabilităţii de selecţie are la bază presupunerea că, în medie, un individ cu calitatea apropiată de media valorilor funcţiei de evaluare, deci aflat la jumătatea şirului ordonat, trebuie să aibă, relativ la întreaga populaţie, o şansă de a se reproduce: pentru

**Observaţie.** Pentru a nu fi necesară scalarea în relaţia de definire a distribuţiei de probabilitate de selecţie de tip rang cu setare liniară, poate fi utilizată definiţia

Într-adevăr,

O altă variantă de stabilire a probabilităţii de selecţie corespunde situaţiei în care constrângerea de selecţie trebuie să fie mai mare, în sensul că probabilitatea de selecţie a indivizilor cu calitatea peste medie trebuie să fie semnificativ mai mare decât a celorlalţi. În acest caz este utilizată o distribuţie de tip exponenţial:

unde este un parametru care asigură faptul că

**Mecanisme de implementare a distribuţiei de probabilitate a selecţiei**

Următoarele mecanisme de proiectare a distribuţiei de probabilitate pentru selectarea părinţilor au la bază ideea că, în varianta ideală, bazinul de recombinare format din cromozomii care interschimbă material genetic corespunde exact distribuţiei de probabilitate de selecţie. În practică acest deziderat nu poate fi atins, deoarece populaţia are un număr finit de indivizi. Bazinul de recombinare poate fi ales astfel încât să corespundă unei probe din distribuţia de probabilitate a selecţiei părinţilor, dar nu o poate reflecta exact.

*Mecanismul de selecţie a părinţilor de tip ruletă* este o variantă de construcţie *a bazinului de recombinare astfel încât să corespundă unei probe din distribuţia de probabilitate a selecţiei*. Distribuţia de probabilitate poate fi în general de tip FPS sau bazată pe ranguri. O variantă de implementare a fost prezentată în §3.1 pentru distribuţia de selecţie de tip fitness (FPS) şi corespunde următorului algoritm.

Pas 1. Pentru fiecare cromozom :

* 1. evaluează performanţa sa, prin funcţia de evaluare ;
  2. calculează probabilitatea de selecţie din mecanismul de tip FPS sau cea bazată pe ranguri, şi probabilitatea cumulată

Pas 2. Pentru execută 2.1 şi 2.2

2.1. generează aleator un număr ;

2.2 dacă parinte(k)=;

altfel, este selectat cromozomul , , cu proprietatea :

parinte(k)=

O variantă mai bună pentru construcţia unui bazin de încrucişare cât mai apropiat de o instanţă a distribuţiei de probabilitate a selecţiei este *algoritmul SUS* (**S**tochastic **U**niversal **S**ampling). Spre deosebire de mecanismul de tip ruletă, în care o ruletă cu un braţ este rotită de ori, mecanismul SUS asigură o rotire a unei rulete cu braţe echidistante.

Algoritmul este următorul:

Pas 1. Pentru fiecare cromozom :

* 1. evaluează performanţa sa, prin funcţia de evaluare ;
  2. calculează probabilitatea de selecţie din mecanismul de tip FPS sau cea bazată pe ranguri, şi probabilitatea cumulată

Pas 2. generează aleator un număr ; i=1; k=1;

Pas 3. Cât timp execută 3.1 şi 3.2

3.1. cât timp execută 3.1.1, 3.1.2 şi 3.1.3

3.1.1. parinte(k)=;

3.1.2. ;

3.1.3. ;

3.2.

Deoarece r este generat aleator în intervalul şi, după fiecare selecţie efectuată este mărit cu , numărul de copii ale fiecărui cromozom în bazinul de recombinare este cel puţin şi cel mult , unde este partea întreagă a numărului real *x*.

În următoarea funcţie MATLAB este implementat algoritmul SUS pentru selecţia părinţilor în cadrul problemei prezentate în 3.1, cu probabilitatea de selecţie corespunzătoare mecanismului FPS standard.

function [parinti,p,q]=selectie\_SUS(pop);

[dim,m]=size(pop);

p=zeros(dim,1);

p(1:dim)=pop(1:dim,6);

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=zeros(dim,m);

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=q(i))

parinti(k,1:m)=pop(i,1:m);

r=r+1/dim;

k=k+1;

end;

i=i+1;

end;

end

Păstrând celelalte funcţii neschimbate, la o iteraţie a algoritmului GA cu implementarea selecţiei SUS poate fi obţinut rezultatul:

Populaţia iniţială

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Indice cromozom | Cromozom | Valoarea funcţiei obiectiv | Probabilitatea de selecţie FPS | Probabilitatea cumulată |
| 1 | 1 1 0 0 1 | 625 | 0.3531 | 0.3531 |
| 2 | 1 0 0 1 1 | 361 | 0.2040 | 0.5571 |
| 3 | 0 0 0 0 0 | 0 | 0 | 0.5571 |
| 4 | 1 1 1 0 0 | 784 | 0.4429 | 1.0000 |

Populaţia selectată pentru recombinare şi populaţia rezultată

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bazinul de împerechere selectat şi modul de împerechere | | | | Indivizii rezultaţi (copiii) | |
| Indice cromozom | Cromozom | Valoarea funcţiei obiectiv | Împerecherea | Cromozom | Valoarea funcţiei obiectiv |
| 1 | 1 1 0 0 1 | 625 | Cromozomii 4 şi 2, punctul de încrucişare 2 | 1 1 0 1 1 | 729 |
| 2 | 1 0 0 1 1 | 361 | 1 0 1 0 0 | 400 |
| 3 | 1 1 1 0 0 | 784 | Cromozomii 3 şi 1, punctul de încrucişare 3 | 1 1 1 0 1 | 841 |
| 4 | 1 1 1 0 0 | 784 | 1 1 0 0 0 | 576 |

Nu este efectuată nici o operaţiei de mutaţie.

**Selecţia de tip turneu**

În situaţiile în care dimensiunea populaţiei este foarte mare sau populaţia este distribuită în sisteme paralele, calculul probabilităţilor de selecţie de tip FPS poate dura foarte mult sau poate fi imposibil. Selecţia de tip turneu este un operator cu proprietatea importantă că nu necesită informaţii globale relativ la populaţia curentă şi presupune definirea unei relaţii de ordine pe doi (sau în general k, unde ) indivizi. Aplicarea operatorului de selecţie de tip turneu pentru selectarea a părinţi revine la implementarea următorului algoritm.

Pas 1. Pentru execută 1.1, 1.2 şi 1.3

1.1. generează aleator *k* indivizi din populaţie în setul *S* (cu sau fără înlocuire);

1.2. calculează , cel mai bun individ din *S*;

1.3. parinte(i)=;

Un astfel de operator de selecţie a fost implementat în §2.4 pentru rezolvarea problemei rucsacului de tip 0-1, unde şi generarea este fără înlocuire.

Probabilitatea de selecţie a unui individ în sistemul turneu depinde de următorii factori:

* rangul (poziţia ocupată în cadrul populaţiei);
* dimensiunea turneului, *k*; cu cât *k* este mai mare, cu atât este mai mare probabilitatea de a include în turneu indivizi cu calitate superioară mediei;
* *p*, probabilitatea de a selecta cel mai bun individ din turneu; în cele mai multe situaţii şi conform algoritmului prezentat, , adică turneul este determinist, dar pot fi folosite şi variante stochastice, în care şi, evident, constrângerea de selecţie este scăzută;
* modul de alegere a indivizilor pentru turneu; dacă alegerea se face fără înlocuire, în cazul turneului determinist, cei mai slabi *k*-1 indivizi din populaţie nu vor fi niciodată selectaţi, indiferent de faptul că generare este aleatoare; în caz contrar, poate fi selectat şi cel mai slab individ (evident, cu o probabilitate extrem de mică).

Deoarece, pentru selectarea a indivizi este necesară organizarea a turnee, acest operator de selecţie are acelaşi inconvenient ca şi selecţia de tip ruletă: indivizii selectaţi nu respectă distribuţia de probabilitate teoretică. În ciuda acestui inconvenient, operatorul de selecţie de tip turneu este unul dintre cei mai utilizaţi în GA, datorită simplităţii implementării şi a faptului că este uşor de controlat constrângerea selecţiei prin modificarea dimensiunii turneului, *k*.

* 1. **Selecţia generaţiei următoare**

Mecanismul de selectare a generaţiei următoare (numit şi schimbul de generaţii) este responsabil de modalitatea de a selecta, din indivizi ai populaţiei curente şi progenituri (eventual mutante) indivizi care să formeze generaţia următoare. Aşa cum am menţionat şi în capitolul 2, există două clase de mecanisme care să asigure schimbul de generaţii.

**Schimbul de generaţii bazat pe vârstă**

Ideea care stă la baza schimbului de generaţii efectuat în funcţie de factorul vârstă este de a asigura prezenţa fiecărui individ într-un număr fixat de generaţii şi nu în funcţie de calitatea indivizilor. Un mecanism de acest tip este utilizat de obicei în cazul GA simpli.

Deoarece, în general, numărul urmaşilor, , este egal cu dimensiunea populaţiei, , strategia schimbă populaţia curentă cu multisetul de progenituri (eventual mutante) create, indiferent dacă populaţia curentă şi/sau multisetul progeniturilor conţin indivizi care se repetă (sunt mutiseturi de cromozomi, nu mulţimi în sens matematic).

La polul opus este strategia care constă în crearea unui singur cromozom copil la fiecare ciclu şi înlocuirea celui mai „bătrân” individ al populaţiei curente cu noul cromozom creat. O astfel de strategie este de corespunde unei organizări FIFO (First-In-First-Out) a cromozomilor. O alternativă la această strategie utilizată în modelele cu stări stabile revine la selectarea aleatoare a unui individ din populaţie şi înlocuirea lui cu cromozomul copil nou creat. Aplicarea acestei strategii conduce la obţinerea de GA cu performanţe care pot varia foarte mult comparativ cu GA generaţionali, motivul principal fiind acela că probabilitatea ca cel mai bun individ din populaţie să fie eliminat la o alegere aleatoare este mult superioară situaţiei strategiei FIFO de eliminare a celui mai vechi individ (De Jong, Sarma, 1992; Smith, Vavac,1999).

**Schimbul de generaţii bazat pe calitatea indivizilor**

Schimbul de generaţii dirijat de valorile funcţiei de evaluare poate fi realizat prin strategii utilizate şi în selectarea părinţilor, cele mai utilizate fiind FPS, selecţia de tip turneu şi varianta stochastică a selecţiei bazate pe ranguri. În plus faţă de acestea, sunt uzual folosite mecanismul GENITOR şi elitismul.

*Schema de substituire GENITOR* presupune înlocuirea celor mai slabi indivizi din populaţia curentă. Acest mecanism conduce la îmbunătăţirea semnificativă a calităţii globale a populaţiei (măsurată ca media calităţii indivizilor care compun populaţia), dar are dezavantajul că poate conduce la convergenţă prematură. Din acest motiv modelul este utilizat pentru populaţii de dimensiuni mari sau în cazul populaţiilor care nu conţin duplicate.

*Elitismul* este o strategie utilizată în combinaţie cu scheme de substituire bazate pe vârstă şi scheme stochastice de înlocuire bazate pe funcţia de evaluare, scopul fiind acela de a evita pierderea celor mai bine adaptaţi indivizi la schimbul de generaţii. Aceasta presupune urmărirea celui mai bun individ din populaţia curentă, *b*: dacă *b* este ales pentru înlocuire şi nici unul dintre urmaşii care sunt selectaţi pentru schimbul de generaţii nu are valoarea funcţiei obiectiv cel puţin egală cu cea corespunzătoare lui *b*, atunci *b* este menţinut în generaţia următoare şi este eliminat unul dintre urmaşii selectaţi pentru înlocuire.

În următoarea funcţie MATLAB este implementată o schemă de tip elitist pentru selecţia generaţiei următoare în cadrul problemei prezentate în 3.1.

function [generatieN]=schimba\_generatii(pop,popN);

% in ultimul argument, din fiecare linie este situat meritul individului

% corespunzator liniei

generatieN=popN;

[dim,n]=size(pop);

[max1,i]=max(pop(:,n));

[max2,j]=max(popN(:,n));

if(max1>max2)

[min1,k]=min(popN(:,n));

generatieN(k,:)=pop(i,:);

end;

end

În următorul exemplu, populaţia generată la momentul iniţial conţine individul cel mai bun,

1 1 1 1 1

cu valoarea funcţiei obiectiv 961 (marcat cu roşu). În urma operaţiilor de încrucişare şi mutaţie, populaţia rezultată are drept cel mai bun individ

1 1 1 0 0

cu valoarea funcţiei obiectiv 784 şi un cel mai slab individ

0 1 1 1 1

cu valoarea funcţiei obiectiv 225 (marcat cu albastru). Selectarea generaţiei următoare (de la momentul de timp 1) consideră populaţia rezultată în urma aplicării operatorilor de variaţie şi înlocuieşte cel mai slab individ din aceasta cu individul cel mai bun al populaţiei de la momentul 0 (acesta nefiind membru în multisetul urmaşilor). Rezultă o evoluţie de tipul următor.

Populatia initiala si valorile functiei obiectiv

0 0 1 1 1 49

1 0 0 0 0 256

0 1 1 0 0 144

0 1 1 1 0 196

0 1 0 0 0 64

1 1 1 1 1 961

Bazinul de imperechere si valorile functiei obiectiv

1 0 0 0 0 256

0 1 1 0 0 144

0 1 0 0 0 64

1 1 1 1 1 961

1 1 1 1 1 961

1 1 1 1 1 961

Operatia de incrucisare

Incrucisare intre indivizii din pozitiile

2

5

Punctul de incrucisare

3

Incrucisare intre indivizii din pozitiile

3

6

Punctul de incrucisare

2

Incrucisare intre indivizii din pozitiile

1

4

Punctul de incrucisare

3

Copii rezultati

0 1 1 1 1 225

1 1 1 0 0 784

0 1 1 1 1 225

1 1 0 0 0 576

1 0 0 1 1 361

1 1 1 0 0 784

Operatia de mutatie -

Populatia rezultata

0 1 1 1 1 225

1 1 1 0 0 784

0 1 1 1 1 225

1 1 0 0 0 576

1 0 0 1 1 361

1 1 1 0 0 784

Generatia urmatoare

1 1 1 1 1 961

1 1 1 0 0 784

0 1 1 1 1 225

1 1 0 0 0 576

1 0 0 1 1 361

1 1 1 0 0 784

* 1. **Aplicaţii**

În continuare sunt prezentate câteva probleme rezolvate cu ajutorul GA, pentru diferite alegeri ale tipului de populaţie, operatorilor de variaţie şi parametrilor asociaţi acestora, mecanisme de selectare a părinţilor şi respectiv metode de schimbare a generaţiilor.

**3.8.1. Rezolvarea problemei One-Max**

Problema One-Max este printre primele prezentate în literatura de specialitate, datorită pe de o parte simplităţii ei şi, pe de altă parte, multiplelor modalităţi de rezolvare permise, care oferă posibilitatea efectuării de comparaţii între diversele opţiuni de rezolvare (Eiben, Smith, 2003).

Funcţia corespunzătoare problemei One-Max este definită pe mulţimea şirurilor binare de lungime *L* prin:

Problema este de a maximiza funcţia *f*. Evident optimul global este *L* şi este atins pentru .

În continuare sunt prezentate două implementări GA, pentru . În ambele implementări sunt considerate următoarele setări:

* spaţiul genotipurilor este identic cu spaţiul fenotipurilor: ;
* populaţia iniţială este generată aleator, dimensiunea populaţiei este 100;
* operatorul de recombinare: încrucişare uni-punct, cu probabilitatea ;
* operatorul mutaţie: inversarea valorii biţilor, cu probabilitatea ;
* schimbul de generaţii: strict generaţional, pe baza vârstei, cu înlocuirea completă a populaţiei curente cu multisetul progeniturilor;
* condiţia de oprire: maxim 100 de iteraţii sau atingerea valorii maxime, *L*.

În prima variantă de implementare, selecţia părinţilor este realizată prin algoritmul SUS cu probabilitatea de selecţie din modelul FPS standard, în cea de-a doua prin metoda turneu cu dimensiune 2. La fiecare generaţie se reprezintă într-un grafic cu axa timpului următorele valori: calitatea celui mai slab membru al populaţiei, calitate celui mai bun membru al populaţiei şi calitatea medie.

Următoarele funcţii MATLAB implementează prima metodă:

function []=GA\_SUS(dim,L,pc,pm,MNrIt);

[pop]=genereaza\_ini(L,dim);

maxc=0;i=0;

Mmax=[];

Mmed=[];

Mmin=[];

while((maxc<L)&&(i<MNrIt))

%selectia paritilor

parinti=selectie\_SUS(pop);

%aplicarea recombinarii

popN1=crossover(parinti,pc);

%aplicarea mutatiei

popN=mutatie(popN1,pm);

%calculul meritului fiecarui individ

fob=zeros(dim,1);

for k=1:dim

fob(k)=f\_one\_max(popN(k,:));

end;

% meritul mediu

medie=sum(fob)/dim;

Mmed=[Mmed medie];

%meritul minim

minim=min(fob);

Mmin=[Mmin minim];

%meritul maxim

maxim=max(fob);

Mmax=[Mmax maxim];

maxc=maxim;

i=i+1;

pop=popN;

end;

[val,indice]=max(Mmax);

figure

x=1:i-1;

plot(x,Mmin(x),'m','LineWidth',1);

hold on;

plot(x,Mmed(x),'b','LineWidth',2);

hold on;

plot(x,Mmax(x),'r','LineWidth',4);

hold on;

plot(indice,val,'rs','MarkerEdgeColor','k','MarkerFaceColor','y',

'MarkerSize',12);

axis([0 MNrIt 0 L]);

end

function [pop]=genereaza\_ini(L,dim);

pop=zeros(dim,L);

z=ones(1,L);

for i=1:dim

x=unidrnd(2,1,L)-z;

pop(i,1:L)=x(1:L);

end;

end

function [val]=f\_one\_max(x);

val=sum(x);

end

function [parinti]=selectie\_SUS(pop);

[dim,L]=size(pop);

fob=zeros(dim,1);

for i=1:dim

fob(i)=f\_one\_max(pop(i,:));

end;

p=fob;

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=zeros(dim,L);

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=q(i))

parinti(k,1:L)=pop(i,1:L);

r=r+1/dim;

k=k+1;

end;

i=i+1;

end;

end

function [popN]=crossover(pop,pc);

[dim,L]=size(pop);

popN=zeros(dim,L);

V=[];

for k=1:dim/2

% determinarea a doi indivizi distincti si care nu au fost selectati

% anetrior

poz=zeros(2,1);

for i=1:2

poz(i)=unidrnd(dim);

while(ismember(poz(i),V))

poz(i)=unidrnd(dim);

end;

V=[V poz(i)];

end;

% incrucisarea celor doi indivizi; rezulta y1,y2

x1=pop(poz(1),1:L);y1=x1;

x2=pop(poz(2),1:L);y2=x2;

r=unifrnd(0,1);

if(r<=pc)

% genereaza pozitia din care se face interschimbul

pozitie=unidrnd(L-1);

y1(pozitie+1:L)=x2(pozitie+1:L);

y2(pozitie+1:L)=x1(pozitie+1:L);

end; %altfel sunt copiati parintii:rezultat al incrucisarii asexuate

popN(k,1:L)=y1;

popN(k+1,1:L)=y2;

end;

end

function [popN]=mutatie(pop,pm);

[dim,L]=size(pop);

popN=pop;

for i=1:dim

for k=1:L

r=unifrnd(0,1);

if(r<pm)

popN(i,k)=not(popN(i,k));

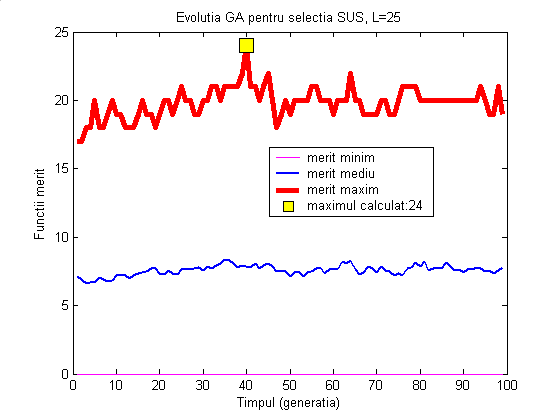
end;

end;

end;

end

O posibilă evoluţie a algoritmului la apelul *GA\_SUS(100,25,0.8,1/25,100)* este prezentată în următoarea figură. În general, după mai multe apeluri ale acestei funcţii maximul obţinut este între valorile 20 şi 24 (maximul este ), undeva pe parcursul evoluţiei generaţiilor. În figura următoare este prezentată prima poziţie în care este obţinută valoarea maximă.



În cea de-a doua variantă este modificată doar funcţia de selecţie, astfel.

function [parinti]=selectie\_turneu(pop);

[dim,L]=size(pop);

parinti=zeros(dim,L);

i=0;

for i=1:dim

p1=unidrnd(dim);

p2=unidrnd(dim);

while(p1==p2)

p2=unidrnd(dim);

end;

if(f\_one\_max(pop(p1,:))>=f\_one\_max(pop(p2,:)))

parinti(i,:)=pop(p1,:);

else

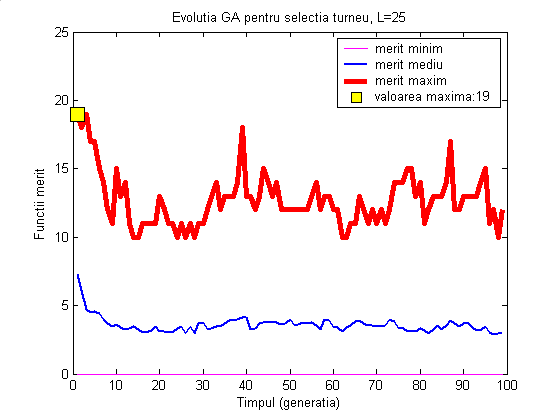
parinti(i,:)=pop(p2,:);

end;

end;

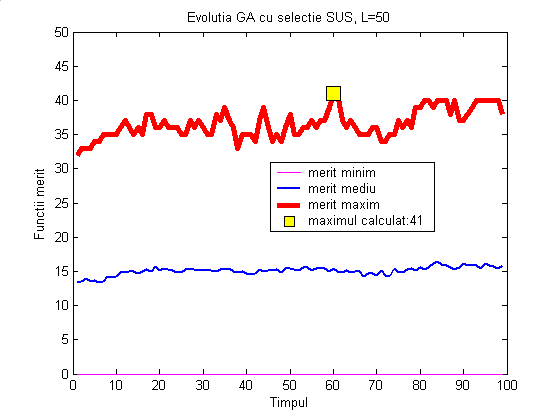
end

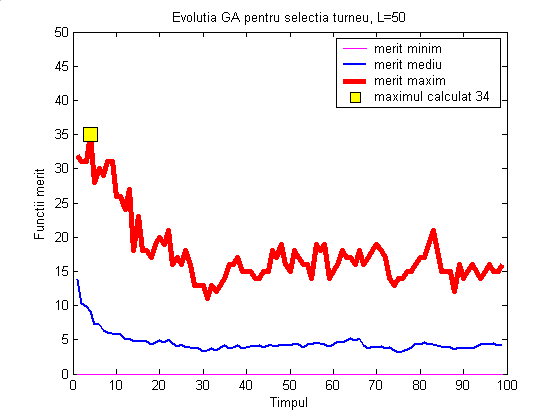
O posibilă evoluţie a algoritmului la apelul *GA\_turneu(100,25,0.8,1/25,100)* este prezentată în următoarea figură. În general, după mai multe apeluri ale acestei funcţii maximul obţinut este în general între 19 şi 21 (maximul este ), către începutul evoluţiei. Selecţia de tip turneu este mai puţin potrivită în acest caz. În figura următoare este prezentată prima poziţie în care este obţinută valoarea maximă.



Pentru (maximul teoretic este ) şi aceiaşi parametri, în cazul selecţiei SUS maximul calculat este situat în general între 40 şi 43, în timp ce în cazul selecţiei turneu, maximul calculat a fost obţinut în general între valorile 32 şi 41, tot la începutul evoluţiei. În acest caz diferenţa de evoluţie este şi mai pregnantă, aşa cum rezultă din următoarele figuri.

Evident, rezultate mai bune pot fi obţinute dacă dimensiunea populaţiei şi/sau numărul de iteraţii au valori mai mari. De exemplu, la apelul *GA\_SUS(1000,50,0.8,1/50,500)* maximul obţinut este în general între valorile 45 şi 47.





**Observaţie**. Dacă schimbul de generaţii este de tip „super elitist”, adică din populaţia curentă cu *dim* indivizi şi multisetul urmaşilor direcţi, format tot din *dim* indivizi, sunt aleşi cei mai buni *dim* indivizi (indiferent de factorul vârstă), este obţinut maximul teoretic în câteva zeci de iteraţii, în ambele variante de alegere a operatorului de selecţie a părinţilor (turneu sau SUS).

Codul funcţiei MATLAB este următorul:

function []=GA\_var(dim,L,pc,pm,MNrIt);

[pop]=genereaza\_ini(L,dim);

maxc=0;i=0;

Mmax=[];

Mmed=[];

Mmin=[];

while((maxc<L)&&(i<MNrIt))

%selectia paritilor

parinti=selectie\_turneu(pop);

%aplicarea recombinarii

popN1=crossover(parinti,pc);

%aplicarea mutatiei

popN=mutatie(popN1,pm);

%calculul meritului fiecarui individ

popNN=[pop;popN];

fob1=zeros(2\*dim,1);

for k=1:2\*dim

fob1(k)=f\_one\_max(popNN(k,:));

end;

popc=zeros(2\*dim,L+1);

popc(1:2\*dim,1:L)=popNN(1:2\*dim,1:L);

popc(1:2\*dim,L+1)=fob1(1:2\*dim);

tt=sortrows(popc,L+1);

popNou=zeros(dim,L);

popNou=tt(dim+1:2\*dim,1:L);

fob=zeros(dim,1);

for k=1:dim

fob(k)=f\_one\_max(popNou(k,:));

end;

% meritul mediu

medie=sum(fob)/dim;

Mmed=[Mmed medie];

%meritul minim

minim=min(fob);

Mmin=[Mmin minim];

%meritul maxim

maxim=max(fob);

Mmax=[Mmax maxim];

maxc=maxim;

i=i+1;

pop=popNou;

end;

[val,indice]=max(Mmax);

disp(val);

disp(indice);

figure

x=1:i-1;

plot(x,Mmin(x),'k','LineWidth',1);

hold on;

plot(x,Mmed(x),'b','LineWidth',2);

hold on;

plot(x,Mmax(x),'r','LineWidth',4);

hold on;

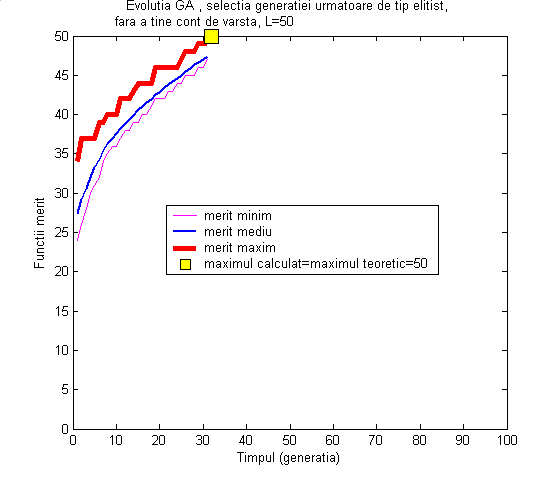
plot(indice,val,'rs','MarkerEdgeColor','k','MarkerFaceColor','y','Marker

Size',12);

axis([0 MNrIt 0 L]);

end

O evoluţie posibilă este prezentată mai jos. Valoarea maximă este obţinută în 32 de generaţii.



* + 1. **Rezolvarea problemei comis-voiajorului (TSP)**

Fie *n* oraşe interconectate două câte două şi D matricea costurilor de deplasare între orașe: 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.

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

În scopul obţinerii unei probleme de maxim, funcţia obiectiv definită pe spaţiul genotipurilor este

Evident, funcţia este bine definită, deoarece nici un cost nu poate fi nul.

În ambele implementări sunt considerate următoarele setări:

* spaţiul genotipurilor: permutări
* populaţia iniţială este generată aleator, dimensiunea populaţiei este *dim* (*dim* este stabilită în funcţie de numărul de oraşe);
* operatorul de recombinare: CX, cu probabilitatea ;
* operatorul mutaţie: inversiunea unui subşir aleator într-un cromozom, cu probabilitatea ;
* selecţia părinţilor:
  + V1. pe baza algoritmului SUS, cu probabilitate de selecţie din modelul FPS standard;
  + V2. pe baza algoritmului SUS, cu probabilitate de selecţie de tip rang (cu presiune de selecţie medie 1.5)
  + V3. selecţia de tip turneu
* condiţia de oprire: a fost depăşit un număr maxim de iteraţii;
* schimbul de generaţii a fost realizat în două variante:
  + V1. strict generaţional, pe baza vârstei, cu înlocuirea completă a populaţiei curente cu multisetul progeniturilor;
  + V2. 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.

În continuare este prezentată varianta în care operatorul de selecţie a părinţilor este de tip **SUS, cu probabilitate de selecţie din modelul FPS standard** şi schimbul de generaţii este realizat conform variantei V2. Implementarea funcţiei de recombinare CX a fost prezentată în §3.4, singura diferenţă fiind aceea că genotipurile care interschimbă material genetic sunt parametri de intrare (nu sunt generaţi aleator în interiorul funcţiei).

function []=GA\_TSP(nume,dim,pc,pm,NMax);

[d]=citeste\_date(nume);

[n,n]=size(d);

pop=gen\_ini(dim,n);

Maxx=[];

perm=[];

for i=1:NMax

%selectia parintilor de tip SUS

[parinti,castiguri,castiguri1]=selectie\_SUS(pop,d);

[popN1,castiguriN1]=crossover(parinti,castiguri,pc,d);

[popN,castiguriN]=mutatie\_inversiune(popN1,castiguriN1,pm,d);

% supravietuire de tip elitist

[pop,castiguriN]=selectie\_parinti\_E(pop,popN,castiguri1,castiguriN);

% supravietuire generationala

%pop=popN;

[Maxim,k]=max(castiguriN);

Maxx=[Maxx Maxim];

perm=[perm;pop(k,1:n)];

end;

[mmm,poz]=max(Maxx);

ppp=perm(poz,1:n);

disp('Costul minim:');

disp(1/mmm);

disp('Permutarea');

disp(ppp);

for i=1:NMax

Mini(i)=1/Maxx(i);

end;

maximm=max(Mini);

figure

i=1:NMax;

plot(i,Mini(i),'ks-');

hold on

plot(poz,1/mmm,'rs');

axis ([1 NMax 0 maximm+1]);

end

function [parinti,castiguri,castiguri1]=selectie\_SUS(pop,d);

[dim,n]=size(pop);

castiguri1=zeros(dim,1);

for i=1:dim

castiguri1(i)=1/cost(pop(i,:),d);

end;

p=castiguri1;

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=zeros(dim,n);

castiguri=zeros(dim,1);

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=q(i))

parinti(k,1:n)=pop(i,1:n);

castiguri(k)=castiguri1(i);

r=r+1/dim;

k=k+1;

end;

i=i+1;

end;

end

function [popN,castiguriN]=crossover(pop,castiguri,pc,d);

popN=pop;

[dim,n]=size(pop);

castiguriN=castiguri;

for k=1:2:dim

x1=pop(k,1:n);

y1=pop(k+1,1:n);

r=unifrnd(0,1);

if(r<=pc)

[x2,y2]=CX(x1,y1);

popN(k,1:n)=x2;

popN(k+1,1:n)=y2;

castiguriN(k)=1/cost(x2,d);

castiguriN(k+1)=1/cost(y2,d);

end;

end;

end

function [popN,castiguriN]=mutatie\_inversiune(pop,castiguri,pm,d);

popN=pop;

castiguriN=castiguri;

[dim,n]=size(pop);

for k=1:dim

r=unifrnd(0,1);

if(r<pm)

i1=unidrnd(n-1);

j1=unidrnd(n);

while(j1==i1)

j1=unidrnd(n);

end;

i=min([i1,j1]);j=max([i1,j1]);

x=mutatie\_inversiune\_cromozomi(pop(k,:),i,j);

castiguriN(k)=1/cost(x,d);

popN(k,:)=x;

end;

end;

end

function [y]=mutatie\_inversiune\_cromozomi(x,i,j);

y=x;

y(i:j)=x(j:-1:i);

end

function [rezultat,castiguriR]=

selectie\_parinti\_E(pop,popN,castiguri,castiguriN);

rezultat=popN;

castiguriR=castiguriN;

[max1,i]=max(castiguri);

[max2,j]=max(castiguriN);

if(max1>max2)

[min1,k]=min(castiguriN);

rezultat(k,:)=pop(i,:);

castiguriR(k,:)=max1;

end;

end

function [c]=cost(x,d);

[m,n]=size(x);

c=d(x(n),x(1));

for i=1:n-1

c=c+d(x(i),x(i+1));

end;

end

function [d]=citeste\_date(nume);

d=load(nume);

end;

Pentru situaţia şi matricea distanţelor

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

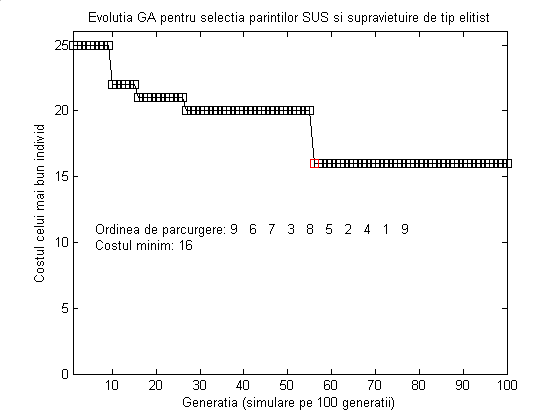
5 4 1 6 8 2 0 8 8

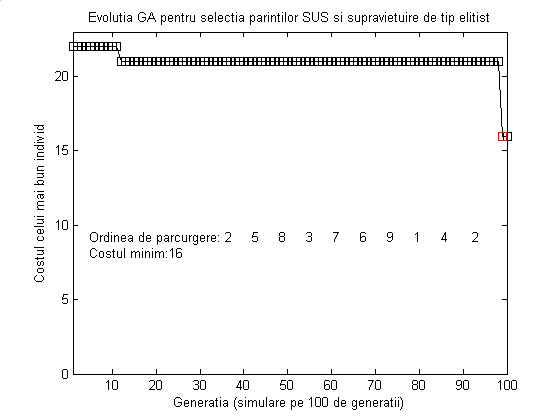
5 8 3 7 2 6 8 0 4

2 4 5 8 5 1 8 4 0

un traseul optim este: 1, 4, 2, 5, 8, 3, 7, 6, 9, 1, costul acestuia fiind 16.

La apelul *GA\_TSP('distante1.txt',300,0.8,0.01,100)* poate fi obţinută o soluţie optimă, conform figurilor de mai jos.





**Observaţii**

1. Evident, cele două rezultate prezentate mai sus sunt similare (nu are importanţă oraşul de plecare, iar ordinea de parcurgere este aceeaşi) şi corespund traseului optim.
2. Probabilitatea de mutaţie este superioară valorii , deci, pentru obţinerea unor rezultate mai bune, în medie mai mult de un cromozom per generaţie suferă o mutaţie
3. Pentru a asigura calculul unei valori mai bune, este recomandat ca dimensiunea populaţiei să fie mai mare (500, 600).

Pentru situaţia şi matricea distanţelor

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

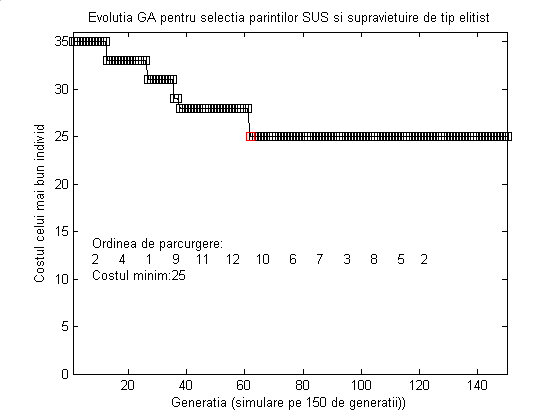
4 5 2 3 2 2 5 5 3 0 5 1

7 6 8 5 8 8 5 8 3 5 0 4

3 7 7 3 3 4 5 6 4 1 4 0

un traseu optim este: 1, 4, 2, 5, 8, 3, 7, 6, 10, 12, 11, 9, 1, costul acestuia fiind 25.

La apelul *GA\_TSP('distante2.txt',1000,0.8,0.01,150)* poate fi obţinută o soluţie optimă, conform figurii de mai jos. Costul minim calculat este minimul gobal.



Pentru situaţia şi matricea distanţelor

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

câteva trasee optime, cu costul minim 29, sunt:

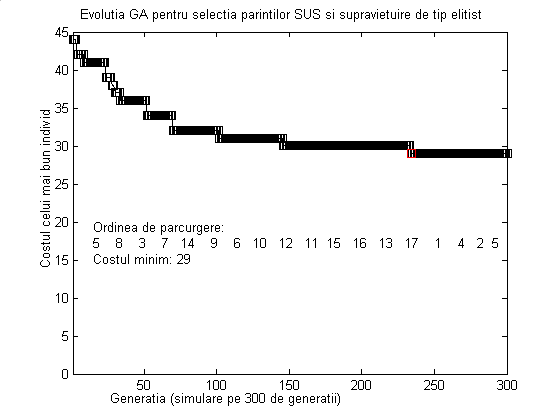
1, 17, 14, 10, 12, 11, 15, 16, 13, 9, 6, 7, 3, 8, 5, 2, 4, 1;

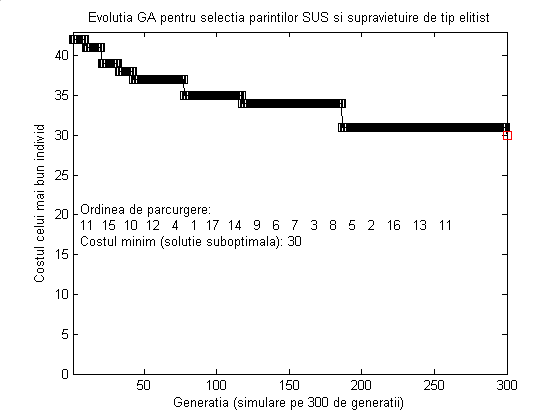
1, 4, 2, 5, 8, 3, 7, 6, 10, 12, 11, 15, 16, 13, 9, 14, 17, 1;

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

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

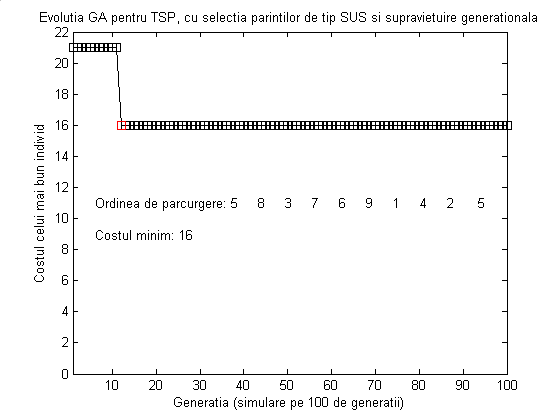
În prima figură de mai jos este prezentată soluţia optimă, obţinută la un apel *GA\_TSP('distante3.txt',4000,0.8,0.07,300)*, în timp ce în cea de-a doua este afişată o soluţie suboptimală, obţinută la un apel *GA\_TSP('distante3.txt',4000,0.8,0.01,300)*.



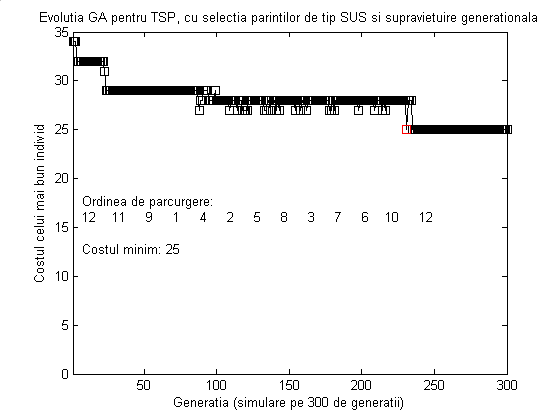


În cazul în care selecţia generaţiei următoare este strict *generaţională*, indivizii rezultaţi din recombinare şi mutaţie înlocuind complet populaţia curentă, pot fi obţinute următoarele evoluţii.

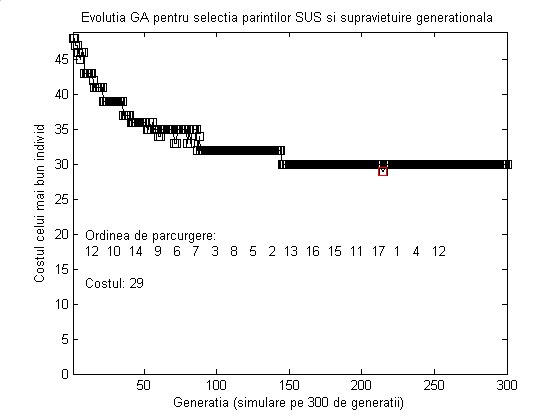
Pentru exemplul cu 9 oraşe, *GA\_TSP('distante1.txt',300,0.8,0.01,100)* poate furniza rezultatul optim, conform graficului de mai jos.

****

Pentru exemplul cu 12 oraşe, *GA\_TSP('distante2.txt',1000,0.8,0.01,300)* poate furniza rezultatul optim, conform graficului de mai jos.

****

Pentru exemplul cu 17 oraşe, GA\_TSP('distante3.txt',4000,0.8,0.05,300); poate furniza rezultatul optim, conform graficului de mai jos.



O altă variantă de implementare consideră pentru selectarea părinţilor **metoda SUS bazată pe probabilitatea de selecţie dată de funcţia rang**, cu presiunea de selecţie *s* (*s* a fost ales 1.5, pentru a evita pierderea excesivă de variaţie în cadrul populaţiei). Supravieţuirea este de tip elitist. Funcţia MATLAB folosită este prezentată în continuare.

function [parinti,castiguri,castiguri1]=selectie\_SUS\_rang(pop,d,sel);

[dim,n]=size(pop);

% este evaluata calitatea populatiei curente, pop

castiguri1=zeros(dim,1);

for i=1:dim

castiguri1(i)=1/cost(pop(i,:),d);

end;

p=castiguri1;

% populatia este sortata in functie de meritul sau

pop\_eval=[pop p];

pop\_sort=sortrows(pop\_eval,n+1);

% este obtinuta populatia sortata crescator in functie de merit

% calculeaza probabilitatea de selectie bazata pe rang

q=zeros(dim,1);

for i=1:dim

q(i)=((2-sel)/dim)+((2\*i\*(sel-1))/(dim\*(dim-1)));

end;

sss=sum(q);

q=q/sss;

% calculul probabilitatilor cumulate

sq=zeros(dim,1);

for i=1:dim

sq(i)=sum(q(1:i));

end;

% selecteaza parintii pe baza probabilitatii de selectie rang, cu

% algoritmul SUS

parinti=zeros(dim,n);

castiguri=zeros(dim,1);

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=sq(i))

parinti(k,1:n)=pop\_sort(i,1:n);

castiguri(k)=pop\_sort(i,n+1);

r=r+1/dim;

k=k+1;

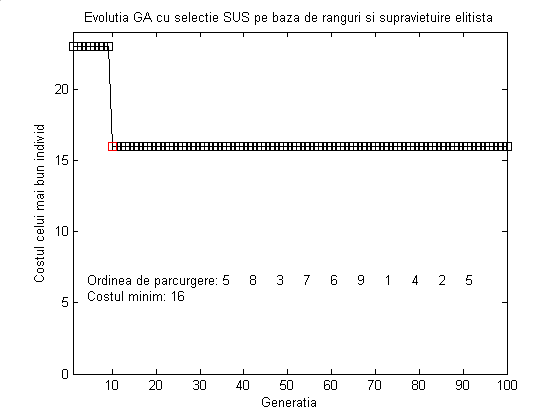
end;

i=i+1;

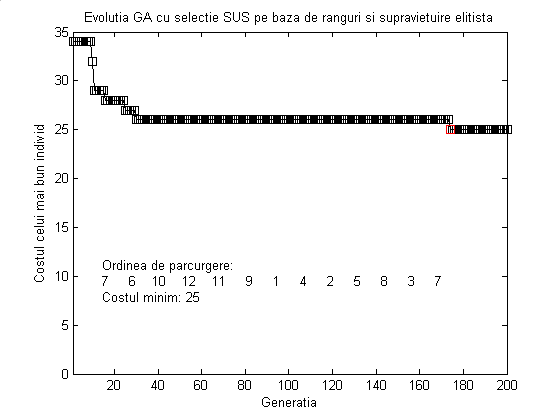
end;

end

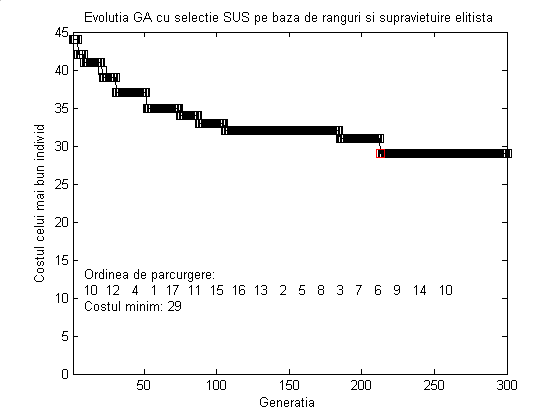
Pentru exempul cu 9 oraşe, la apelul *GA\_TSP('distante1.txt',300,0.8,0.01,100)* poate fi obţinută evoluţia prezentată în figura de mai jos.



Pentru exempul cu 12 oraşe, la apelul *GA\_TSP('distante3.txt',1000,0.8,0.01,200)* poate fi obţinută evoluţia prezentată în figura de mai jos.



Pentru exempul cu 17 oraşe, la apelul *GA\_TSP('distante3.txt',4000,0.8,0.01,300)* poate fi obţinută evoluţia prezentată în figura de mai jos.



Cea de-a treia variantă de implementare consideră pentru selectarea părinţilor metoda de tip turneu. Supravieţuirea este de tip elitist. Funcţia MATLAB folosită este prezentată în continuare.

function [popN,castiguri,castiguri1]=selectie\_turneu(pop,d);

[dim,n]=size(pop);

popN=zeros(dim,n);

i=0;

castiguri1=zeros(dim,1);

for i=1:dim

castiguri1(i)=1/cost(pop(i,:),d);

end;

castiguri=castiguri1;

for i=1:dim

p1=unidrnd(dim);

p2=unidrnd(dim);

while(p1==p2)

p2=unidrnd(dim);

end;

if(1/cost(pop(p1,:),d)>=1/cost(pop(p2,:),d))

popN(i,:)=pop(p1,:);

castiguri(i)=castiguri1(p1);

else

popN(i,:)=pop(p2,:);

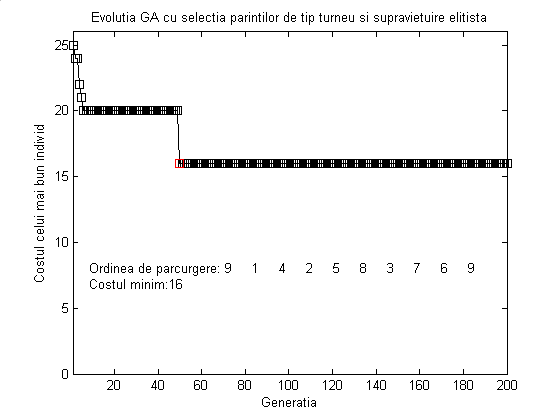
castiguri(i)=castiguri1(p2);

end;

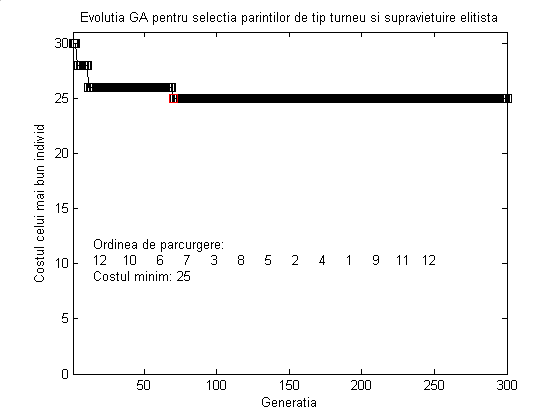
end;

end

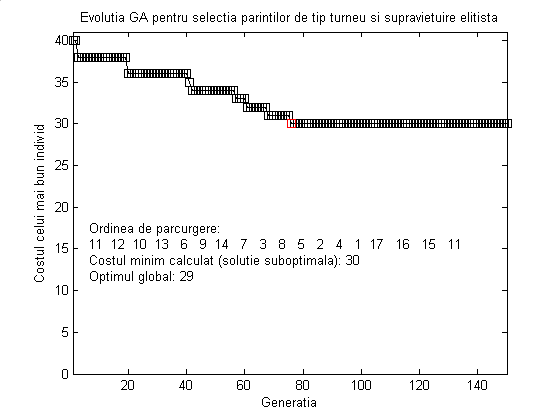
Pentru exempul cu 9 oraşe, la apelul *GA\_TSP('distante1.txt',300,0.8,0.01,200)* poate fi obţinută evoluţia prezentată în figura de mai jos.



Pentru exempul cu 12 oraşe, la apelul *GA\_TSP('distante2.txt',1500,0.8,0.01,300)* poate fi obţinută evoluţia prezentată în figura de mai jos.



Pentru exempul cu 17 oraşe, la apelul *GA\_TSP('distante3.txt',4000,0.8,0.05,150)* poate fi obţinută evoluţia prezentată în figura de mai jos.



**Observaţie.** Evident, în toate situaţiile prezentate mai sus pot fi obţinute soluţii suboptimale. Pentru a limita numărul acestor situaţii, dimensiunea populaţiei şi numărul de iteraţii trebuie să fie mai mare. În toate variantele însă, dacă parametrii sunt setaţi corespunzător, soluţiile suboptimale calculate de algoritmii genetici dezvoltaţi sunt în apropierea optimului teoretic. De exemplu, pe modelul cu 17 oraşe, dacă dimensiunea unei populaţii este în jurul valorii 4500, minimul calculat este în general sub valoarea 32 (optimul global este 29).

* + 1. **Rezolvarea problemei planificării activităţilor de tip sisteme de producţie bazate pe comandă (Job Shop Scheduling - JSS)**

Pentru problema planificării activităţilor în varianta JSS datele şi ipotezele de lucru sunt următoarele:

* *J*, un set de *n* sarcini, ;
* *O*, un set de operaţii; fiecare sarcină este constituită dintr-o secvenţă de *m* activităţi (operaţii), *m* independent de , , ; evident, ;
* *M*, o mulţime de *m* maşini care execută operaţii din *O*; activităţile fiecărei sarcini se execută pe câte o maşină din *M* (în cadrul unei sarcini nu există două operaţii care să fie executate pe aceeaşi maşină);
* în cadrul fiecărei sarcini, activităţile respectă o relaţie de precedenţă din punctul de vedere al execuţiei (o ordine de execuţie), astfel încât două sau mai multe activităţi nu pot fi executate simultan; cu alte cuvinte, fiecare sarcină este definită de un set ordonat de operaţii, ordinea fiind predefinită;
* o funcţie care asociază fiecărei operaţii o durata de execuţie; durata efectuării unei operaţii este independentă de planificare;
* activităţile, odată programate, nu îşi pot întrerupe execuţia;
* la fiecare moment de timp, pe oricare maşină, poate fi executată o singură operaţie.

Planificarea unei operaţii *o* revine la a îi aloca un timp de execuţie (pe maşina corespunzătoare lui *o*), un plan fiind o colecţie de astfel de alocări cu proprietatea că fiecare operaţie este inclusă cel mult o dată. Problema este de a găsi un plan cu proprietăţile:

* completitudine: planul conţine toate operaţiile mulţimii *O*;
* corectitudine: sunt satisfăcute toate restricţiile impuse prin ordinea de execuţie;
* optimalitate: durata totală a planului este minimă.

Rezolvarea problemei prin implementare GA este realizată prin menţinerea unei populaţii de genotipuri care reprezintă permutări ale mulţimii posibile de operaţii. Fie *n\*m* dimensiunea mulţimii de operaţii. O permutare, *pg*, corespunde ordinii de analizare a operaţiilor în scopul planificării lor şi are corespondent într-un plan (fenotip), *pf*, care constă în programarea fiecărei operaţii conform unei analize efectuate în ordinea în care aceasta apare în permutarea genotip. Planul corespuzător unei permutări *pg* poate fi construit pe baza următoarei proceduri.

Pentru efectuează:

* ;
* determină maşina pe care este executată , ;
* asignează timpul de start minim pentru , ţinând cont de ocuparea maşinii şi relaţiile de precedenţă pe care trebuie să le respecte relativ la operaţiile din cadrul sarcinii din care face parte şi care au fost deja planificate.

Funcţia de evaluare aplicată unui genotip furnizează durata de execuţie a planului corespunzător în spaţiul fenotipurilor. Scopul este de a minimiza funcţia de evaluare.

Alegerea operatorilor de variaţie este realizată astfel încât progeniturile să rămână în spaţiul genotipurilor. În cazul mutaţiei poate fi folosit de exemplu operatorul interschimbare. Deoarece relaţiile de adiacenţă nu au nici o semnificaţie în această problemă, operaţia de recombinare poate fi aleasă din oricare dintre cele prezentate în §3.4. În implementarea GA a fost utilizată recombinarea PMX.

În cazul acestei probleme operatorii de selecţie sunt, de asemenea, independenţi de reprezentarea cromozomială, deci pot fi utilizaţi oricare dintre cei prezentaţi în acest capitol, singurul scop fiind acela de a minimiza funcţia obiectiv. Operatorul de selecţie a părinţilor este de tip SUS cu distribuţia de probabilitate de selecţie FPS standard, în timp ce generaţia următoare este construită pe baza urmaşilor cu propagarea celui mai bun cromozom din generaţia curentă în locul celui mai slab urmaş, dacă progeniturile sunt sub acel cromozom din punct de vedere al funcţiei de evaluare.

Stabilirea populaţiei iniţiale este realizată aleator, iar condiţia terminală este de tip prag: de exemplu, dacă a fost atins un număr de iteraţii sau au fost evaluaţi un număr maxim de cromozomi.

În continuare este prezentată implementarea GA, în situaţia în care funcţia obiectiv, care trebuie maximizată, este definită astfel:

unde, pentru fiecare plan fezabil *x*, desemnează durata de execuţie a lui *x* (este întotdeauna nenulă).

Funcţiile *gen\_perm* şi *PMX* sunt identice cu cele prezentate în capitolele 2, respectiv 3 (mai puţin faptul că, în prezentarea PMX de mai sus părinţii sunt specificaţi direct în funcţie, în timp ce, pentru rezolvarea acestei probleme, ei sunt parametri de intrare). Funcţia *gen\_ini* este cea prezentată în problema TSP.

Evident, partea cea mai dificilă este aceea de a asocia unei permutări un plan care să respecte cerinţele de completitudine şi corectitudine, construcţia neasigurând, în acest caz, un plan optim, ci un plan cât mai bun conform permutării date. O variantă de rezolvare este următoarea. Fie *x* permutarea (genotipul) a cărui plan trebuie determinat (fenotipul) şi *n\*m* numărul de operaţii planificate. Vom presupune în continuare că şi aceasta este şi ordinea de efectuare a operaţiilor (ele sunt înscrise în sarcină în ordinea în care trebuie executate). Fiecare operaţie este codificată (reprezentată) prin numărul .

Iniţial, planifică toate operaţiile de la momentul 0 şi marchează fiecare activitate ca neexecutată (prin intermediul menţinerii unui vector binar, iniţializat cu toate valorile 0).

Cât timp nu au fost planificate toate activităţile

Pentru fiecare :

1. fie neplanificat, lista operatorilor care preced o, lista operatorilor care succed o; pot fi determinate astfel:
   * dacă face parte din sarcina şi este pe poziţia *r* în aceasta:

; evident, dacă , lista este vidă

* + altfel, face parte din sarcina şi este ultima operaţie:

;

* + , dacă , altfel este vidă (ultima operaţie din )

1. pentru fiecare
   * dacă *op* a fost executată, timpul de început al execuţiei operaţiei *o* trebuie sa fie după timpul de sfârşit al lui *op* (dacă o este planificată înainte, este resetat timpul de început cu timpul de sfârşit al lui *op*: );
2. fie plan*(o*) timpul de start calculat până la acest moment; dacă maşina pe care trebuie să se execute *o* nu este liberă, adică există un interval planificat pentru o altă activitate care se intersectează cu, atunci planifică *o* după acel interval; operaţia este repetată până când, pe maşina corepunzătoare, nu există programări conflictuale;
3. marchează *o* ca operaţie programată;
4. pentru fiecare
   * dacă *op* a fost executată, şi timpul de început al execuţiei operaţiei *op* este sub timpul de sfârşit al lui *o* (situaţie de programare eronată), atunci renunţă la planificarea lui *op* (este scoasă programarea de pe maşina care o execută) şi marchează *op* ca neplanificat.

Algoritmul furnizează, pe lângă un plan, costul acestuia şi asocierile la nivel de maşină (pentru fiecare operaţie este specificat timpul de început al execuţiei şi timpul la care operaţia se încheie).

function []=GA\_plan(dim,pc,pm,NMax);

[m,nm,durate,mo]=citeste\_date;

pop=gen\_ini(dim,nm);

Maxx=[];

perm=[];

Maxim1=0;

for i=1:NMax

[parinti,fob,fob1]=selectie\_SUS(pop,m,nm,durate,mo);

[popN1,fobN1]=crossover(parinti,m,nm,durate,mo,fob,pc);

[popN,fobN]=mutatie\_perm\_inserare(popN1,m,nm,durate,mo,fobN1,pm);

[popN,fobN]=selectie\_generatie\_urmatoare(pop,popN,fob1,fobN);

[Maxim,k]=max(fobN);

[asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,popN(k,1:nm));

Maxx=[Maxx Maxim];

if(Maxim>Maxim1)

A=asociere;

Maxim1=Maxim;

end;

perm=[perm;popN(k,1:nm)];

pop=popN;

end;

[mmm,poz]=max(Maxx);

ppp=perm(poz,1:nm);

disp('Costul minim:');

disp(1/mmm);

disp('Permutarea');

disp(ppp);

for i=1:NMax

Mini(i)=1/Maxx(i);

end;

maximm=max(Mini);

disp('Planificarea');

for k=1:m

disp('Masina');

disp(k);

disp(' Start op Stop');

A{k}=sortrows(A{k},1);

disp(A{k});

end;

figure

i=1:NMax;

plot(i,Mini(i),'ks-');

hold on

plot(poz,1/mmm,'rs');

axis ([1 NMax 0 maximm+1]);

end

function [m,nm,durate,mo]=citeste\_date();

f=fopen('operatii-durata1.txt');

nm=fscanf(f,'%d',1);

durate=fscanf(f,'%f',nm);

fclose(f);

%disp(durate);

f=fopen('masini-operatii1.txt');

m=fscanf(f,'%d',1);

%disp(m);

mo=fscanf(f,'%d',nm);

%disp(mo);

fclose(f);

end

function [asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,x);

% pentru fiecare masina: asocierea start op, sfarsit op

asociere=cell(1,m);

% operatie

% executat(i)=0 daca operatia i nu a fost executata;

%altfel executat(i)=1

executat=zeros(1,nm);

% un plan este un vector care asociaza fiecarei operatii 1..n un timp de

% inceput, plan(o)

cost=0;

plan=zeros(1,nm);

gata=0;

while(~gata)

gata=1;

for i=1:nm

o=x(i);

if(~executat(o))

% determinarea operatiilor care preced

pr=[];cat=fix(o/m);rest=o-cat\*m;

if(rest)

for k=1:rest-1

pr=[pr cat\*m+k];

end;

else

for k=1:m-1

pr=[pr (cat-1)\*m+k];

end;

end;

succ=[];

if(rest)

for k=rest+1:m

succ=[succ cat\*m+k];

end;

end;

[tt,nrp]=size(pr);

%masina pe care se executa

masina=mo(o);

%stabilirea timpului de executie in functie de predecesori

for k=1:nrp

% op precede o;

op=pr(k);

% daca op a fost efectuata

if(executat(op))

if(plan(o)<plan(op)+durate(op))

plan(o)=plan(op)+durate(op);

end;

end;

end;

[d1,d2]=size(asociere{masina});

gata1=0;

while(~gata1)

gata1=1;

for tt=1:d1

% daca operatia nu pate fi planificata la plan(o) din cauza ocuparii

%masinii cautam, dupa plan(o), primul interval liber in care sa incapa

if(((asociere{masina}(tt,1)<plan(o))&&(asociere{masina}(tt,3)>plan(o)))||...

((asociere{masina}(tt,1)<plan(o)+durate(o))&&(asociere{masina}(tt,3)>plan(o)+durate(o)))||...

((asociere{masina}(tt,1)>=plan(o))&&(asociere{masina}(tt,3)<=plan(o)+durate(o))))

plan(o)=asociere{masina}(tt,3);

gata1=0;

end;

end;

end;

asociere{masina}=[asociere{masina};[plan(o),o,plan(o)+durate(o)]];

executat(o)=1;

if(cost<plan(o)+durate(o))

cost=plan(o)+durate(o);

end;

[tt,nrs]=size(succ);

% pentru toti succesorii executati deja, elimina, daca este cazul,

% planificarea lor

for k=1:nrs

op=succ(k);

if((executat(op))&&(plan(op)<plan(o)+durate(o)))

masina=mo(op);

%scot planificarea lui op de pe masina

[d1,d2]=size(asociere{masina});

for tt=1:d1

if(asociere{masina}(tt,1)==plan(op))

ind=tt;

break;

end;

end;

xx=asociere{masina}(1:tt-1,:);

yy=asociere{masina}(tt+1:d1,:);

asociere{masina}=[xx;yy];

executat(op)=0;

gata=0;

end;

end;

end;

end;

end;

%verificam corectitudinea; nu va fi afisata nici o pereche de operatii efectuate in

%ordine incorecta

incorect=[];

for k=1:nm/m

for kk=1:m-1

ope=[(k-1)\*m+kk (k-1)\*m+kk+1];

% ope(1) trebuie sa se termine inainte de ope(2), conditie violata de plan

if(plan(ope(1))+durate(ope(1))>plan(ope(2)))

incorect=[incorect;ope];

end;

end;

end;

disp(incorect);

end

function [parinti,fob,fob1]=selectie\_SUS(pop,m,nm,durate,mo);

[dim,nm]=size(pop);

fob1=zeros(dim,1);

for i=1:dim

[asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,pop(i,1:nm));

fob1(i)=1/cost;

end;

p=fob1;

s=sum(p);

p(1:dim)=p(1:dim)/s;

q=zeros(dim,1);

for i=1:dim

q(i)=sum(p(1:i));

end;

parinti=pop;

fob=fob1;

i=1;k=1;r=unifrnd(0,1/dim);

while(k<=dim)

while(r<=q(i))

parinti(k,1:nm)=pop(i,1:nm);

fob(k)=fob1(i);

r=r+1/dim;

k=k+1;

end;

i=i+1;

end;

end

function [popN,fobN]=crossover(pop,m,nm,durate,mo,fob,pc);

popN=pop;

[dim,nm]=size(pop);

fobN=fob;

for k=1:2:dim

x1=pop(k,1:nm);

y1=pop(k+1,1:nm);

r=unifrnd(0,1);

if(r<=pc)

[x2,y2]=PMX(nm,x1,y1);

popN(k,1:nm)=x2;

popN(k+1,1:nm)=y2;

[asociere,plan,cost1]=permutare\_plan(m,nm,durate,mo,x2);

fobN(k)=1/cost1;

[asociere,plan,cost2]=permutare\_plan(m,nm,durate,mo,y2);

fobN(k+1)=1/cost2;

end;

end;

end

function [popN,fobN]=mutatie\_perm\_inserare(pop,m,nm,durate,mo,fob,pm);

popN=pop;

fobN=fob;

[dim,nm]=size(pop);

for i=1:dim

r=unifrnd(0,1);

if(r<pm)

p=zeros(1,2);

p(1)=unidrnd(nm);

p(2)=unidrnd(nm);

while(p(1)==p(2))

p(2)=unidrnd(nm);

end;

poz=sort(p);

popN(i,1:poz(1))=pop(i,1:poz(1));

popN(i,poz(1)+1)=pop(i,poz(2));

popN(i,poz(1)+2:poz(2))=pop(i,poz(1)+1:poz(2)-1);

popN(i,poz(2)+1:nm)=pop(i,poz(2)+1:nm);

[asociere,plan,cost]=permutare\_plan(m,nm,durate,mo,popN(i,1:nm));

fobN(i)=1/cost;

end;

end;

end

function [rezultat,fob]=selectie\_generatie\_urmatoare(pop,popN,fob1,fobN);

rezultat=popN;

fob=fobN;

[max1,i]=max(fob1);

[max2,j]=max(fobN);

if(max1>max2)

[min1,k]=min(fobN);

rezultat(k,:)=pop(i,:);

fob(k)=max1;

end;

end

*Setul de test 1*

Ca date de intrarea au fost folosite:

* numărul sarcinilor: 5;
* numărul maşinilor: 4, maşinile fiind identificate prin numere de la 1 la 4;
* numărul operaţiilor: 20, operaţiile fiind considerate numere de la 1 la 20;
* duratele de execuţie ale operaţiilor de la 1 la 20 (vectorul *durate*):

[3 2 5 1 3 5 9 4 3 7 3 5 4 4 7 6 4 9 2 7] ;

* alocarea operaţiilor (de la 1 la 20) pe maşini (vectorul *mo*: maşini-operaţii):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [2 1 4 3 | 2 1 3 4 | 1 2 3 4 | 4 1 2 3 | 2 1 3 4] |
| corespunzător | corespunzător | corespunzător | corespunzător | corespunzător |

* precedenţele: aşa cum am specificat în prezentarea modalităţii de generare a unui plan corespunzător unei permutări, în fiecare sarcină operaţiile sunt identificate prin , ordinea efectuării operaţiilor fiind de la cea etichetată cu valoarea minima (), crescător până la ultima operaţie (etichetată cu valoarea minima ()).

La un apel

>> GA\_plan(300,0.8,0.02,80);

pot fi obţinute rezultatele:

Costul minim: 33

Planificarea cea mai buna calculata de GA

Masina 1

Start op Stop

0 9 3

3 6 8

9 2 11

11 14 15

15 18 24

Masina 2

Start op Stop

0 5 3

3 1 6

6 17 10

10 10 17

17 15 24

Masina 3

Start op Stop

8 7 17

17 11 20

20 4 21

24 19 26

26 16 32

Masina 4

Start op Stop

0 13 4

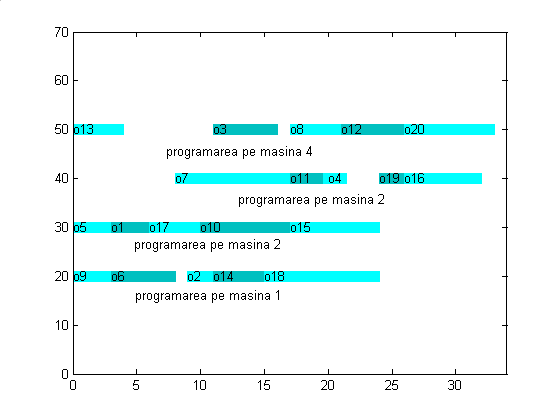
11 3 16

17 8 21

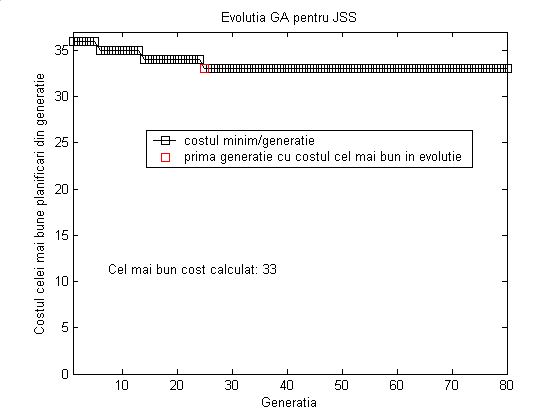
21 12 26

26 20 33

Rezultatul corespunde următoarei planificări, unde fiecare activitate *a* este marcată prin *oa*.



Evoluţia GA este prezentată în următorul grafic.



*Setul de test 2*

Ca date de intrarea au fost folosite:

* numărul sarcinilor: 5;
* numărul maşinilor: 7, maşinile fiind identificate prin numere de la 1 la 7;
* numărul operaţiilor: 35, operaţiile fiind considerate numere de la 1 la 35;
* duratele de execuţie ale operaţiilor de la 1 la 35 (vectorul *durate*):

[4 11 6 5 7 2 3 2 9 3 12 3 4 5 1 3 5 9 4 8 7 3 5 4 4 7 6 4 9 4 7 4 15 11 3] ;

* alocarea operaţiilor (de la 1 la 35) pe maşini (vectorul *mo*: maşini-operaţii):

|  |  |
| --- | --- |
| Maşinile | Operaţiile din fiecare sarcină, în ordinea execuţiei lor pe maşini |
| 2 1 4 6 5 3 7 | corespunzător |
| 4 1 2 3 7 6 5 | corespunzător |
| 7 1 3 4 2 5 6 | corespunzător |
| 1 2 3 4 5 6 7 | corespunzător |
| 7 5 3 4 2 1 6 | corespunzător |

* precedenţele: aşa cum am specificat în prezentarea modalităţii de generare a unui plan corespunzător unei permutări, în fiecare sarcină operaţiile sunt identificate prin , ordinea efectuării operaţiilor fiind de la cea etichetată cu valoarea minima (), crescător până la ultima operaţie (etichetată cu valoarea minima ()).

La un apel

>> GA\_plan(500,0.8,2/500,50);

pot fi obţinute rezultatele:

Costul minim 57

Planificarea cea mai buna calculată de GA

Masina 1

Start op Stop

0 22 3

3 16 6

11 9 20

20 2 31

43 34 54

Masina 2

Start op Stop

0 1 4

4 23 9

20 10 23

23 19 27

28 33 43

Masina 3

Start op Stop

6 17 11

11 24 15

16 31 23

23 11 35

52 6 54

Masina 4

Start op Stop

0 8 2

11 18 20

20 25 24

24 32 28

31 3 37

Masina 5

Start op Stop

12 30 16

25 26 32

32 20 40

45 5 52

52 14 57

Masina 6

Start op Stop

32 27 38

38 4 43

43 13 47

47 21 54

54 35 57

Masina 7

Start op Stop

0 15 3

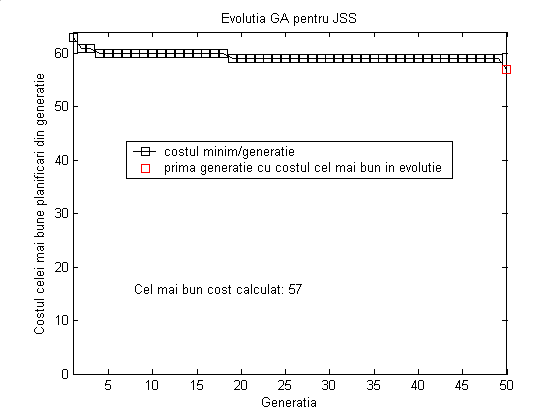
3 29 12

35 12 38

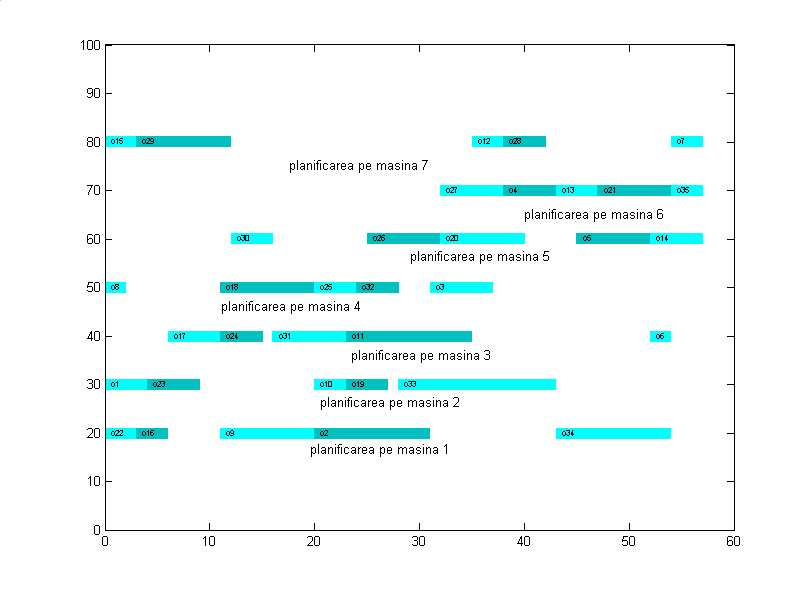
38 28 42

54 7 57

Evoluţia GA este prezentată în următorul grafic.



Rezultatul corespunde următoarei planificări, unde fiecare activitate *a* este marcată prin *oa*.



*Setul de test 3*

Ca date de intrarea au fost folosite:

* numărul sarcinilor: 8;
* numărul maşinilor: 7, maşinile fiind identificate prin numere de la 1 la 7;
* numărul operaţiilor: 56, operaţiile fiind considerate numere de la 1 la 56;
* duratele de execuţie ale operaţiilor de la 1 la 35 (vectorul *durate*):

[4 11 6 5 7 4 3 4 9 5 12 5 4 4 7 5 9 4 8 7 4 5 5 4 7 6 4 9 4 7 6 15 11 4 6 15 11 4 5 4 6 7 8 5 11 5 4 7 4 4 5 7 8 6 9 5 4 12 5];

* alocarea operaţiilor (de la 1 la 56) pe maşini (vectorul *mo*: maşini-operaţii):

|  |  |
| --- | --- |
| Maşinile | Operaţiile din fiecare sarcină, în ordinea execuţiei lor pe maşini |
| 2 1 4 6 5 3 7 | corespunzător |
| 4 1 2 3 7 6 5 | corespunzător |
| 7 1 3 4 2 5 6 | corespunzător |
| 1 2 3 4 5 6 7 | corespunzător |
| 7 5 3 4 2 1 6 | corespunzător |
| 1 3 4 5 6 2 7 | corespunzător |
| 7 3 1 6 5 2 4 | corespunzător |
| 1 3 2 5 6 4 7 | corespunzător |

* precedenţele: aşa cum am specificat în prezentarea modalităţii de generare a unui plan corespunzător unei permutări, în fiecare sarcină operaţiile sunt identificate prin , ordinea efectuării operaţiilor fiind de la cea etichetată cu valoarea minima (), crescător până la ultima operaţie (etichetată cu valoarea minima ()).

La un apel

>> GA\_plan(100,0.8,10/100,70);

pot fi obţinute rezultatele:

Costul minim:83

Planificarea cea mai bună calculată deGA

Masina 1

Start op Stop

0 36 5

5 22 9

9 50 17

17 16 24

24 9 33

33 45 40

40 2 51

66 34 77

Masina 2

Start op Stop

0 1 4

9 23 14

23 52 32

33 10 38

38 19 42

42 41 47

51 33 66

66 48 71

Masina 3

Start op Stop

11 37 15

17 51 23

24 17 29

29 44 33

33 24 38

38 31 45

45 11 57

69 6 73

Masina 4

Start op Stop

0 8 4

19 38 25

29 18 38

38 25 42

45 32 51

51 3 57

57 55 69

71 49 78

Masina 5

Start op Stop

21 30 25

25 39 32

37 53 42

42 26 49

49 20 57

57 47 61

62 5 69

73 14 78

Masina 6

Start op Stop

32 40 40

40 46 44

53 54 57

57 4 62

62 21 69

69 13 73

73 27 79

79 35 83

Masina 7

Start op Stop

0 15 4

7 43 12

12 29 21

47 42 58

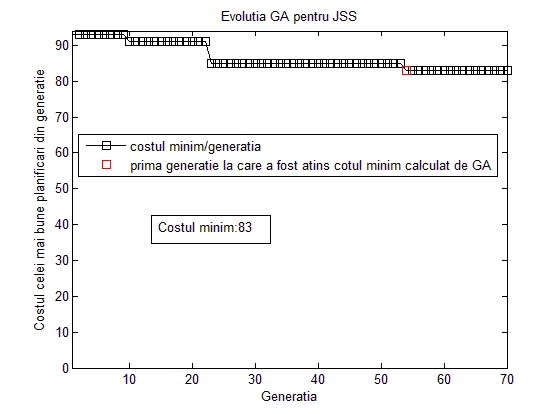
58 12 63

69 56 74

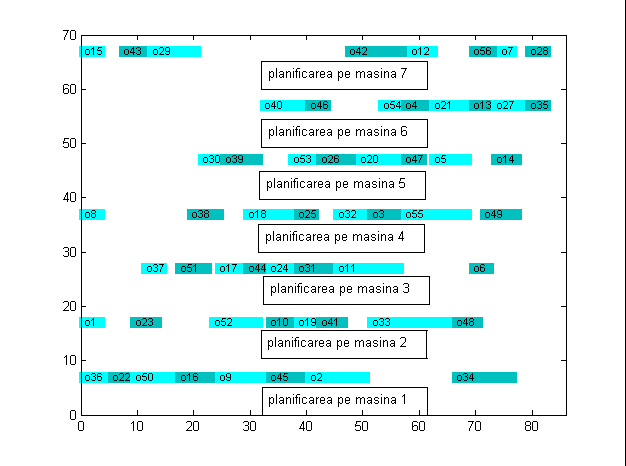
74 7 77

79 28 83

Evoluţia GA este prezentată în următorul grafic.



Rezultatul corespunde următoarei planificări, unde fiecare activitate *a* este marcată prin *oa*.



La un apel

>> GA\_plan(160,0.8,20/160,250);

Costul minim:

79

Planificarea cea mai buna calculata

Masina 1

Start op Stop

0 36 5

5 22 9

11 50 19

19 16 26

26 9 35

35 45 42

42 2 53

64 34 75

Masina 2

Start op Stop

0 1 4

9 23 14

25 52 34

35 10 40

40 19 44

44 41 49

49 33 64

64 48 69

Masina 3

Start op Stop

5 37 9

9 44 13

14 24 19

19 51 25

26 17 31

31 31 38

40 11 52

71 6 75

Masina 4

Start op Stop

0 8 4

9 38 15

19 25 23

31 18 40

40 32 46

53 3 59

59 55 71

71 49 78

Masina 5

Start op Stop

15 39 22

24 30 28

28 26 35

35 53 40

44 20 52

52 47 56

64 5 71

71 14 76

Masina 6

Start op Stop

22 40 30

35 27 41

44 46 48

48 54 52

52 21 59

59 4 64

65 13 69

75 35 79

Masina 7

Start op Stop

0 15 4

4 43 9

13 29 22

41 28 45

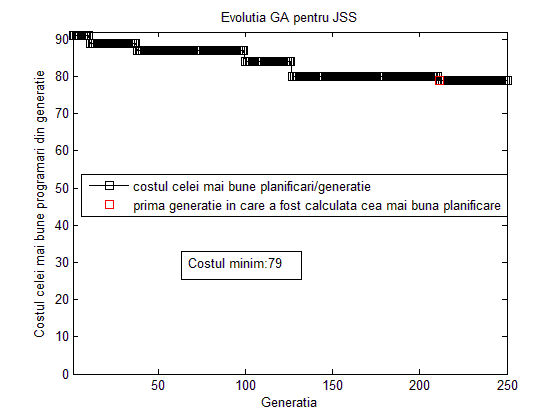
49 42 60

60 12 65

71 56 76

76 7 79

Evoluţia GA este prezentată în următorul grafic.



Rezultatul corespunde următoarei planificări, unde fiecare activitate *a* este marcată prin *oa*.

