**TC 3. Problema rucsacului de tip 0-1**

Problema rucsacului de tip 0-1 este enunţată în cele ce urmează. Fiind date *m* obiecte, fiecare având asociate o valoare şi respectiv un cost de selecţie, trebuie determinat un set de obiecte cu proprietatea că este de valoare maximă (unde valoarea unei mulţimi de obiecte este definită ca suma valorilor obiectelor ce o compun) şi costul (definit ca suma costurilor obiectelor setului) este sub un prag dat, *Cmax*. Problema este de tip 0-1 pentru că nu pot fi selectate fracţiuni dintr-un obiect, ci întregul obiect (dacă prin această operaţie nu este depăşit *Cmax*).

O soluţie posibilă este dată de un set de obiecte selectate astfel încât costul lui este inferior valorii *Cmax*. Dacă *val* este vectorul valorilor asociate celor *m* obiecte şi *cost* este vectorul costurilor, reprezentarea unui candidat la soluţie poate fi realizată printr-un vector de *m* elemente, *v*, unde

şi

Funcţia de evaluare asociată lui *v* calculează valoarea asociată selecţiei reprezentate de *v*:

Este evident că fiecare genotip *v* corespunde unui număr şi *v* este reprezentarea binară a lui *R-1* şi cu proprietatea că nu este depăşit pragul *Cmax*.

La fiecare moment de timp, populaţia este formată din *N* indivizi reprezentaţi aşa cum a fost explicat mai sus. La momentul iniţial, este generat aleator un număr , genotipul corespunzător lui fiind reprezentarea binară a lui *R-1*; *R* este selectat pentru reprezentare numai dacă nu este depăşit pragul *Cmax*. Procedeul continuă până la generarea a *N* cromozomi.

Selecţia părinţilor este realizată prin următorul procedeu. De *N* ori sunt alese perechi de cromozomi din populaţia curentă şi este selectat cel mai bun dintre ei, din punctul de vedere al funcţiei de evaluare. Recombinarea este realizată cu o probabilitate *pc* () şi este proiectată astfel încât progeniturile să rămână admisibile (costul asociat să fie inferior lui *Cmax*): pentru doi indivizi şi pentru o poziţie generată aleator sunt obţinute 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

Dacă astfel obţinuţi nu îndeplinesc proprietatea de admisibilitate, este selectată o altă pereche de indivizi şi este aplicat acelaşi mecanism, până când este obţinută o pereche de genotipuri (soluţii admisibile ale problemei de rezolvat). Dacă o pereche de părinţi nu este selectată pentru încrucişare, aceasta este menţinută în populaţia următoare.

Operatorul mutaţie este aplicat cu o probabilitate pm (în general ) unui individ *v* şi revine la selectarea aleatoare a unei poziţiei şi modificarea valorii şi efectuarea operaţiei (valoarea genei poz este schimbată din 0 în 1 sau invers). Dacă rezultatul nu este admisibil, este selectat un alt individ pentru mutaţie.

Mecanismul de înlocuire a populaţiei revine la selectarea acelor indivizi aleşi ca părinţi dar pentru care nu s-a realizat operaţia de încrucişare (valoarea generată aleator este inferioară valorii *pc*) şi pentru progeniturile (eventual mutante) rezultate. Fiecare generaţie are deci dimensiunea constantă, *N*.

Condiţia terminală controlează numărul de iteraţii efectuate. Se poate implementa şi o variantă în care să fie controlată şi calitatea populaţiei, evaluată ca medie a funcţiei de evaluare sau ca valoarea maximă a funcţiei de evaluare calculată pentru fiecare din indivizii populaţiei curente.

În continuare sunt prezentate funcţiile MATLAB pentru implementarea căutării evolutive. Datele sunt preluate din fişierul nume şi sunt reprezentate de costurile şi valorile asociate celor *m* obiecte. Testele au fost efectuate pentru *m*=16 şi *m*=32.

% citirea datelor din fisier

function [val,cost,costM]=citeste\_date(nume);

f=fopen(nume);

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

val=fscanf(f,'%lg',[m,1]);

cost=fscanf(f,'%lg',[m,1]);

costM=fscanf(f,'%lg',1);

fclose(f);

end

% generarea populaţiei iniţiale

function [pop,val,cost,costM]=genereza\_ini(nume,N);

[val,cost,costM]=citeste\_date(nume);

[m,p]=size(val);

pop=zeros(N,m);

i=0;

while(i<N)

%se lucreaza pe m biti, generez R, un numar intre 1 si 2^m

%si consideram reprezentarea binara

R=unidrnd(2^m);

x=repr\_bin(R-1,m);

[ok,sum]=verifica(x,cost,costM);

if(ok)

i=i+1;

pop(i,:)=x;

end;

end;

end

% obtinerea reprezentarii binare a unui numar

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

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

end

% verificarea admisibilitatii unei solutii potentiale

function [OK,sum]=verifica(x,cost,costM);

sum=x\*cost;

OK = sum<=costM;

end

% operatorul de mutatie

function [y]=mutatie(x,poz);

y=x;

y(poz)=not(x(poz));

end

% operatorul de incrucisare

function [y1,y2]=crossover(x1,x2,poz);

[m,p]=size(x1);

y1=x1;y2=x2;

y1(1:poz)=x1(1:poz);

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

y2(1:poz)=x2(1:poz);

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

end

% selectia parintilor

function [parinti]=selectie(pop,val,cost,costM);

[N,m]=size(pop);

parinti=zeros(N,m);

i=0;

for i=1:N

p1=unidrnd(N);

p2=unidrnd(N);

%functia de fitness - suma vi\*gi, gi=1,

%daca a fost ales obiectul i, 0 altfel

if(pop(p1,:)\*val>=pop(p2,:)\*val)

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

else

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

end;

end;

end

% inlocuirea populatiei curente cu generatia urmatoare

function [popN]=trecere(pop,val,cost,costM,pc,pm);

[parinti]=selectie(pop,val,cost,costM);

[N,m]=size(pop);

popN=zeros(N,m);

i=0;

%aplica operatia de crossover

for i=1:2:N

p1=unidrnd(N);x1=parinti(p1,:);

p2=unidrnd(N);x2=parinti(p2,:);

prc=unifrnd(0,1);

if(prc<=pc)

cont=1;

%trebuie ca progentiturile sa fie solutii admisibile

while(cont)

poz=unidrnd(m);

%disp(poz);

[y1,y2]=crossover(x1,x2,poz);

[OK1,sum1]=verifica(y1,cost,costM);

[OK2,sum2]=verifica(y2,cost,costM);

if((OK1==1)&&(OK2==1))

popN(i,:)=y1;

popN(i+1,:)=y2;

cont=0;

end;

end;

else

popN(i,:)=x1;

popN(i+1,:)=x2;

end;

end;

%aplica mutatia

for i=1:N

prm=unifrnd(0,1);

if(prm<=pm)

cont=1;

%trebuie ca progentitura sa fie solutie admisibila

while(cont)

poz=unidrnd(m);

x=popN(i,:);

[y]=mutatie(x,poz);

[OK,sum]=verifica(y,cost,costM);

if(OK==1)

popN(i,:)=y;

cont=0;

end;

end;

end;

end;

end

% calculul valorii maxime a functiei de evaluare

function [Q,x,V]=maximQ(pop,val);

[N,m]=size(pop);

Q=0;

V=zeros(N);

for i=1:N

qm=pop(i,:)\*val;

V(i)=qm;

if(qm>=Q)

Q=qm;

x=pop(i,:);

end;

end;

end

% implementarea cautarii evolutive

function []=GA\_Knapsack(nume,N,pc,pm);

%GA\_Knapsack('date4.txt',2000,0.9,1.0/8);

% - obtinerea solutiei exacte

%pentru 32 de componente

%pentru 16 componente GA\_Knapsack('date.txt',500,0.8,1.0/8);

%- solutia exacta

[pop,val,cost,costM]=genereaza\_ini(nume,N);

[Q,x,V]=maximQ(pop,val);

%disp(x);

%disp(Q);

figure;

i=1:N;

plot(i,V(i),'r-');

hold on;

for j=1:15

[popN]=trecere(pop,val,cost,costM,pc,pm);

pop=popN;

[Q,x,V]=maximQ(pop,val);

%disp(Q);

%disp(x);

end;

[Q,x,V]=maximQ(pop,val);

%disp(x);

i=1:N;

plot(i,V(i),'k-');

hold on

for j=1:50

[popN]=trecere(pop,val,cost,costM,pc,pm);

pop=popN;

[Q,x,V]=maximQ(pop,val);

%disp(Q);

%disp(x);

end;

[Q,x,V]=maximQ(pop,val);

disp('Solutia optima calculata');

disp(x);

disp('Valoarea optima calculata');

disp(Q);

i=1:N;

plot(i,V(i),'b-');

end

La apelul GA\_Knapsack('date.txt',500,0.8,1.0/8); fişierul *date.txt* conţine următoarele informaţii:

Numărul de componente:

16

Valoarea obţinută la alegerea fiecărei componente

4.5 6 8 5.4 10.2 3.2 4.2 8.3 3.2 4.5 9 10.9 5 6.2 7 8.2

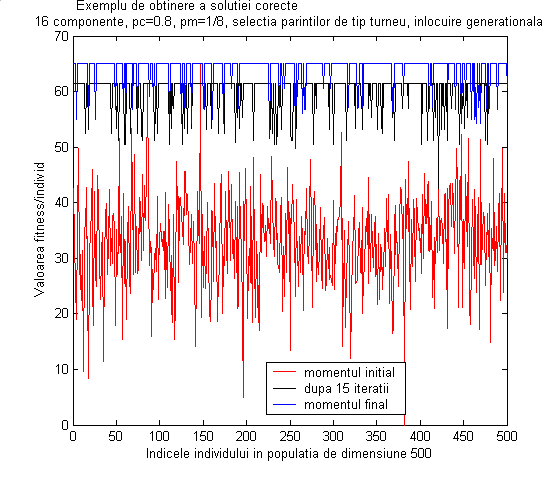
Costul alegerii fiecărei componente

3 2 2.4 6.8 5 6.2 5 4.8 5 0.2 3 4.8 7 7.1 9.7 4

Costul maxim permis

26.3

În continuare este prezentat un exemplu de evoluţie a EA:



Soluția optimă calculată:

0 1 1 0 1 0 0 1 0 1 1 1 0 0 0 1

Valoarea optimă calculată:

65.1000

La apelul GA\_Knapsack('date4.txt',2000,0.9,1.0/8); fişierul *date4.txt* conţine următoarele informaţii:

Numărul de componente:

32

Valoarea obţinută la alegerea fiecărei componente

2.5 6 8 3.4 10.2 3.2 4.2 8.3 3.2 4.5 9 10.9 5 6.2 7 8.2 9.2 1.3 4 5 8.9 0.4 7.3 4.1 8

3 11 2.5 3 1.2 5 2.3

Costul alegerii fiecărei componente

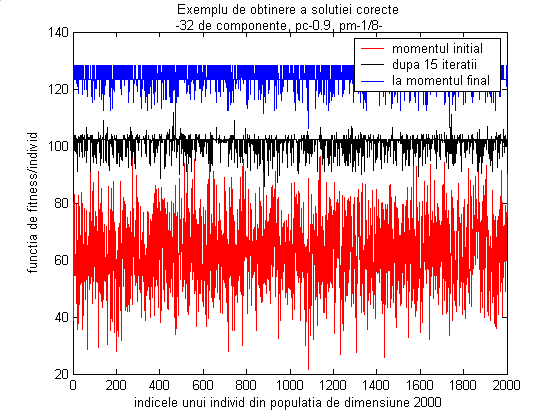
7 3.1 2.4 8.8 5 5.2 5 4.8 5 5.2 6 4.8 7 1.1 2.7 4.9 5 5.6 8 2.5 4 4 2.2 5.2 3.1 6.4

3.3 7 6.5 8 1.4 9.8

Costul maxim permis

56.6

În continuare este prezentat un exemple de evoluţie a EA:



Soluția optimă calculată:

0 1 1 0 1 0 0 1 0 0 1 1 0 1 1 1

1 0 0 1 1 0 1 0 1 0 1 0 0 0 1 0

Valoarea optimă calculată

128.2000