**TC 5-6. Problema celor opt regine. Aplicații ale strategiilor evolutive și programării evolutive în probleme de optimizare**

Problema este de a determina o configuraţie în care 8 regine pot fi plasate pe o tablă de şah astfel încât să nu se atace reciproc (nici o pereche de regine nu trebuie plasată pe aceeaşi linie, coloană sau diagonală). Aceasta poate fi generalizată imediat la cazul a *N* regine pentru o tablă de şah *N* x *N* -dimensională.

Deoarece pe fiecare linie şi coloană a tablei de şah trebuie plasată o singură regină, configuraţiile sunt reprezentate succesiv printr-un vector cu *N* componente. Valoarea componentei *i* dintr-un vector configuraţie reprezintă indicele coloanei în care este plasată regina aflată pe linia *i*. Din această reprezentare rezultă că două regine nu pot fi plasate pe aceeaşi linie a tablei de şah. De asemenea, dacă vom presupune că elementele vectorului configuraţie sunt distincte (deci vectorul configuraţie corespunde unei permutări pe mulţimea ), rezultă că două regine nu pot fi plasate pe aceeaşi coloană a tablei de şah.

Rezolvarea clasică a unei astfel de probleme este prin căutare directă, utilizând metoda backtracking: iniţial este plasată o regină; după plasarea a *n* regine, este căutată o poziţie posibilă pentru plasarea celei de-a *n*+1-a regine (o poziţie din care noua regină să nu se afle în situaţie de atac cu nici una din celelalte *n* regine deja plasate pe tabla de şah). Dacă nu există nici o astfel de poziţie, cea de-a *n*-a regină este repoziţionată pe tabla de şah, dacă acest lucru este posibil.

O abordare evolutivă a acestei probleme este total diferită de căutarea directă şi este de tip non-incremental. Soluţiile candidat sunt complete, în sensul că fiecare vector configuraţie este instanţiat cu o permutare. Din punct de vedere al reprezentării, un fenotip, care reprezintă o configuraţie a tablei de şah astfel încât două regine să nu fie plasate pe aceeaşi linie sau aceeaşi coloană (este soluţie potenţială), este codificat prin intermediul vectorului configuraţie, permutare a mulţimii . Spaţiul genotipurilor este deci format din mulţimea permutărilor setului . Evident, nu toate elementele acestui spaţiu sunt soluţii admisibile (două regine pot fi plasate astfel încât să fie pe o aceeaşi diagonală a tablei de şah, deci să se afle în poziţie de atac reciproc).

Calitatea fiecărui genotip *x*, , poate fi evaluată în termenii numărului de perechi de regine care se află în poziţie de atac (numărul de perechi cu proprietatea că ): dacă acesta este 0 atunci genotipul este soluţie, în caz contrar valoarea calculată este strict pozitivă. Problema de optim care trebuie rezolvată este deci de minimizare funcţie *f* sau, echivalent, maximizarea funcţiei , unde este numărul maxim de perechi de regine care se pot afla în poziţie de atac (de exemplu în cazul ). În implementarea algoritmului, fiecărui genotip îi este asociată valoarea funcţiei de evaluare, deci datele cu care se lucrează sunt vectori *N*+1-dimensionali: primele *N* componente reprezintă configuraţia şi ultima componentă valoarea funcţiei de evaluare corespunzătoare acesteia.

La fiecare moment de timp, populaţia este constituită dintr-un multiset cu *dim* elemente (de exemplu, pentru , pentru ). Pentru această problemă, la fiecare generaţie sunt înlocuiţi câte 2 indivizi (cei mai slabi din punctul de vedere al funcţiei de evaluare) cu copii a doi părinţi. Cei doi părinţi sunt selectaţi aleator din populaţia curentă, . Operatorul de încrucişare este proiectat astfel încât progeniturile să păstreze proprietatea de a fi permutări: pentru doi indivizi şi pentru o poziţie sunt obţinute progeniturile astfel:

* copiază primele *poz* elemente din , respectiv în , respectiv
* copiază în ultimele *dim-poz+1* poziţii din , respectiv , elementele din , respectiv , începând cu primul şi până când au fost completate poziţiile din , respectiv , astfel încât , respectiv să nu conţină dubluri

Operatorul mutaţie este aplicat cu probabilitatea de 0.8 uneia din cele două progenituri rezultate după aplicarea operatorului de recombinare şi revine la selectarea aleatoare a două poziţii diferite din genotipul modificat şi interschimbarea valorilor din acele poziţii.

Condiţia terminală este formulată astfel: a fost atinsă o configuraţie soluţie sau a fost depăşit pragul dat pentru numărul maxim de generaţii.

În continuare sunt prezentate funcţiile MATLAB utilizate şi câteva exemple de aplicare a căutării evolutive descrise mai sus.

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

pop=zeros(dim,N+1);

% fiecare membru al populatiei este o permutare pe 1,2,..,N

% la care este adaugata valoarea functiei obiectiv N-numarul de

% perechi de regine care se ataca reciproc

for i=1:dim

x=gen\_perm(N);

pop(i,1:N)=x;

pop(i,N+1)=valoare(x);

end;

end

% generarea aleatoare a unei permutari pe {1,2,…,N}

function [y]=gen\_perm(N);

y=zeros(1,N);

for i=1:N

gata=0;

while(~gata)

v=unidrnd(N);

if(~ismember(v,y))

y(i)=v;

gata=1;

end;

end;

end;

disp(y);

end

% functia de evaluare calculata pentru un genotip

function [val]=valoare(x);

[m,N]=size(x);

v=0;

for i=1:N-1

for j=i+1:N

if(abs(i-j)==abs(x(i)-x(j)))

v=v+1;

end;

end;

end;

val=N\*(N-1)/2-v;

end

% recombinarea

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

[m,p]=size(x1);

N=p-1;

y1=x1;y2=x2;

y1=copiaza\_rest(y1,x2,poz,N);

y2=copiaza\_rest(y2,x1,poz,N);

y1(N+1)=valoare(y1(1:N));

y2(N+1)=valoare(y2(1:N));

%disp(y1);

%disp(y2);

end

function [y]=copiaza\_rest(y,x,poz,N)

i=poz+1;

while(i<=N)

for j=1:N

if(~ismember(x(j),y(1:i-1)))

y(i)=x(j);

i=i+1;

break;

end;

end;

end;

end

% mutatia

function [y]=mutatie(x,poz1,poz2);

[m,p]=size(x);

N=p-1;

y=x;

y(poz1)=x(poz2);

y(poz2)=x(poz1);

y(N+1)=valoare(y(1:N));

end

% selectarea parintilor – cate doi/popuatie

function [parinti]=selectie(pop);

[dim,p]=size(pop);

N=p-1;

parinti=zeros(2,N+1);

p1=unidrnd(dim);

p2=unidrnd(dim);

while(p2==p1)

p2=unidrnd(dim);

end;

parinti(1,:)=pop(p1,:);

parinti(2,:)=pop(p2,:);

end

% trecerea de generatia curenta la generatia urmatoare

function [popNou,evalmax]=trecere(pop,pm);

[parinti]=selectie(pop);

[dim,m]=size(pop);

N=m-1;

poz=unidrnd(N-1);

% aplica operatia de crossover

y=zeros(2,N+1);

[y(1,:),y2(2,:)]=crossover(parinti(1,:),parinti(2,:),poz);

% aplica mutatia

prm=unifrnd(0,1);

if(prm<=pm)

i=unidrnd(2);

poz1=unidrnd(N);

poz2=unidrnd(N);

while(poz2==poz1)

poz2=unidrnd(N);

end;

y(i,:)=mutatie(y(i,:),poz1,poz2);

end;

popN=[pop;y(1,:);y2(2,:)];

tt=sortrows(popN,N+1);

%disp(tt);

popNou=zeros(dim,N+1);

popNou=tt(3:dim+2,:);

evalmax=popNou(dim,N+1);

%disp(popNou);

end

% algoritmul evolutiv

function []=EA\_Queens(N,dim,pm,Max);

%N este umarul de regine, dim este dimensiunea populatiei, pm este

% probabilitatea de efectuare a unei mutatii si Max este numarul

% maxim de iteratii(generatii)

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

evalmax=0;

t=0;

V=[];

while(evalmax<N\*(N-1)/2 && t<Max)

[popN,evalmax]=trecere(pop,pm);

pop=popN;

disp(popN(dim,:));

V=[V popN(dim,N+1)];

t=t+1;

end;

figure

i=1:t;

plot(i,V(i),'ro:');

axis([0 t V(1)-1 (N\*(N-1)/2)+5]);

figure

for i=0:N

plot( [0 N],[i i],'k-');

plot( [i i],[0 N],'k-');

hold on

end;

hold on

for i=1:N

x=i;

y=pop(dim,i);

k=x-1:0.1:x;

l=y-1:0.1:y;

plot(k,l,'bv');

hold on

k=x:-0.1:x-1;

l=y-1:0.1:y;

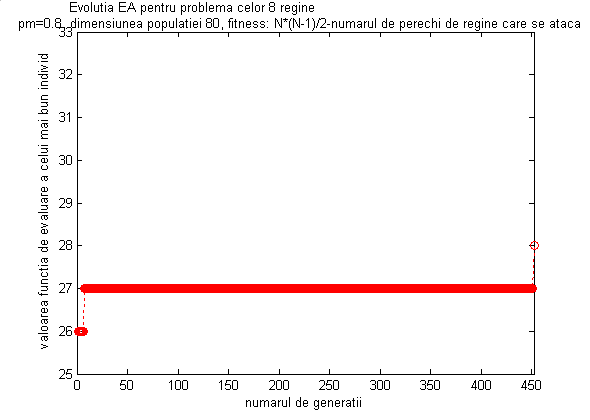
plot(k,l,'bv');

hold on

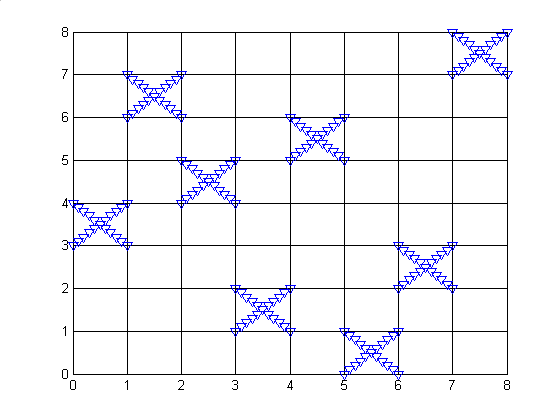
end;

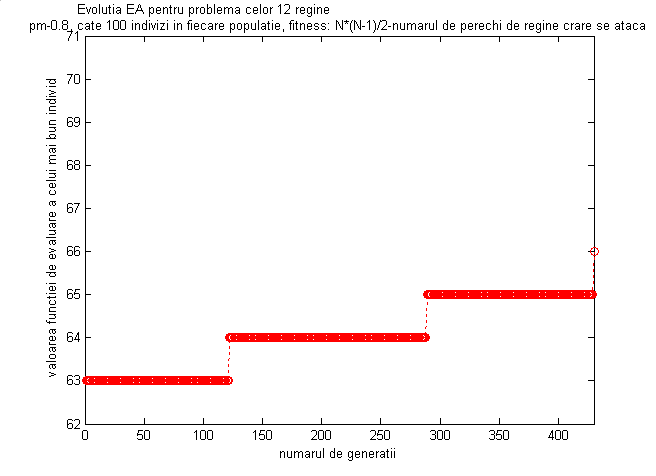
axis([0 N 0 N]);

end



O configuraţie corectă obţinută prin aplicarea algoritmului de mai sus:





O configuraţie corectă obţinută prin aplicarea algoritmului de mai sus:

