-PROIECT PEAG-

ENUNT PROBLEMA

Utilizați un algoritm genetic pentru determinarea unui drum optim într-un graf ponderat conex – exprimată în termeni de distanțe între puncte de interes (orașe, puncte de aprovizionare etc.). Se cunoaște matricea ponderilor grafului și cele două puncte între care trebuie determinat drumul (S – punct de pornire și F – punct final). Valoare unui drum este suma ponderilor arcelor care compun drumul respectiv. Drumul oprim este drumul cu valoarea cea mai mică posibilă.

STRUCTURA GENERALĂ A UNUI ALGORITM GENETIC

Un algoritm genetic este un tip de algoritm evolutiv in care un candidat la solutie e reprezentat prin intermediul unui sir definit pe un alpabet finit.

1. Reprezentarea (definirea membrilor populatiei)
2. Functia de evaluare (de tip fitness)
3. Populatia
4. Mecanismul de selectare a parintilor (indivizii care interschimba material genetic)
5. Operatorii de variatie (recombinarea si mutatia)
6. Mecanismul de selectare a membrilor generatiei urmatoare (actualizarea populatiei)
7. Definirea modulului de initializare (determinarea populatiei initiale)
8. Definirea conditiei terminale.

IPOTEZE

Fie n –numarul de puncte de interes si D matricea ponderilor grafului si a costurilor de deplasare intre cele doua puncte D(i,j), unde i j si i,j [1,n] reprezinta costul trecerii din orasul i ,in orasul j.

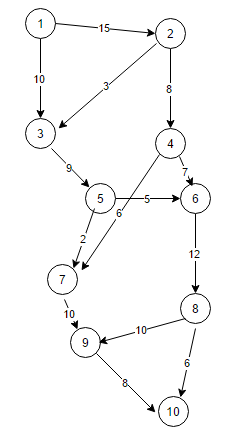
Fie s- punctual de pornire si f-punctul de oprire. Problema este de a se gasi o drumul cu costul cel mai mic prin care se poate ajunge de la s la f.

O solutie a problemei e reprezentata prin intermediul unei permutari x cu n elemente. Costul parcurgerii drumului fiind :

cost(x)=

Pentru a obtine o problema de maxim functia obiectiv care se doreste a fi cat mai mare este:

Rezultat(x)=

ELEMENTELE UNUI GA REGASITE IN PROBLEMA

Fie graful din figura alaturata:

n=10.

s=1; f=10 unde s=punctual de start, f=punctual final de sosire.

* D matricea costurilor este:

0 15 10 0 0 0 0 0 0 0

0 0 3 8 0 0 0 0 0 0

0 0 0 0 9 0 0 0 0 0

0 0 0 0 0 7 6 0 0 0

0 0 0 0 0 5 2 0 0 0

0 0 0 0 0 0 0 12 0 0

0 0 0 0 0 0 0 0 10 0

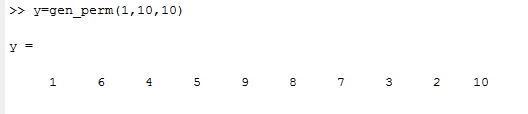
0 0 0 0 0 0 0 0 10 6

0 0 0 0 0 0 0 0 0 8

1. 0 0 0 0 0 0 0 0 0
2. Definirea membrilor populatiei se realizeaza prin intermediul permutarilor.

* permutarile au n elemente si au fixate pe prima,respective ultima pozitie punctul de start,respectiv de final.
* valorile din intervalul [2,n-1] vor fi generate aleator
* o permutare astfel generate reprezinta un potential drum optim

Functia care realizeaza generarea permutarilor este: gen\_perm(s,f,dim)

Pentru exemplul de mai sus se obtine urmatorul rezultat:

1. Functia de evaluare f\_obiectiv(c,d) verifica daca permutarea generata este una valida, si daca da ii calculeaza functia fitness.

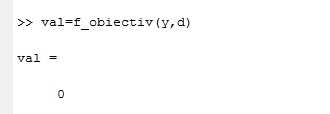
c- cromozomul/permutarea careia I se va calcula calitatea

d-matricea distantelor, care va fi preluata dintr-un vector d=load(‘numefis.txt’), numefis-fiind numele fisierului unde am stocat matricea costurilor

O permutare este valida daca exista un drum in permutarea generata. In cazul in care va exista se va calcula costul acelui drum, iar aceasta valoarea va fi raportata la 1.

Obiectivul este sa avem o valoarea a costului cat mai mica, deci o valoare rezultata cat mai mare.

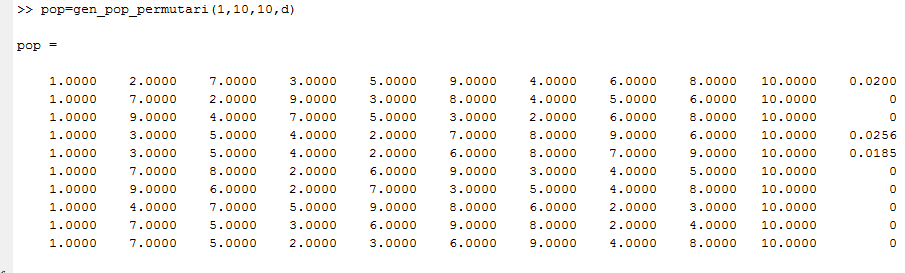
Daca aplicam functia pe permutarea de mai sus se va obtine 0, intrucat nu exista drum de la 1 la 10 in acea permutare.



1. Populatia initiala este o matrice de dimensiune dim x (m+1) unde,

* dim=numarul de indivizi generati , de cromozomi
* m=numarul de gene din fiecare cromozom
* pe pozitia m+1 a fiecarui cromozom va fi memorata calitatea sa evalutata prin functia obiectiv.

Populatia e generate prin functia: gen\_pop\_permutari(s,f,dim,d)



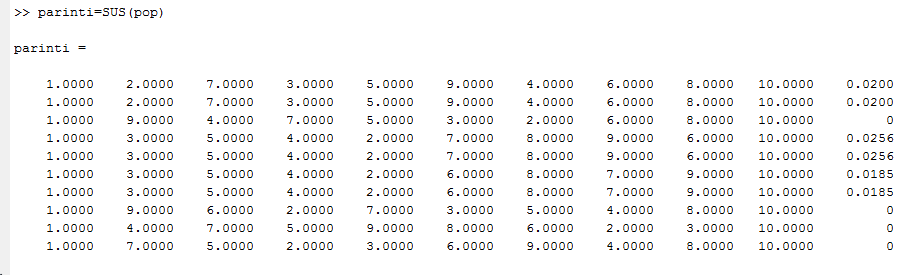
Se observa ca de exemplu prima permutare are ultimul element nenul,ceea ce inseamna ca exista drum in ea.Luand fiecare element in parte lantul rezultat este: 1,2,3,5,6,8,10.

Costul sau este: D(1,2)+D(2,3)+D(3,5)+D(5,6)+D(6,8)+D(8,10)=15+3+9+5+12+6=50

Valoare functiei obiectiv pentru primul cromozom este : 1/50=0.0200

1. Selectarea parintilor se realizeaza prin algoritmul SUS(Stochastic Universal Sampling)

* functia SUS(Pop) impementeaza acest algoritm, urmand ca la aplicarea lui indivizii cu FPS-ul cat mai mare sa supravietuiasca si prin urmare sa faca parte din indivizii care interschimba material genetic.



1. Operatorii de variatie

Recombinarea:

- prin recombinarea a doi sau mai multi parinti se realizeaza noi solutii candidat.

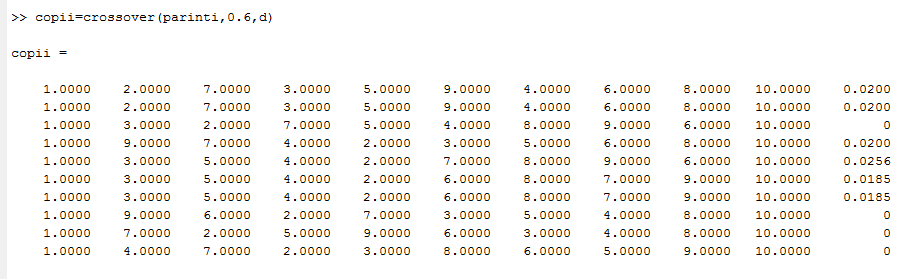
- acesta e aplicat probabilist , cu o rata pc( ce semnifica sansa ca perechea de parinti selectata sa fie supusa recombinarii) care apartine intervalului [0.5,1]. In exemplul rulat s-a utilizat valoarea 0.6.

- in acest process se selecteaza doua solutii candidat parinte , apoi se genereaza aleator un numar din [0,1) care se compara cu pc

- daca r<=pc indivizii noi sunt create prin recombinarea celor doi parinti , altfel sunr create prin duplicarea parintilor.

In agoritm a fost utilizat operatorul PMX(Partially Mapped Crossover) , specific recombinarii in reprezentarea prin permutari.

copii=crossover(parinti,0.6,d)



Mutatia:

A fost aplicata mutatia prin inserare, care presupune:

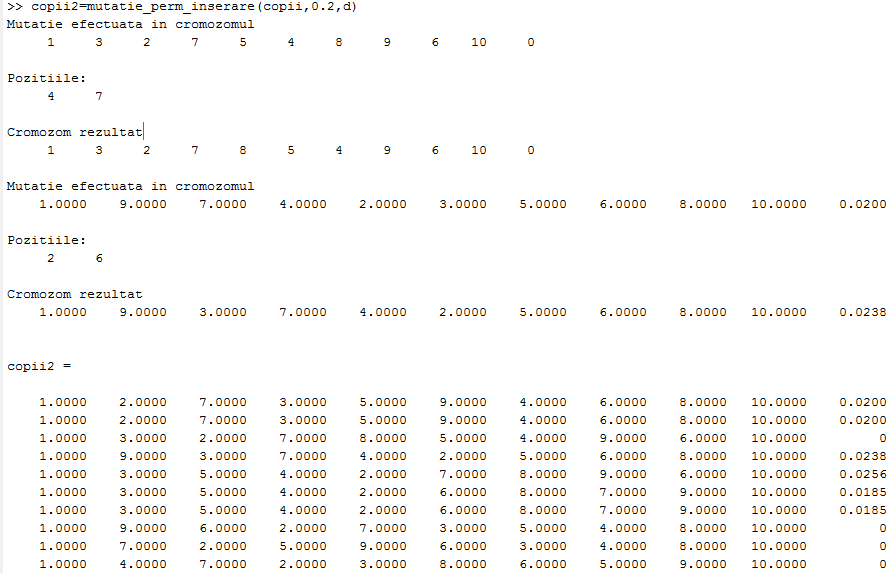
- selectarea aleatoare a 2 gene 1<=i<=j<=n

-memorarea alelei din gena j: A

- deplasarea cu o pozitie catre dreapta a valorilor genelor i+1,i+2….j si plasarea valorii A pe pozitia i+1

Pm=0.2 este rata de mutatie ce corespunde probabilitatii modificarii unui genotip.

Copii rezultati in urma mutatiei:

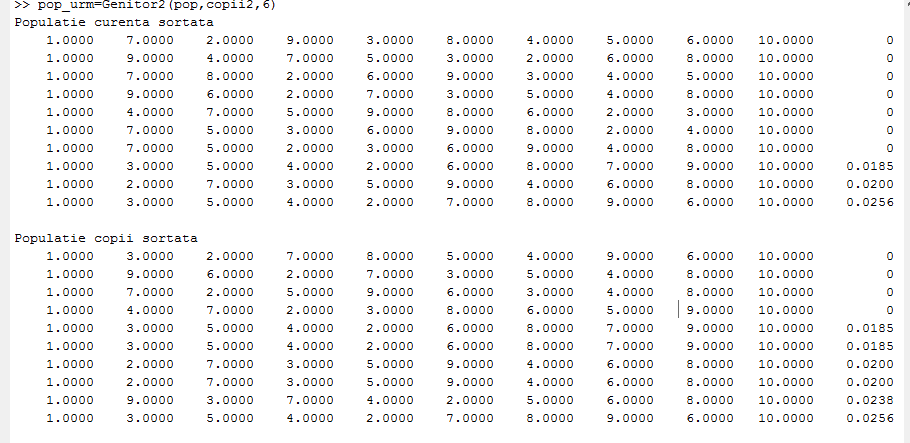


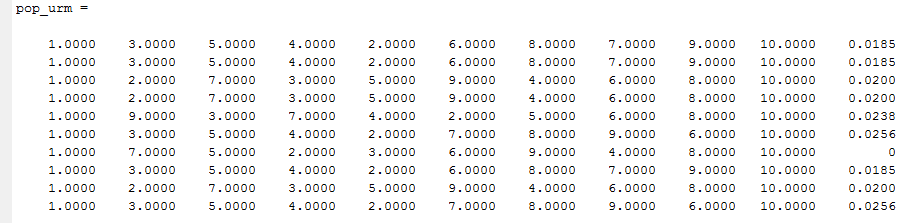
1. Selectia generatiei urmatoare

-se realizeaza prin mecanismul Genitor

- se lucreaza pe populatia curenta si pe copii

-ambele sunt sortate dupa valorile functiei fitness si sunt inlocuiti cei mai slabi “n” indivizi din populatia curenta cu “n” copii (cu functia obiectiv cea mai mare)



Populatia urmatoare:

1. Toate functiile de mai sunt sunt integrate in functia GA determinandu-se populatia initiala.
2. Conditia de terminare a intregului algoritm este atingerea unui numar maxim “nmax” de iteratii / generate de populatii.

GA(s,f,dim,d,pc,pm,nmax,nr)

s-start ; f-final

dim-nr de cromozomi

d-matricea costurilor

pc-rata de combinare ; pm-rata de mutate

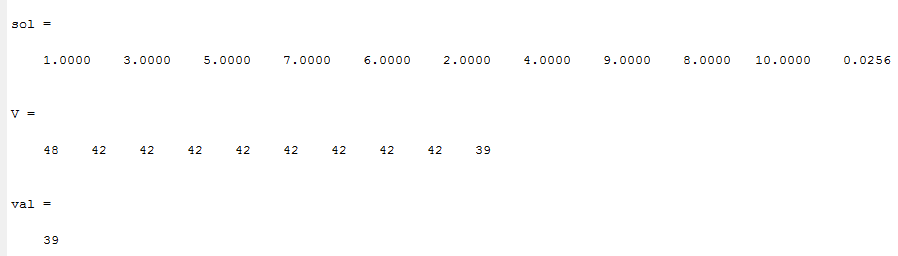
nmax-numarul maxim de iteratii/generatii

nr-nr de cromozomi inlocuiti in algoritmul Genitor .

In urma executarii rezulta:

d=load('fis.txt')

sol=GA(1,10,10,d,0.6,0.1,2,5)

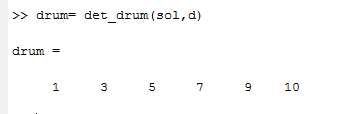


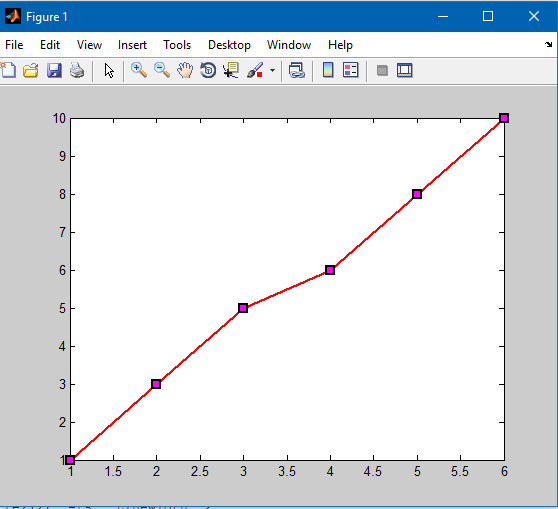
Sol-permutarea in care exista drumul optim si val-valoarea costului solutiei optime

V-vectorul cu cele mai bune solutii din fiecare iteratie

Aplicand functia det\_drum(sol,d) se va obtine exact drumul:

>> det\_drum(sol,d)





Sursa dezvoltata in Matlab:

function [y]=gen\_perm(s,f,m)

y = zeros(1,m);

y(1)=s;

y(m)=f;

for i=2:m-1

gata=0;

while(~gata)

x=unidrnd(m);

if(~ismember(x,y))

y(i)=x;

gata=1;

end;

end;

end;

end

function [val] =f\_obiectiv(c,D)

[~,n]=size(c);

val=0;

i=1;

for j=i+1:n

if( D(c(i),c(j)))

val=val+D(c(i),c(j));

i=j;

end;

end;

if(val && i==n)

val =1/val;

else

val=0;

end;

end

function [ Pop ] = gen\_pop\_permutari(s,f,dim,D)

[~,n]=size(D);

Pop=zeros(dim,n+1);

for i=1:dim

Pop(i,1:n)=gen\_perm(s,f,n);

Pop(i,n+1)=f\_obiectiv(Pop(i,1:n),D);

end;

end

function [Parinti] =SUS(Pop)

[dim,m]=size(Pop);

n=m-1;

c=2;

medie=mean(Pop(:,n+1));

sigma=std(Pop(:,n+1));

val=medie-c\*sigma;

g=zeros(1,dim);

for i=1:dim

g(i)=max([Pop(i,n+1)-val 0]);

end;

s=sum(g);

p=g/s;

q=zeros(1,dim);

for i=1:dim

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

end;

Parinti=zeros(dim,m);

r=unifrnd(0,1/dim);

i=1;j=1;

while (i<=dim)

while(r<=q(j))

Parinti(i,:)=Pop(j,:);

i=i+1;

r=r+1/dim;

end;

j=j+1;

end;

end

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

n=length(x1);

gen=0;

while(~gen)

poz=unidrnd(n,[1 2]);

p1=min(poz);p2=max(poz);

if(p1<p2)

gen=1;

end;

end;

c1=c\_PMX(x1,y1,p1,p2);

c2=c\_PMX(y1,x1,p1,p2);

end

function [c]=c\_PMX(x1,y1,p1,p2)

n=length(x1);

c=zeros(1,n);

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

A=[];

for i=p1:p2

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

A=[A; [y1(i) i]];

end;

end;

[p,~]=size(A);

for i=1:p

va=A(i,1);pa=A(i,2);

b=c(pa);

[~,j]=ismember(b,y1);

while(c(j))

b=c(j);

[~,j]=ismember(b,y1);

end;

c(j)=va;

end;

for i=n:-1:1

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

[~,j]=ismember(0,c);

c(j)=y1(i);

end;

end;

end

function [Copii] = crossover(Pop,pc,D)

[dim,m]=size(Pop);

Copii=Pop;

n=m-1;

for t=1:dim/2

r=unifrnd(0,1);

if(r<pc)

[x,y]=PMX(Pop(2\*t-1,1:n),Pop(2\*t,1:n));

Copii(2\*t-1,1:n)=x;

Copii(2\*t-1,n+1)=f\_obiectiv(x,D);

Copii(2\*t,1:n)=y;

Copii(2\*t,n+1)=f\_obiectiv(y,D);

end;

end;

end

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

[dim,n]=size(pop);

popN=pop;

for i=1:dim

r=unifrnd(0,1);

if(r<pm)

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

p=zeros(1,2);

p(1)=1+unidrnd(n-3);

p(2)=1+unidrnd(n-3);

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

p(2)=1+unidrnd(n-2);

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

popN(i,n)=f\_obiectiv(popN(i,1:n-1),d);

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

end;

end;

end

function [Pop\_urm] = Genitor2(Pop\_curenta,Copii,ni)

[dim,m]=size(Pop\_curenta);

Pop=sortrows(Pop\_curenta,m);

disp('Populatie curenta sortata');

disp(Pop);

PopC=sortrows(Copii,m);

disp('Populatie copii sortata');

disp(PopC);

Pop\_urm=Pop;

Pop\_urm(1:ni,:)=PopC(dim-ni+1:dim,:);

end

function [sol,V,val]=GA(s,f,dim,d,pc,pm,nmax,nr)

[~,n]=size(d);

pop = gen\_pop\_permutari(s,f,dim,d)

t = 0;

V = zeros(1,nmax);

while (t<nmax)

parinti=SUS(pop);

desc=crossover(parinti,pc,d);

desc=mutatie\_perm\_inserare(desc,pm,d);

pop=Genitor2(pop,desc,nr);

t=t+1;

[vmax,poz]=max(pop(:,n+1));

sol=pop(poz,:);

V(t)=1/vmax;

val=min(V);

end;

end

function det\_drum(sol,d)

n=length(sol);

drum=zeros(1,n-1);

drum(1)=sol(1);

i=1; k=2;

for j=i+1:n-1

if( d(sol(i),sol(j)))

drum(k)=sol(j);

k=k+1;

i=j;

end;

end;

rez=zeros(1,k-1);

for i=1:k-1

rez(i)=drum(i);

end;

disp('Drumul final este: ');

disp(rez);

figure

z=1:k-1;

plot(z,rez(z),'-rs','LineWidth',2,...

'MarkerEdgeColor','k',...

'MarkerFaceColor','m',...

'MarkerSize',8);

end