**MUTARE PENTRU REPREZENTARE BINARA:**

function [ y ] = m\_binara( x, m )

%op de mutare pt reprez binara

%I : x-individ curent, pm-probabilitatea de mutatie

%E : y-individul mutant rezultat

[~,n]=size(x);

y=x;

for i=1:n

r=unifrnd(0,1);

if r<pm

y(i)=~r(n);

end;

end;

end

**FLUAJ NUMAR INTREG:**

function [ y ] = m\_fluaj\_i( x, pm, a, b, d)

% fluaj nr intreg

% I : x-individ curent, pm-probabilitatea de mutatie, a,b-capete, d-dispersie

% E : y-ind mutant rezultat

[~,n]=size(x);

y=x;

for i=1:n

max=unidrnd(0,1);

if r<pm

y(i)=x(i)+fix(normrnd(0,d));

if y(i)<a

y(i)=a;

end;

if y(i)>b

y(i)=b;

end;

end;

end;

end

**FLUAJ NUMAR REAL:**

function [ y ] = m\_fluaj\_r( x, pm, a, b, d)

% fluaj nr reale

% I : x-individ curent, pm-probabilitatea de mutatie, a,b-capete, d-dispersie

% E : y-ind mutant rezultat

[~,n]=size(x);

y=x;

for i=1:n

r=unifrnd(0,1);

if r<pm

y(i)=x(i)+normrnd(0,d);

if y(i)<a

y(i)=a;

end;

if y(i)>b

y(i)=b;

end;

end;

end;

end

**MUTATIE PRIN INTERSCHIMBARE:**

function [ y ] = m\_interschimbare( x, pm )

%mutatie prin interschimbare

%I : x-individ curent, pm-probabilitatea de mutatie

%E : y-individul mutant rezultat

[~,n]=size(x);

y=x;

r=unifrnd(0,1);

if r<pm

p1=unidrnd(n-1);

p2=unidrnd(n-p1)+p1;

y(p1)=x(p2);

y(p2)=x(p1);

end;

end

**MUTATIE PRIN INVERSARE:**

function [ y ] = m\_inversare( x, pm )

%mutatie prin inversare

%I : x-individ curent, pm-probabilitatea de mutatie

%E : y-individul mutant rezultat

[~,n]=size(x);

y=x;

r=unifrnd(0,1);

%unifrnd-genereaza o valoare reala intre 0 si 1

if r<pm

p1=unidrnd(n-1);

%unidrnd-genereaza o valoare intreaga intre 1 si valoarea pe care o primeste

p2=unidrnd(n-p1)+p1;

y(1,p1:p2)=x(1,p2:-1:p1);

disp(p1);

disp(p2);

end;

end

**GENERARE PERMUTARE:**

function [ p ] = permutare( n )

% generare permutare

% i : n - dim permutare, E : p - permutare

p=zeros(1,n);

for i=1:n

gata=0;

while ~ gata

x=unidrnd(n);

if ~ismember(x,p)

p(i)=x;

gata=1;

end;

end;

end;

end

**ARITMETICA TOTALA:**

function [ a,b ] = r\_aritmetica\_totala( x,y,p,w )

% I : x,y-parinti, p-probabilitatea, w-ponderea

% E : a,b-descendenti

a=x;

b=y;

%verificam pt fiecare pereche de gene probabilitatea de recombinare

d=length(x);

for i=1:d

%generam o val aleatoare intre 0 si 1

r=unifrnd(0,1);

if r<p

%primul descendent primeste valoarea...

a(i)=x(i)\*w+y(i)\*(1-w);

b(i)=y(i)\*w+x(i)\*(1-w);

end;

end;

end

**MULTIPUNCT:**

function [ a,b ] = r\_multipunct( x,y,p,n )

%I : x,y - parinti, p - probabilitate de recombinare, n - nr de puncte de

%incrucisare

%E : a,b - descendenti

%clonam parintii

a=x;

b=y;

%verificam daca ne incadram in probabilitate-generam o valoare aleatoare r

r=unifrnd(0,1);

if r<p

%umplem vectorul poz cu 0 (vector de pozitii)

poz=zeros(1,n);

%dimensiunea unui cromozom-length(cea mai mare dimensiune din masiv)

d=length(x);

%generam n pozitii

for i=1:n

t=unifrnd(d);

%verificam daca este membru si daca este, generam alta valoare

while is member(t,poz)%t-element,poz-vector

t=unidrnd(d);

end;

poz(i)=t;

end;

%sortare

poz=sort(poz);

poz=[poz,d];

for i=1:2:n

a(poz[i]:poz[i+1])=y(poz[i]:poz[i+1]);

b(poz[i]:poz[i+1])=x(poz[i]:poz[i+1]);

end;

end;

end

**OCX:**

function [ a,b ] = r\_ocx( x,y,p )

%I : x,y-parinti ; p-probabilitatea de crossover

%E : a,b-descendenti

a=x;

b=y;

r=unifrnd(0,1);

if r<p

n=length(x);

p1=unidrnd(n-1)+1;

p2=unidrnd(p2-1);

a=r\_ocx\_copil(x,y,p1,p2);

b=r\_ocx\_copil(y,x,p1,p2);

end;

end

function [ d ] = r\_ocx\_copil( x,y,p1,p2 )

n=length(x);

d=zeros(1,n);

d(p1:p2)=x(p1:p2)

for i=p2+1:n

if ~is member(y(i),d)

d(poz)=y(i);

poz=poz+1;

if poz>n

poz=1;

end;

end;

end;

for i=1:p1-1

if ~is member(y(i),d)

d(poz)=y(i);

poz=poz+1;

end;

end’;

end

**RULETA:**

function [ parinti ] =s\_p\_fps\_ruleta( pop )

%UNTITLED2 Summary of this function goes here

% Detailed explanation goes here

[m,n] = size(pop);

p = zeros(1,m);

q = zeros(1,m);

parinti = zeros(m,n);

s = sum(pop(:,n));

for i = 1:m

p(i) = pop(1,n) / s;

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

end;

disp p;

disp q;

for i = 1:m

r = unifrnd(0,1);

j = 1;

while(r>q(j) && j<1)

j = j+1;

end;

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

end;

end

**RULETA SUS:**

function [ parinti ] =s\_p\_fps\_sus( pop )

%UNTITLED2 Summary of this function goes here

% Detailed explanation goes here

[m,n] = size(pop);

p = zeros(1,m);

q = zeros(1,m);

parinti = zeros(m,n);

s = sum(pop(:,n));

for i = 1:m

p(i) = pop(1,n) / s;

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

end;

disp p;

disp q;

r = unifrnd(0,1/m);

i=1; %i-parinti

k=1;

while(i<=m)

while(r<q(k))

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

r=r+1/m;

i=i+1; %am adaugat un parinte

end;

k=k+1;

end;

end

**TURNIR:**

function [ parinti ] = s\_p\_turnir( pop,k )

%selectie parinti prin metoda turnir

% i: pop curenta-pop si dimensiunea turnirului-k

% o: parinti

[m,n]=size(pop)

parinti=zeros(m,n);

turneu=zeros(k,n);

for i=1:m

for j=1:k

r=unidrnd(m);

turneu(j,:)=pop(r,:);

end;

[~,poz]=max(turneu(:,n));

parinti(i,:)=turneu(poz,:);

enD

end

**GENERARE POPULATIE PERMUTARI:**

function [pop]=gen\_pop\_perm(dim,n)

% generarea unei populatiei initiale de permutari

% I: dim - nr. indivizi, n - dim. permutare

% E: populatia aleatoare

pop=zeros(dim,n+1);

for i=1:dim

pop(i,1:n)=gen\_permutare(n);

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

end;

end

**INTERSCHIMBARE:**

function [y]=m\_interschimbare(x,pm)

% operatorul de mutatie prin interschimbare pe permutari

% I: x - individul (permutarea), pm - probabilitatea de mutatie

% E: y - individul modificat

y=x;

r=unifrnd(0,1);

if r<pm

n=length(x);

poz=unidrnd(n,1,2);

while poz(1)==poz(2)

poz(2)=unidrnd(n);

end;

poz=sort(poz);

y(poz(1))=x(poz(2));

y(poz(2))=x(poz(1));

end;

end

**RECOMBINARE ORDER CROSSOVER:**

function [a,b]=r\_OCX(x,y,pr)

% Operatorul de recombinare Order Crossover

% I: x,y - cromozomii parinti, pr - probabilitatea de recombinare

% E: a, b - descendenti

a=x;

b=y;

r=unifrnd(0,1);

if r<pr

[~,m]=size(x);

p=[0 0];

p(1)=unidrnd(m);

p(2)=unidrnd(m);

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

p(2)=unidrnd(m);

end;

p=sort(p);

a=OCX(x,y,p);

b=OCX(y,x,p);

end;

end

function d=OCX(x,y,p)

% Operatorul de incrucisare OCX, pentru un singur descendent

% pentru al doilea descendent se apeleaza din nou, cu parintii y, x

% I: x, y - cromozomii parinti, cele doua pozitii (in vectorul p)

% E: d - primul cromozom descendent

[~,m]=size(x);

d=zeros(1,m);

d(p(1):p(2)) = x(p(1):p(2));

unde=p(2)+1;

for i=[p(2)+1:m 1:p(2)]

if ~ismember(y(i),d)

if unde>m

unde=1;

end;

d(unde)=y(i);

unde=unde+1;

end;

end;

end

**SELECTAREA ELITISTA A GENERATIEI NOI:**

function [Generatie]=s\_gen\_elitist(pop, desc)

% selectarea elitista a generatiei noi

% I: pop - populatia curenta, desc - descendentii generati

% pe ultima coloana se afla valoarea functiei obiectiv

% E: Generatie - generatia noua

Generatie=desc;

[dim,n]=size(pop);

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

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

if max1>max2

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

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

end;

end

**SELECTIA PARINTILOR DE TIP SUS CU PROBABILITATI FPS STANDARD**:

function [parinti]=s\_p\_FPS\_SUS(pop)

% selectia parintilor tip SUS cu probabilitati FPS standard

% I: pop - populatia curenta, pe ultima coloana e val. fct. obiectiv

% E: parinti - parintii selectati

[m,n]=size(pop);

p=zeros(1,m);

p(1:m)=pop(1:m,n);

s=sum(p);

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

q=zeros(1,m);

for i=1:m

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

end;

parinti=zeros(m,n);

i=1;

k=1;

r=unifrnd(0,1/m);

while(k<=m)

while(r<=q(i))

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

r=r+1/m;

k=k+1;

end;

i=i+1;

end;

end

**CALCULAREA DESCENDENTILOR DIN POPULATIA CURENTA:**

function [desc]=recombinare(parinti,pr,dist)

% calcularea descendentilor din populatia curenta

% I: parinti - multisetul parintilor, pr - probabilitatea de

% recombinare, dist - matricea distantelor

% E: desc - multisetul descendentilor

[dim,n]=size(parinti);

desc=zeros(dim,n);

% selectia aleatoare a perechilor de parinti = o permutare

% perechi=gen\_permutare(dim);

for i=1:2:dim

x=parinti(i,1:n-1);

y=parinti(i+1,1:n-1);

[d1,d2]=r\_PMX(x,y,pr);

desc(i,1:n-1)=d1; desc(i,n)=f\_obiectiv(d1,dist);

desc(i+1,1:n-1)=d2; desc(i+1,n)=f\_obiectiv(d2,dist);

end;

end

**OPERATORUL DE RECOMBINARE PARTIALLY MAPPED CROSSOVER PMX:**

function [a,b]=r\_PMX(x,y,pr)

% operatorul de recombinare Partially Mapped Crossover

% I: x,y - cromozomii parinti, probabilitatea de recombinare

% E: a,b - descendenti

a=x;

b=y;

r=unifrnd(0,1);

if r<pr

[~,m]=size(x);

p=[0 0];

p(1)=unidrnd(m);

p(2)=unidrnd(m);

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

p(2)=unidrnd(m);

end;

p=sort(p);

a=PMX(x,y,p);

b=PMX(y,x,p);

end;

end

function d=PMX(x,y,p)

% Operatorul de incrucisare PMX, pentru un singur descendent

% pentru al doilea descendent se apeleaza din nou, cu parintii y, x

% I: x, y - cromozomii parinti, cele doua pozitii (in vectorul p)

% E: d - primul cromozom descendent

[~,m]=size(x);

d=zeros(1,m);

d(p(1):p(2)) = x(p(1):p(2));

for k=p(1):p(2)

if ~ismember(y(k),d)

i = find(y==d(k));

while d(i)~=0

i = find(y==d(i));

end;

d(i)=y(k);

end;

end;

for i=1:m

if ~ismember(y(i),d)

j = find(d==0);

d(j(1))=y(i);

end;

end;

end

**GENERAREA POPULATIEI INITIALE DE PERMUTARI PE MULTIMEA M:**

function Pop=gen\_pop\_perm(dim,m)

%generarea populatiei initiale de permutari pe multimea m

%I: dim - nr. indivizi, m - dimensiune individ (permutare)

%E: pop - populatia

Pop=zeros(dim,m);

for i=1:dim

Pop(i,1:m)=gen\_perm(m);

end;

end

**GENERARE POPULATIE INITIALA NUMERE REALE:**

function [popi] = gen\_ini\_n\_reale(a,b,dim)

%genereaza o populatie initiala din [a1,b1]x...x[an,bn]

% I: dim - dim. populatie; a - capetele din st.; b - capetele din dr.

[~,n]=size(a);

popi=zeros(dim,n);

for k=1:dim

popi(k,:)=unifrnd(a,b,1,n);

end;

end

**GENERATIE POPULATIE CU INDIVIZI REPREZENTATI CA SIRURI BINARE:**

function [pop]=gen\_ini\_binar(m,dim)

% genereaza populatie cu indivizi reprezentati ca siruri binare

% I: dim - dimensiune populatie, m - dimensiune individ

% E: pop - populatia

pop=zeros(dim,m);

for i=1:dim

x=unidrnd(2^m-1);

pop(i,1:m)=bitget(x,m:-1:1);

end;

end

**RESETARE ALEATOARE SAU UNIFORMA:**

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: unidrnd(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