**Algoritmi metaeuristici.**

**Lab 6: Optimizare multicriterială**

---

## 1. Optimizare multicriterială.

Optimizarea multicriterială are ca scop optimizarea simultană a mai multor criterii. Funcția obiectiv este de forma $F:R^n->R^r$, iar componentele sunt $F=(f_1,f_2,\ldots,f_r)$.

Criteriile de optimizat sunt de regulă conflictuale, astfel că nu există o soluție unică a problemei. In aceste condiții se caută soluții în sens Pareto, care se caracterizează prin faptul că nu există alte configurații care să fie mai bune în raport cu toate criteriile de optimizat (orice îmbunătățire în raport cu unul dintre criterii conduce la o înrăutățire în raport cu un alt criteriu). Astfel de soluții se numesc nedominate (în sensul că nu există alte soluții care să le domine în raport cu fiecare dintre criteriile de optimizat).

Există mai multe abordări ale unei astfel de probleme. Principalele categorii de metode sunt:

- *Metode bazate pe tehnica agregării:* se transformă problema inițială de optimizare multicriterială într-una de optimizare uni-criterială prin combinarea criteriilor. Noua funcție de optimizat este $f(x)=w_1f_1(x)+w_2f_2(x)+\ldots+w_rf_r(x)$ unde $w_1,w_2,\ldots,w_r$ sunt ponderi asociate criteriilor. Pentru fiecare set de ponderi se poate obține o altă soluție.
- *Aproximarea directă a mulțimii Pareto optimale:* se utilizează o populație de elemente care vor aproxima mulțimea Pareto optimală (multimea tuturor soluțiilor nedominate).

Exemple de funcții test utilizate pentru a analiza performanțele algoritmilor de optimizare multicriterială sunt disponibile la: http://en.wikipedia.org/wiki/Test_functions_for_optimization sau la http://people.ee.ethz.ch/~sop/download/supplementary/testproblems/

**Aplicație 1.** Se consideră funcția $F:[0,4]->RxR$, $F(x)=((x-1)^2,(x-2)^2)$. Să se aproximeze mulțimea Pareto optimală și frontul Pareto corespunzător (frontul Pareto reprezintă mulțimea valorilor funcțiilor obiectiv corespunzătoare elementelor din mulțimea Pareto optimală).

*Varianta 1.* Se aplică tehnica agregării

a) se construiește funcția obiectiv:

```
function y=fw(x)
    w=0.1;
    y1=(x-1)*(x-1);
    y2=(x-2)*(x-2);
    y=w*y1+(1-w)*y2;
endfunction
```

b) se aplică succesiv o strategie evolutivă (de exemplu cea descrisă în SE.sci – vezi lab3) pentru optimizarea functiei obiectiv pentru următoarele valori ale ponderii w: (0.1,0.2,0.3,…,0.9) și se stochează valoarea într-un vector x

c) se vizualizează valorile funcțiilor obiectiv pentru valorile stocate la pasul anterior (va reprezenta o aproximare a frontului Pareto):

```
function pareto(x)
    f1=(x-1).^2;
    f2=(x-2).^2;
    plot(f1,f2,'*');
```

```
endfunction
```

*Varianta 2.* Se folosește implementarea algoritmului NSGA-II (funcția optim_nsga2 în Scilab) sau a algoritmului MOGA (funcția optim_moga în Scilab).

Indicație: vezi exNSGA.sci

**Exercitiu:** Să se analizeze comportamentul algoritmilor MOGA și NSGA2 pentru funcțiile de test ZDT1, ZDT3 (de la http://people.ee.ethz.ch/~sop/download/supplementary/testproblems/)

*Implementarea rețelelor RBF – funcții SciLab pentru crearea și antrenarea unei rețele cu un nivel ascuns*

Crearea unei rețele RBF

```
function network=RBFcreate(N, K, M, sigma)
  network=tlist(["RBF NN", "N", "K", "M", "X0", "Y0", "X1", "Y1", "X2", "Y2", "C", "W", "f", "sigma"],
                N, K, M, zeros(N,1), zeros(N,1), zeros(K,1), zeros(K+1,1), zeros(M,1), zeros(M,1),
                zeros(K,N), zeros(M,K+1), gaussian,sigma);
endfunction
```

Funcție de activare:

```
function output=gaussian(x)
  output=exp(-x^2/2);
endfunction
```

Funcție de agregare:

```
function d=dist(x, y)
  d=sqrt((x-y)'*(x-y));
endfunction
```

Calcul semnal de ieșire:

```
function network=forward(network, X)
 network.X0=X; network.Y0=network.X0;
 for k=1:network.K
   network.X1(k)=dist(network.C(k)',network.Y0);
 end
 network.Y1=[-1; feval(network.X1./network.sigma,network.f)];
 network.X2=network.W*network.Y1;
 network.Y2=network.X2;
endfunction
```

Calcul funcție de eroare:

```
function err=error_computation(network, tset)
err=0;
[N,L]=size(tset.X);
for i=1:L
    network=forward(network,tset.X(:,i));
    delta=(tset.d(i)-network.Y2).^2;
    err=err+sum(delta);
end
err=err/L
endfunction
```

Algoritm de antrenare (centrii coincid cu datele din setul de antrenare)

```
function [network, ap, aE]=train(network, tset, pmax, Emax, eta)
L=tset.L;
network.C=tset.X';
network.W=2*rand(network.W)-ones(network.W);
E=error_computation(network,tset);
p=0;
ap=[];aE=[];
while p<pmax & E>Emax
  E=error_computation(network,tset);
  for i=1:L
     network=forward(network,tset.X(:,i));
     y=network.W*network.Y1;
     delta=tset.d(i)-y;
     network.W=network.W+eta*(delta*network.Y1')
  end;
  E=error_computation(network,tset);
  if (modulo(p,10)==0) then
      disp(p,"Iteration:");
      disp(E,"error=");
      ap=[ap p]; aE=[aE E];
  end
  p=p+1;
 end
 disp(E,"error=");
endfunction
```

**Exemplu:** regresie neliniară (fisier RBFnetwork.sci)

Construire set de antrenare:

```
function tset=trainingSet()
   tset=tlist(["Training set","L","X","d"],0,zeros(1,100),zeros(1,100));
   tset.X=[0:1:10*%pi];
   tset.L=length(tset.X);
   tset.d=sin(tset.X)+0.2*rand(tset.X)-0.1*ones(tset.X);
endfunction
```

Creare rețea, antrenare, vizualizare funcție aproximată, vizualizare eroare:

```
function tset=regressionRBF(eta, epochs, sigma)
tset=trainingSet();          // construirea setului de antrenare
rbf=RBFcreate(1,length(tset.X),1,sigma); // crearea retelei
disp(tset,"tset=");
[rbfa,ap,aE]=train(rbf,tset,epochs,0.0001,eta); // antrenarea retelei
// testarea retelei
test_plot(tset,rbfa);
pause;    // pauza in executie;  se continua tastand <resume>
clf;
plot(ap,aE);
endfunction
```

**Temă:**

Să se modifice algoritmul de antrenare al unei rețele RBF astfel încât numărul de centri și vectorii prototip să fie stabiliți în mod dinamic (algoritm de antrenare incrementală – curs 11 slide 62).