Calcul neuronal si evolutiv.

Lab 7: Optimizare multicriterială. Metaeuristici inspirate de natură: Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO)

---

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

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 aplica succesiv o strategie evolutivă (de exemplu cea descrisă în SE.sci) 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 foloseşte implementarea algoritmului NSGA-II (funcţia optim_nsga2 în Scilab) sau a algoritmului MOGA (funcţia optim_moga în Scilab).

Indicaţie: vezi exNSGA.sci

2. Modelul coloniei de furnici: Ant Colony Optimization (ACO)

ACO este o metaeuristică inspirată de comportamentul coloniilor de furnici. Este folosită în special în rezolvarea problemelor de optimizare combinatorială (identificare trasee optime, planificare). Ideea de bază este de a utiliza o populaţie de furnici artificiale (agenţi). Fiecare dintre aceşti agenţi construieşte la fiecare generaţie câte o soluţie. Completarea componentelor soluţiilor se realizează în manieră probabilistă iar probabilităţile de selecţie a uneia sau alteia dintre valorile posibile se calculează folosind atât informaţii de natură locală privind problema de rezolvat (ceea ce observă furnica) cât şi informaţii de natură globală (obţinute prin comunicarea indirectă dintre furnici prin intermediul feromonilor).

Rezolvarea problemei comis voiajorului folosind ACO. Datele de intrare ale problemei corespund grafului asociat drumurilor existente între cele n oraşe şi a costurilor acestor drumuri (matricea de costuri).

Se foloseşte o populaţie de *m* furnici plasate iniţial aleator în noduri (se poate considera că toate furnicile sunt iniţial plasate în acelaşi nod – de exemplu în primul nod). La fiecare generaţie, fiecare furnică efectuează *n* deplasări între oraşe. Furnicile memorează oraşele vizitate astfel încât să nu treacă de două ori prin acelaşi oraş. Trecerea furnicii *k* din oraşul *i* în oraşul *j*, la etapa *t*, se bazează pe distribuţia de probabilitate:

$$P_t^k(i,j) = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l\in N(i,k)}\tau_{il}^{\alpha}\eta_{il}^{\beta}} & j \text{ nu a fost vizitat} \\ 0 & j \text{ a fost vizitat} \end{cases}$$

Factorii ce intervin în calculul probabilităţii au următoarea semnificaţie:

- τ_{ij} modelează cantitatea de feromoni depusă de furnici pe arcul *(i,j)*; se iniţializează aleator şi după fiecare generaţie este actualizat de către fiecare furnică k care a parcurs arcul respectiv, după regula:

  $$\tau_{ij} = (1-\rho)\tau_{ij} + Q_{ij} / \cos t(T_k)$$

  - ρ este un coeficient subunitar care controlează evaporarea feromonului, Q_{ij} este 0 daca (i,j) nu aparţine traseului parcurs de furnica k şi este diferit de 0 în caz contrar (în cazul cel mai simplu Q=1).
  - cost(T_k) reprezintă costul traseului.

  Obs: o variantă a acestui tip de ajustare este cea în care se realizează modificarea folosind doar cel mai bun traseu.
- η_{ij} modelează informaţia locală privind calitatea arcului; cea mai simplă variantă este când η_{ij} este invers proporţional cu costul arcului (i,j)
- α si β sunt parametri care controlează ponderea relativă a celor două tipuri de informaţii: cea furnizată de concentraţia de feromoni şi cea bazată pe informaţia locală privind calitatea arcului.

- N(i,k) reprezintă lista vecinilor nodului i care nu au fost încă vizitați de către furnica k

Aplicație 2. Să se implementeze varianta descrisă a algoritmului ACO și să se testeze pentru problema comis-voiajorului
Indicatie. Vezi funcția ACO_TSP.sci

Exercițiul 1. Să se modifice implementarea anterioară astfel încât la ajustarea elementelor matricii cu valori ale feromonilor (tau) să se utilizeze traseele descoperite de către toate furnicile.

Indicație. Termenii de ajustare corespunzători elementelor matricii de feromoni se cumulează pe măsură ce se construiesc traseele.

3. Modelul ansamblului de particule: Particle Swarm Optimization (PSO)

PSO este o metaeuristică utilizată în special pentru optimizarea funcțiilor continue. Se utilizează o populație de m "particule", fiecare particulă *i* fiind caracterizată prin poziția sa (x_i) și viteza de deplasare (v_i). In plus fiecare particulă reține cea mai bună poziție pe care a vizitat-o ($xbest_i$) iar variabila *best* reține cea mai bună poziție vizitată de către populație. Procesul evolutiv constă în modificarea, la fiecare generație *t*, a pozițiilor tuturor particulelor din populație în conformitate cu următoarele reguli:

$$v_i(t+1) = \gamma(v_i(t) + r_1 rand(0,1)(xbest_i - x_i(t)) + r_2 rand(0,1)(best - x_i(t)))$$
$$x_i(t+1) = x_i(t) + v_i(t+1)$$

Diferența principală dintre PSO și algoritmii evolutivi este faptul că în PSO nu este folosit un proces de selecție.

Aplicație 3. Să se implementeze varianta descrisă a algoritmului PSO și să se testeze pentru funcția pătratică și funcția Griewank (utilizate și la testarea strategiilor evolutive).
Indicatie. Vezi funcția PSO.sci

Exercițiul 2. Să se modifice implementarea anterioară astfel încât pentru fiecare particulă *i* , elementul *best* să fie calculat ținând cont doar de elementele aflate în vecinătatea particulei *i* (pentru o vecinătate de dimensiune *K* acestea sunt elementele cu indicii: *i-K, i-K+1,...,i-1,i+1,...,i+K-1,i+K*). Topologia folosită pentru stabilirea vecinătăților este circulară (elementul *m* este vecin cu elementul *1*).