Algoritmi metaeuristici.

Lab 5:
Alte metaeuristici inspirate de natură:

- **Modelul coloniei de furnici (ACO-Ant Colony Optimization)**
- **Modelul ansamblului de particule (PSO – Particle Swarm Optimization)**
- **Differential Evolution. Algoritmi coevolutivi.**

---

1. 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 populație de furnici artificiale (agenți). Fiecare dintre acești agenți construiește la fiecare generație câte o soluție. Completarea componentelor soluțiilor se realizează în manieră probabilistă iar probabilitățile de selecție a uneia sau alteia dintre valorile posibile se calculează folosind atât informații de natură locală privind problema de rezolvat (ceea ce observă furnica) cât și informații de natură globală (obținute 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 orașe și a costurilor acestor drumuri (matricea de costuri).

Se folosește o populație de *m* furnici plasate inițial aleator în noduri (se poate considera că toate furnicile sunt inițial plasate în același nod – de exemplu în primul nod). La fiecare generație, fiecare furnică efectuează *n* deplasări între orașe. Furnicile memorează orașele vizitate astfel încât să nu treacă de două ori prin același oraș. Trecerea furnicii *k* din orașul *i* în orașul *j*, la etapa *t*, se bazează pe distribuția 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 probabilității au următoarea semnificație:

- τ_{ij} modelează cantitatea de feromoni depusă de furnici pe arcul *(i,j)*; se inițializează aleator și după fiecare generație 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 aparține traseului parcurs de furnica k și 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 informația locală privind calitatea arcului; cea mai simplă variantă este când η_{ij} este invers proporțional cu costul arcului (i,j)

- α si β sunt parametri care controlează ponderea relativă a celor două tipuri de informații: cea furnizată de concentrația de feromoni și cea bazată pe informația locală privind calitatea arcului.
- N(i,k) reprezintă lista vecinilor nodului i care nu au fost încă vizitați de către furnica k

Aplicație 1. Să se implementeze varianta descrisă a algoritmului ACO și să se testeze pentru problema comis-voiajorului
Indicatie. Vezi funcția ACO_TSP.sci

Exercițiu. 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.

2. 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 2. 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

Exercițiu. 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*).

3. Differential Evolution.

Este o tehnică de optimizare foarte populară care se bazează pe o regulă simplă de construire a unor noi soluții candidat ce implică utilizarea unor diferențe între elementele populației. Ideea fundamentală este că, la fiecare iterație, pentru fiecare element x(i) al populației se efectuează următoarele prelucrări:

- Se construiește un vector "mutant", y, prin combinarea unor elemente ale populației. Două dintre cele mai folosite variante de construire a unui mutant sunt:

- DE/rand/1/bin: y=x(r1)+F*(x(r2)-x(r3)) unde r1,r2,r3 sunt indicii unor elemente selectate aleator din populație (cu restricția să fie diferite între ele)
- DE/best/1/bin: y=x(ibest)+F*(x(r2)-x(r3)) unde x(ibest) este cel mai bun element din populație iar r1,r2 sunt indicii unor elemente selectate aleator din populație (cu restricția să fie diferite între ele)
- Obs: in ambele cazuri F este un factor de scalare cu valori în (0,2).

- Se construiește o nouă soluție candidat, z, prin încrucișarea componentelor lui y cu cele ale elementului current x(i) după regula (cunoscută sub numele de încrucișare binomial):
  - z(j)=y(j) cu probabilitatea CR
  - z(j)=x(i,j) cu probabilitatea 1-CR
  - Obs: CR (cu valori în (0,1)) reprezintă cel de al doilea parametru al algoritmului

Aplicatie 3. Să se implementeze algoritmul DE (ambele variante) și să se compare performanța lui în raport cu cel al unei strategii evolutive simple (vezi lab 3).

Indicație: un exemplu de implementare este în fișierul DE.sci; comparația se poate efectua folosind functiile test utilizate la lab 3.

4. Algoritmi coevolutivi.

Eficiența metaeuristicilor bazate pe populații scade în cazul problemelor de dimensiuni mare (cu mai mult de 100 de variabile). O posibilitate de a asigura scalabilitatea acestor metode este de a utiliza tehnica coevoluției cooperative care se bazează pe ideea descompunerii problemei inițiale în mai multe subprobleme de dimensiune mai mică:

- Cele n variabile ale problemei se grupează în k<n componente, C1,C2,…,Ck; ideal ar fi ca variabilele care sunt corelate între ele să fie plasate în aceeași componentă; în cazul în care nu se cunosc interacțiunile dintre variabile alocarea variabilelor la componente se poate face aleator
- Pentru fiecare componentă se aplică un algoritm metaeuristic (se consideră că fiecare componentă definește o problemă de dimensiune mai mică); pentru a evalua fiecare componentă trebuie construită o soluție candidat completă (prin adăugarea variabilelor corespunzătoare celorlalte componente, care formează contextual de evaluare a componentei curente). Există mai multe moduri de a construi contextul:
  - Se consideră variabilele corespunzătoare celui mai bun element din populația corespunzătoare generației anterioare
  - Se consideră variabilele corespunzătoare unui element aleator din populația corespunzătoare generației anterioare

Obs.

- Algoritmul corespunzător fiecărei componente poate fi executat pentru o singură iterație sau pentru mai multe iterații înainte de sincronizarea acestora.
- Algoritmii aplicați componentelor pot fi executați și în manieră asincronă, adică modificările corespunzătoare componentei sunt operate asupra la nivelul populației complete imediat dupa finalizarea iterațiilor aferente componentei (fără a se aștepta etapa de sincronizare).

Aplicație 4. Modificați algoritmul pentru DE (de la aplicația 3) pentru a implementa varianta coevolutivă și testați-l pentru probleme de dimensiune 100, 500 și 1000.