Algoritmi metaeuristici.

Lab 3: Implementarea algoritmilor evolutivi:
- **Optimizare combinatorială (algoritmi genetici)**
- **Optimizare continuă (strategii evolutive)**

---

1. Etapele implementării unui algoritm genetic.

Implementarea unui algoritm genetic presupune definirea următoarelor componente (module sau funcții):

- *Inițializare populație*: elementele populației se inițializează aleator în domeniul de definiție al problemei. In cazul algoritmilor genetici acesta este de regulă $\{0,1\}^n$
- *Evaluare populație*: calculul valorii fitness-ului (scorul sau funcția obiectiv) pentru fiecare element al populației (depinde direct de problema de rezolvat)
- *Selecție părinți*: din populația curentă se selectează o mulțime de părinți care vor participa la construirea de noi elemente prin încrucișare. În general se folosește un algoritm de selecție care implică decizii aleatoare:
  - *Selecție proportională* (probabilitatea de selecție este direct proporțională cu valoarea funcției scor - scopul algoritmului este de a maximiza valoarea scorului)
  - *Selecție bazată pe ranguri* (probabilitatea de selecție este direct proporțională cu rangul elementului - populația este ordonată crescător după valoarea scorului)
  - *Selecție de tip turneu* (se selectează uniform aleator un eșantion și din acesta se alege cel mai bun element)
- *Incrucișare:*
  - Cu un punct de tăietură
  - Cu două puncte de tăietură
  - Uniformă
- *Mutație:*
  - La nivel de cromozom (se selectează aleator, cu o probabilitate de mutație dată, un element al populației și în cadrul acestuia se complementează o poziție – aleasă aleator)
  - La nivelul întregii populații (pentru fiecare dintre pozițiile fiecărui element al populației se decide aleator, pe baza probabilității de mutație dacă se complementează sau nu)
- *Alegerea mecanismului de selecție a supraviețuitorilor.* Din populația reunită a părinților și copiilor se poate construi populația corespunzătoare noii generații prin:
  - *Selecție de tip turneu:* se selectează aleator câte două sau mai multe elemente din populația reunită și se transferă în noua populație cel de cost mai mic (scor mai mare)
  - *Selecție locală:* din mulțimea celor 4 elemente implicate în încrucișare și mutație se aleg cele mai bune două elemente
  - *Selecție prin trunchiere:* se aleg cele mai bune m elemente din populatia reunită a părinților și copiilor (m e dimensiunea populației)

- *Alegerea parametrilor de control.* Principalii parametri de control sunt:

  - Dimensiunea populației (m); se alege corelat cu dimensiunea problemei (numărul de variabile ale funcției obiectiv)

- Probabilitatea de mutație (pm); se aleg valori mici, pm<=0.1, uneori correlate cu caracteristicile populatiei sau ale problemei (de exemplu pm=1/m sau pm=1/n, m fiind dimensiunea populație, iar n fiind numărul de variabile)
- Probabilitatea de încrucișare (pc); se aleg valori relativ mari (pc>0.8)

- *Stabilirea criteriului de oprire.* Criteriul de oprire se poate referi la numărul de iteratii parcurse, la valoarea atinsa de functia de optimizat sau la alte caracteristici ale populatiei (variabilitatea populatiei a scăzut sub un anumit prag)

Aplicație 1. Problema ONEMAX

Se caută șirul binar (x_1, x_2,…, x_n) care maximizează $x_1+x_2+\ldots+x_n$.

Indicatie: vezi funcția onemaxGA.sci

Exercitii :
1. Analizati influența numărului de elemente din populație și a probabilității de mutație asupra calității soluției și a numărului de generații necesar pentru a obține soluția optimă

Aplicatie 2. Problema rucsacului

Se consideră un set de obiecte (caracterizate prin greutăți specificate prin w și valori specificate prin v) și un rucsac de capacitate C. Se caută șirul binar (x_1, x_2,…, x_n) care:
- satisface restricția: $w_1*x_1+w_2*x_2+\ldots+ w_n* x_n<=C$ (suma greutăților obiectelor selectate nu depășește capacitatea rucsacului)
- maximizează funcția: $v_1*x_1+v_2*x_2+\ldots+ v_n* x_n$ (suma valorilor obiectelor selectate este maximă)

Indicatie: se folosește codificare binară (elementul x_i este 1 dacă obiectul i este selectat și 0 în caz contrar) și se include restricția în funcția obiectiv prin tehnica penalizării. Aceasta înseamnă că funcția scor (care trebuie mazimizată) este:

$f(x_1,x_2,\ldots, x_n)= v_1*x_1+v_2*x_2+\ldots+ v_n* x_n$ dacă $w_1*x_1+w_2*x_2+\ldots+ w_n* x_n<=C$ (restricția este satisfăcută)

$f(x_1,x_2,\ldots, x_n)=$lambda $*(v_1*x_1+v_2*x_2+\ldots+ v_n* x_n)+(1-$lambda$)* (C- (w_1*x_1+w_2*x_2+\ldots+ w_n* x_n))$ dacă $w_1*x_1+w_2*x_2+\ldots+ w_n* x_n>C$ (restricția este încălcată)

Obs. Parametrul lambda ia valori în (0,1) și controlează importanța celor două componente: maximizarea scorului respectiv satisfacerea restricției (cu cât alfa este mai mic cu atât este considerată mai importantă satisfacerea restricției).

Exemplu de implementare: knapsack.sci

Exercitii :
1. Analizati influența coeficientului utilizat la includerea termenului de penalizare în funcția obiectiv (lambda) asupra calității soluției. Valori mici ale lui lambda favorizează

îndeplinirea restricției pe când valori mari favorizează maximizarea valorii obiectelor selectate

2. Să se implementeze varianta de selecție în care nu se combină valoarea funcției obiectiv cu gradul de violare a restricției ci se utilizează regula de comparare a soluțiilor candidat pentru problemele de optimizare cu restricții (vezi varianta de la Lab 2 – implementare Tabu Search).

Aplicatie 3. Rezolvarea problemei comis-voiajorului folosind un algoritm genetic (GA). Elementele specifice ale unui algoritm genetic utilizat pentru rezolvarea problemei comis-voiajorului sunt:

- Fiecare element din populație este o permutare de ordin n (n fiind numărul nodurilor)
- Populația inițială poate fi construită folosind permutări aleatoare (similar cu implementarea de la Simulated Annealing – Lab 2)
- Operatorul de încrucișare trebuie să asigure faptul că prin combinarea a două permutări se obțin tot permutări (vectori cu componente distincte). Din acest motiv, încrucișarea cu un punct de tăietură utilizată în exemplele anterioare nu poate fi utilizată direct în cazul codificării de tip permutare. O variantă de încrucișare cu un punct de tăietură în cazul elementelor de tip permutare este:
    - Se alege aleator un punct de tăietură (k).
    - Se transferă primele k componente din primul părinte la primul urmaș; celelalte componente din primul părinte se transferă tot la primul părinte însă în ordinea în care sunt specificate în cel de al doilea părinte.
    - Se procedează similar cu al doilea părinte/urmaș

- Operatorul de mutație (construirea unei noi configurații pornind de la cea curentă) se bazează pe transformările utilizate în algoritmii euristici de rezolvare a problemei comis-voiajorului (de exemplu, 2-opt)

Observatie: în cazul problemei comis-voiajorului mutația joacă un rol mai important decât încrucișarea.

Indicatie. O variantă bazată pe operatorii clasici este descrisa in GA_TSP.sci

Exercitiu:

1. Să se implementeze o mutație bazată pe interschimbarea a două componente (echivalent cu aplicarea unei inversiuni).
2. Să se analizeze comportarea algoritmului în cazul în care nu se folosește încrucișare.

Temă.

Să se modifice algoritmul genetic din onemaxGA.sci:

a) înlocuirea funcției de selecție bazată pe metoda ruletei cu cea bazată pe Stochastic Universal Sampling (vezi curs 4);
b) înlocuirea încrucișării cu un punct de tăietură cu încrucișare uniformă;
c) înlocuirea mutației la nivel de cromozom cu mutație la nivel de populație (ansamblu de gene).