Algoritmi metaeuristici.

Lab 4: Programare genetică.
Alte metaeuristici inspirate de natură:
- **Modelul coloniei de furnici (ACO-Ant Colony Optimization)**
- **Modelul ansamblului de particule (PSO – Particle Swarm Optimization)**

---

1. Programare genetică.

Programarea genetică permite proiectarea în manieră evolutivă a unor structuri destinate rezolvării unei probleme (expresii aritmetice, expresii logice, reguli sau chiar programe). Elementele populației au în general o structură ierarhică (arbori).

Operatorii genetici (încrucișare și mutație) sunt specifici reprezentării ierarhice, iar una dintre problemele cele mai importante se referă la complexitatea structurilor obținute în procesul de evoluție. Ca atare, atât la inițializarea populației cât și pe parcursul aplicării operatorilor se urmărește ca arborii să nu depășească o anumită "dimensiune" (măsurată prin adâncimea arborelui = numărul de nivele în arbore).

Una dintre cele mai sugestive aplicații ale programării genetice este *regresia simbolică* al cărui scop este determinarea unei expresii care fitează un set de date (relație funcțională care descrie legătura dintre un set de valori corespunzătoare unei mărimi dependente și un set de valori corespunzătoare unei mărimi independente).

Aplicație 1a. Să se testeze funcționarea applet-ului pentru regresie simbolică folosind programare genetică accesibil la http://www.geneticprogramming.org/symbolic/main.htm

Etape:
- *Alegerea funcției de test* (Function Settings): se specifică domeniul de definiție (Min X, Max X), funcția care va fi aproximată (Enter Function) și numărul de valori ale funcției folosite în procesul evolutiv (Number of points)
- *Setarea parametrilor algoritmului*: număr generații, dimensiunea populației, adâncimea maximă a arborilor utilizați pentru specificarea expresiilor care fac parte din populația inițială, fractiunea de elemente care participă la încrucișare, fracțiunea de elemente care sunt supuse mutației, adâncimea maximă a arborilor construiți prin mutație, adâncimea maximă a subarborilor folosiți în procesul de mutație.
- *Alegerea modului de inițializare a populației*:
  - Full: se generează noi noduri până când toate ramurile din arbore au lungimea egală cu cea maximă.
  - Grow: extinderea unei ramuri în arbore se oprește fie când s-a atins lungimea maximă fie la realizarea unui eveniment aleator.
  - Ramped half-half: jumătate dintre elementele populației sunt generate după modelul Full, iar cealaltă jumătate după modelul Grow.
- *Alegerea metodei de selecție*:
  - Proporțională
  - De tip turneu
- *Alegerea setului de neterminale* (prin marcare din setul de operatori/funcții afișate: +,-,*,/, *sin,cos,exp, abs, max, min, log*)

- *Alegerea setului de terminale* (variabila *x* și constante aleatoare)

Aplicație 1b. Să se determine expresia care aproximează cel mai bine un set de date (regresie simbolică) folosind pachetul "rgp" din R.

Etape:

- Lansare R
- Incărcare pachet "rgp": Packages ->Load package … sau library("rgp") (dacă pachetul nu apare în lista de pachete instalate trebuie instalat prin Packages-> Install package(s)…
- Definirea setului de neterminale (operatori si funcții) prin functionSet. Exemplu: setNeterminale <- functionSet("+", "*", "-","/")
- Definirea setului de variabile prin inputVariableSet. Exemplu: setVariabile <- inputVariableSet("x")
- Definirea setului de constante prin constantFactorySet. Exemplu: setConstante <- constantFactorySet(function() rnorm(1)) (valori generate folosind repartiția normală standard)
- Definirea datelor de test: valori care vor fi utilizate pentru a evalua calitatea aproximării. Exemplu: dateX <- seq(from = -pi, to = pi, by = 0.1)
- Definirea funcției scor: eroarea medie pătratică (măsură a diferenței dintre valorile funcției de test și valorile corespunzătoare expresiilor din cadrul populației). Exemplu: functieScor <- function(f) rmse(f(dateX), sin(dateX))
- Apelul funcției care simulează procesul evolutiv (funcția geneticProgramming). Exemplu: geneticProgramming(functionSet = setNeterminale,inputVariables = setVariabile,constantSet = setConstante, fitnessFunction = functieScor,stopCondition = makeStepsStopCondition(10000))

Particularități ale variantei de programare genetică din pachetul "rgp":

- Elementele populației sunt expresii R (care sunt implementate ca structuri arborescente)
- Inițializarea populației se bazează pe strategiile de construire:
  - "grow" (extinderea unei ramuri din arbore continuă fie până la atingerea adâncimii maxime fie până la realizarea unui eveniment aleator)
  - „full" (toate ramurile din arbore au lungimea maximă)
  - Varianta combinată (unele elemente sunt construire folosind strategia "grow", altele folosind strategia „full")
- Sunt implementați operatorii tradiționali de încrucișare și mutație la nivel de arbori (vezi slide-uri curs 6)
- Sunt implementate strategii de selecție care folosesc unul sau mai multe criterii (specific optimizării multicriteriale). In varianta multicriterială se urmărește optimizarea simultană a calității și simplității elementelor precum și a diversității populației.

Exercițiu 1: Parcurgeți etapele descrise mai sus și testați influența setului de neterminale asupra calității rezultatului. Indicație: gp1.r

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

Exercițiu 2. 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

Exercițiu 3. 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*).