Algoritmi metaeuristici.

Lab 4: Strategii evolutive
Programare genetică.

---

1. Implementarea strategiilor evolutive

In cazul strategiilor evolutive elementele populației sunt vectori cu valori reale iar elementele specifice (care îi diferențiază de algoritmii genetici sunt):

- *Selecție:* se folosește de regulă doar pentru alegerea supraviețuitorilor și este o selecție deterministă bazată pe alegerea celor mai bune elemente din populația copiilor (în variantele (M,L)) sau din populația reunită a părinților și copiilor (în variantele (M+L)). In notația tradițională M reprezintă numărul de părinți iar L reprezintă numărul de urmași.
- *Recombinare:* cel mai frecvent se folosește recombinarea convexă bazată pe selecției aleatoare a R părinți și construirea unui urmaș.
- *Mutație:* se adaugă o valoare aleatoare (generată în conformitate cu o anumită repartiție) fiecărui element al populației.

Aplicație 1. Implementarea unei strategii evolutive simple caracterizate prin:
- Recombinare bazată pe calculul mediei aritmetice a părinților (numărul de părinți este specificat ca parametru de intrare)
- Mutație bazată pe perturbare aleatoare cu repartiție normala (medie 0 și abatere standard sigma – sigma e specificată ca parametru de intrare)
- Selecție a supraviețuitorilor: prin trunchiere sau de tip turneu

Strategia va fi testată pentru minimizarea unor funcții de test dintre cele disponibile la http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm

Indicatie: vezi SE.sci

Exerciții:
1. Testați algoritmul pentru funcțiile Rastrigin, Ackley și Rosenbrock (de la pagina web menționată mai sus)
2. Analizați influența parametrului sigma asupra comportării algoritmului
3. Analizați influența tipului de selecție (prin trunchiere sau de tip turneu) asupra comportării algoritmului

Aplicație 2. Analiza comportării algoritmului CMA-ES.
Indicație. Se poate folosi pachetul Scilab pentru CMA-ES disponibil la https://www.lri.fr/~hansen/cmaes_inmatlab.html#scilab

Etape:

(a) se descarca arhiva (de la https://atoms.scilab.org/toolboxes/CMA-ES) si se dezarhiveaza sursele intr-un director local
(b) se seteaza directorul local ca director de lucru in Scilab (prin Change current directory)
(c) se executa builder.sce si loader.sce
(d) se instantiaza un obiect de tip cmaes folosind cma_new si se specifica prelucrarea repetitivă care conține: (i) construirea de noi soluții candidat (folosind cma_ask); (ii) actualizarea tuturor componentelor și parametrilor ce vor fi utilizați la iterația următoare (folosind cma_tell).

Exercitiu:
1.Comparați comportarea algoritmului CMA-ES cu cea a strategiei evolutive simple pentru funcția Rosenbrock (cu dimensiunile n=2, 10, 100)

2. 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: Parcurgeți etapele descrise mai sus și testați influența setului de neterminale asupra calității rezultatului. *Indicație:* SymbolicRegression.r