Procesarea Cunoștințelor și Calcul Inteligent

Algoritmul Particle Swarm Optimization – partea I

Generarea populației inițiale și calculul funcției de adaptare.

Algoritmul PSO (Particle Swarm Optimization - Optimizarea Deplasării Roiurilor de Particule) este un algoritm de calcul inteligent, inspirat din natură. Calculul inteligent este un subdomeniu al inteligenţei artificiale şi se referă la algoritme care încorporează însuşiri specifice inteligenţei fiinţelor vii:

- memoria
- învăţarea din experienţa anterioară
- cooperarea între indivizii unei comunităţi pentru a atinge un scop comun.

PSO este, totodată, un algoritm de optimizare metaeuristic şi iterativ.

Optimizare înseamnă găsirea soluţiei optime sau a celei mai bune soluţii pentru o problemă. O problemă de optimizare calculează o funcţie obiectiv, a cărei valoare indică gradul de optimizare atins. Problemele de optimizare sunt

- De maximizare – caută valoarea maximă a funcţiei obiectiv, adică maximizează valoarea funcţiei obiectiv. Maximizarea se referă la avantaje sau caracteristici pozitive (de ex., profit, nivel de siguranţă, fiabilitate).
- De minimizare – caută valoarea minimă a funcţiei obiectiv, adică minimizează funcţia obiectiv. Minimizarea are în vedere dezavantaje sau caracteristici negative (de ex. costuri, pierderi, factori de risc).
- De găsire a unei valori dorite, care satisface anumite cerinţe. În acest caz, nu se caută minimul sau maximul funcţiei obiectiv, ci o anumită valoare numerică sau vectorială.

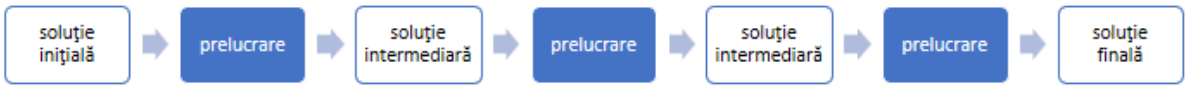
În terminologia ştiinţifică, *metodă euristică* înseamnă o metodă de căutare care nu garantează obţinerea rezultatului celui mai bun, însă dă suficient de repede rezultate suficient de bune ca să fie considerate satisfăcătoare. Metodele de optimizare euristice găsesc nu optimul global („cel mai bun dintre cei mai buni”), ci un optim local (o soluţie suficient de bună)

Metodele euristice aplică unul sau mai multe dintre dintre următoarele principii:

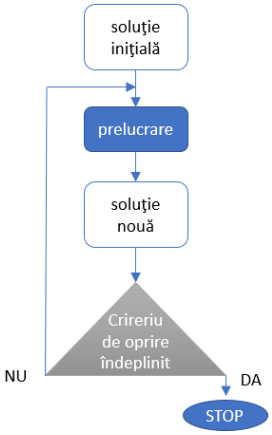
- Dacă nu se cunoaşte soluţia căutată, porneşte de la una sau mai multe soluţii cunoscute, concrete, şi caută alte soluţii noi
- Pentru a obţine soluţii noi mai bune, soluţiile noi sunt căutate parţial prin căutare educată (bazată pe regulile bunului-simţ sau pe experienţa anterioară) şi parţial prin aplicarea unor factori aleatorii (sochastici)
- Soluţiile sunt căutate iterativ (repetitiv)

Metodele euristice sunt construite pentru rezolvarea unei probleme date, clar definite. Metodele **metaeuristice** (meta - dincolo de) sunt metode euristice aplicabile mai multor tipuri de probleme, folosind aceiaşi paşi, adaptaţi la problema respectivă.

Algoritmele **iterative** sunt algoritme care pornesc de la o soluţie iniţială (numită aproximare iniţială a soluţiei), care este trecută de mai multe ori printr-un proces de rafinare sau îmbunătăţire (mereu acelaşi), până la obţinerea unei soluţii finale satisfăcătoare:



sau



Algoritmul PSO

PSO este inspirat din modul specific de deplasare al bancurilor de pești sau a stolurilor de păsări, care urmează un lider și fac schimbări de direcție sincrone fără a se ciocni. Bancurile și stolurile sunt alcătuite din indivizi ai speciei care explorează sau străbat un spațiu de căutare, în căutarea unui scop (hrana) sau a unei destinații (țările calde).

Algoritmul iniţializează aleatoriu o populaţie de soluţii alcătuită din valori numerice sau şiruri de numere, numită *roi (swarm)*, pe care o optimizează într-un proces iterativ, cu scopul de a descoperi soluţia optimă (destinația).

Fiecărei soluţii, numită *particulă (particle)*, îi este asociată o *viteză (speed)* inițială aleatorie, sub forma unui vector de aceeaşi dimensiune, folosită pentru a o deplasa în spaţiul de căutare. La un moment dat, particula este caracterizată printr-o *poziţie (position)* în spaţiul de căutare (adică prin valorile elementelor vectorului) Prin aplicarea vectorului viteză peste vectorul poziţie, particula se deplasează într-o poziţie nouă (elementele componente ale vectorului-particulă se schimbă)

Performanța fiecărei soluții este evaluată printr-o valoare numerică, numită *funcție de adaptare (fitness function)*. Funcția de adaptare se obține aplicând soluția (poziţia) pe problema dată și evaluând rezultatele obținute. Mecanismul PSO este independent de problemă. Căutarea se face exclusiv după valorile funcțiilor de adaptare ale soluțiilor (ale pozițiilor particulelor)

Pentru fiecare particulă, algoritmul PSO reţine permanent cea mai bună poziţionare (structură) atinsă într-o valoare *personal best (pbest, optim individual)*, pentru care se memorează particula propriu-zisă şi valoarea funcţiei sale de adaptare.

De asemenea, este memorată şi cea mai bună poziționare a celei mai bune particule din roi, aşa-numitul *global best (gbest, optim global* sau *lider).*

Într-o iteraţie, fiecare particulă îşi schimbă viteza (accelerează) simultan către cea mai bună poziţie a sa (*pbest*) şi cea mai bună poziţie identificată de întregul roi (*gbest*). Acceleraţia este ponderată aleatoriu pe cele două direcţii.

Algoritmul metodei

Algoritmul standard PSO parcurge următorii paşi:

- Iniţializează o populaţie aleatorie (un roi) de indivizi (particule) cu *d* variabile (dimensiuni), cărora li se asociază câte un vector aleatoriu de viteză, de dimensiune *d*;

  repetă
  Pentru fiecare particulă,
  - Se calculează funcţia de adaptare
  - Se compară funcţia de adaptare curentă a particulei cu valoarea corespunzătoare optimului individual *pbest*. Dacă a fost găsită o valoare mai bună decât *pbest*, particula pbest anterioară este înlocuită de particula curentă, fiind memorată împreună cu funcţia ei de adaptare
  - Se compară funcţia de adaptare curentă a particulei cu valoarea optimă globală descoperită până în prezent de roi, *gbest*. Dacă a fost găsită o valoare mai bună, decât *gbest*, particula *gbest* (*lider*) memorată anterior este înlocuită de particula curentă, împreună cu funcţia ei de adaptare
  - Se actualizează viteza şi poziţia particulei, cu relaţiile:

$$v = v + c_1 \cdot random \cdot (pbest - particula) + c_2 \cdot random \cdot (gbest - particula)$$

$$particula = particula + v$$

  până se atinge un anumit număr de generaţii ori o valoare acceptabilă a funcţiei de adaptare a particulei *gbest*.

În relațiile de calcul de mai sus, *random* este un vector cu numere aleatorii, cu dimensiunea egală cu a unei particule, iar c_1 și c_2 sunt două constante care, de regulă, iau valoarea 2.

Definirea problemei

Vom modela algoritmul PSO pentru o problemă simplă: ghicirea unui cuvânt. Algoritmul va ști ca date de intrare următoarele informații:

- Dimensiunea cuvântului căutat
- Literele alfabetului

Se dă:

O populaţie iniţială de particule, generată aleatoriu

Se cere:

Particula optimă, care rezolvă cel mai bine problema

Modelarea problemei pentru PSO

Ca să putem rezolva o problemă folosind algoritmul PSO, trebuie să modelăm soluţiile posibile şi soluţia optimă a problemei sub formă de vectori, preferabil numerici (ca să-i putem grupa şi manipula uşor sub formă de matrice, specialitatea de bază a Matlab-ului). Cum putem, aşadar, să reprezentăm un cuvânt printr-un şir de numere ?

O metodă simplă este să alocăm un număr fiecărei litere din alfabet. Am putea, de pildă, să facem corespondenţa 1 - a, 2 – b ş.a.m.d.

De fapt, în informatică există deja o asemenea convenţie standardizată la nivel mondial. Pentru reprezentarea simbolurilor grafice afișate pe ecran, calculatoarele folosesc codul ASCII, în care fiecărui caracter îi este asociat un cod numeric. Întregul cod ASCII poate fi consultat la adresa web https://www.ascii-code.com/.

Din întregul cod ASCII de 256 de caractere, pentru problema noastră ne vom limita la literele mici ale alfabetului, care au codurile ASCII 97 – 122.

Astfel, literei a îi corespunde codul 97, literei b, codul 98 ş.a.m.d. până la litera z, care are codul 122.

Prin urmare, folosind codul ASCII, orice cuvânt poate fi reprezentat printr-un şir (vector) de numere întregi, cu valori între 97 şi 122.

De exemplu, cuvântul “albastru” se poate scrie în cod ASCII astfel:

a	l	b	a	s	t	r	u
97	108	98	97	115	116	114	117

Dacă îi cerem algoritmului PSO să ghicească acest cuvânt, va trebui să descopere vectorul numeric asociat, adică (97 108 98 97 115 116 114 117). Pentru aceasta, el dispune de următoarele informaţii:

- Lungimea cuvântului – 8
- Numerele posibile între care poate să realizeze căutarea: 97-122

Soluţia (97 108 98 97 115 116 114 117) este soluţia optimă sau optimul global pe care dorim să îl identificăm. Alte soluţii foarte apropiate de aceasta de pildă (97 **107** 98 97 115 116 114 117) sau (**96** 108 98 97 115 116 **115** 117) sunt soluţii suboptime sau optime locale. Este posibil ca algoritmul nostru să descopere o asemenea soluţie suboptimă, dar se poate să o descopere chiar şi pe cea optimă.

Un fapt interesant este acela că, după cum vom vedea mai târziu, această secvenţă de numere nu are nicio relevanţă pentru algoritm şi mecanismul său de căutare. Iniţial, algoritmul nu ştie la ce soluţie trebuie să ajungă. El va fi direcţionat în căutare de valorile calculate pentru funcţia de adaptare a fiecărei particule.

Generarea populaţiei iniţiale

Deoarece algoritmul nu cunoaşte secvenţa optima de numere care formează soluţia căutată, el pleacă de la un grup sau o populaţie sau un roi de soluţii alese la întâmplare, dar obligatoriu valide.

Soluţie validă înseamnă un vector de 8 numere întregi cu valori cuprinse între 97 şi 122. O populaţie de soluţii valide înseamnă un grup de asemenea vectori.

Cea mai comodă cale de a stoca un grup de vectori de lungime egală este o matrice, în care vectorii respectivi pot fi linii sau coloane. În continuare, vom considera că o soluţie validă este o linie din matricea populaţiei, pe care initial o vom genera aleatoriu, folosind secvenţa de cod deja cunoscută din capitolele anterioare.

Pentru exemplificare, vom genera aleatoriu o populaţie de zece particule (indivizi, vectori). Pentru aceasta, avem nevoie să iniţializăm următoarele variabile: numărul de particule din populaţie (vom numi variabila `np`), numărul de elemente sau dimensiunea unei particule (`dp`), valoarea minimă şi valoarea maximă pe care le poate lua un element dintr-o particulă (`linf`, `lsup`).

Funcţia `rand` va genera rezultate diferite la fiecare rulare. De aceea, ca să puteţi urmări succesiunea şi logica calculelor, vom genera o populaţie iniţială pe care o vom stoca sub formă de variabilă şi o veţi folosi ca atare.

Codul care generează populaţia iniţaială este:

```
% nr particule in populatie
np=10;

% dimensiunea cuvantului cautat (a unei particule)
dp=8;

% limite valori elemente individ
linf=97;
lsup=122;

% generare populatie
pop=round(linf+rand(np,dp)*(lsup-linf));
```

Populaţia generată aleatoriu pe care o vom folosi în continuare pentru teste are următoarea structură:

99	121	98	100	98	102	98	113
104	111	119	113	116	103	114	110
117	110	120	109	110	119	98	121
98	103	117	116	109	98	99	113
120	109	99	115	120	109	110	117
115	113	104	120	112	101	99	108
109	114	105	119	112	121	117	108
111	107	114	105	118	115	117	118
103	106	100	114	117	110	115	99
108	122	115	102	111	109	101	100

Fiecare linie din matricea `pop` corespunde unei combinaţii de numere cu valori între 97 şi 122, care, folosind conversia codurilor ASCII în caractere, produce un cuvânt de opt litere.

De exemplu, prima linie din matrice corespunde următorului cuvânt:

99	121	98	100	98	102	98	113
c	y	b	d	b	f	b	q

Am obţinut literele corespunzătoare codului ASCII folosind comanda Matlab `char`, aplicată primei linii din matrice:

```
v=char(pop(1,:))
cybdbfbq
```

Aplicând comanda `char` întregii matrice, putem citi pe linii zece cuvinte formate din câte opt litere aleatorii:

```
cybdbfbq
howqtgrn
unxmnwby
bgutmbcq
xmcsxmnu
sqhxpecl
mriwpyul
okrivsuv
gjdrunsc
lzsfomed
```

După cum se poate observa, toate aceste cuvinte sunt mult diferite ce cuvântul căutat, “albastru”. Totuşi, algoritmul PSO va folosi această populaţie pentru a încerca să descopere cuvântul căutat. Pentru aceasta, el are nevoie nu de cuvântul căutat, ci de diferenţa dintre cuvântul căutat şi cuvintele cunoscute (aceste înşiruiri ciudate de litere). Reiterăm că algoritmul nu lucrează cu literele, ci cu codurile lor ASCII, care sunt numere. Astfel, poate face diferenţa între fiecare vector numeric din populaţie şi vectorul numeric al cuvântului căutat. Această diferenţă este funcţia de adaptare a fiecărei particule.

Calculul funcţiei de adaptare

Deşi toate cele zece combinaţii de litere sunt foarte îndepărtate de cuvântul căutat, totuşi, soluţiile întruchipate de ele pot fi ierarhizate în raport cu acesta (adică faţă de soluţia optimă a problemei). Una dintre ele este cea mai îndepărtată, una dintre ele este cea mai apropiată de soluţia optimă (de cuvântul „albastru”), iar celelalte sunt intermediare. Valoarea funcţiei de adaptare trebuie să poată face această ierarhie.

Pentru a ierarhiza soluţiile, cel mai simplu este să calculăm diferenţa dintre vectorul numeric al codurilor ASCII pentru literele din componenţa fiecărei particule din populaţie şi vectorul numeric al codurilor ASCII ale literelor din cuvântul căutat. Pentru o populaţie de 10 particule, trebuie să se calculeze, aşadar, 10 diferenţe. Una dintre zece valori, cea mai mică, va indica soluţia cea mai apropiată de soluţia optimă căutată.

Dacă am calcula valoarea funcţiei de adaptare a soluţiei optime, aceasta ar fi nulă (vectorul numeric al soluţiei optime este identic cu vectorul numeric al cuvântului căutat, deci diferenţa dintre ei este 0). Aşadar, problema de optimizare pe care o rezolvăm este de minimizare. Dorim să minimizăm diferenţa dintre vectorii cuvintelor din populaţie şi vectorul cuvântului căutat. Valoarea minimă absolută a acestei diferenţe este 0.

Când calculăm diferenţa între vectorii numerici ai soluțiilor, putem obţine şi diferenţe negative. Aceasta este o capcană, deoarece poate falsifica rezultatele reale. De exemplu, să comparăm diferenţele dintre cuvintele

99	121	98	100	98	102	98	113
c	y	b	d	b	f	b	q

şi

97	108	98	97	115	116	114	117
a	l	b	a	s	t	r	u

Dacă facem diferența (dorit-existent, “albastru”-“ cybdbfbq”), obținem:

97	108	98	97	115	116	114	117
				-			
99	121	98	100	98	102	98	113
				=			
-2	-13	0	-3	17	14	16	4

Sumând valori negative cu valori pozitive, ar rezulta o distanță mai mică între cei doi vectori, decât cea reală. Semnul minus nu este relevant dacă calculăm o distanță. O distanță (-13) între două elemente este egală cu o distanță (13), diferă doar sensul. De aceea, ne va interesa să sumăm nu diferențele reale dintre elementele vectorilor (cu semn), ci valorile absolute ale acestor diferențe (luate întotdeauna ca valoare pozitivă).

Nu `a-b=33`, ci `abs(a-b)=69` este valoarea reală a distanței, în acest caz **.**

Dacă folosim valoarea absolută, nu contează dacă calculăm `abs(a-b)` sau `abs(b-a)`**.**

Pentru că trebuie să calculăm un număr de diferențe între vectori egal cu numărul de particule din populație, iar aceste diferențe sunt valori numerice (sume din x numere, unde x este numărul de litere din cuvântul căutat), avem nevoie de două lucruri pentru calculul funcției de adaptare pentru întreaga populație:

- De un vector numeric cu numărul de elemente egal cu numărul de indivizi din populație - pentru a reține diferențele calculate între fiecare individ și cuvântul căutat
- De un ciclu `for` de la 1 la numărul de indivizi din populație - ca să calculăm fiecare diferență. În acest ciclu:
    - Se extrage cate o linie din matricea pop
    - Se calculează diferența, în valori absolute, între vectorul dorit, cel al cuvântului căutat, și cel extras din populație
    - Se introduce diferența în vectorul funcțiilor de adaptare.

Ca să poată fi calculate cele zece diferențe, înaintea ciclului `for` trebuie calculat vectorul dorit, cel al codurilor ASCII ale cuvântului căutat. Pentru aceasta, se folosește funcția predefinită `double`.

Dacă se calculează funcțiile de adaptare pentru populația generată pentru teste, se vor obține valorile (codul programului este dat mai jos):

populație								funcție de adaptare
99	121	98	100	98	102	98	113	69
104	111	119	113	116	103	114	110	68
117	110	120	109	110	119	98	121	84
98	103	117	116	109	98	99	113	87
120	109	99	115	120	109	110	117	59
115	113	104	120	112	101	99	108	94
109	114	105	119	112	121	117	108	67
111	107	114	105	118	115	117	118	47
103	106	100	114	117	110	115	99	54
108	122	115	102	111	109	101	100	88

Valoarea minimă a diferenței se găsește pe poziția 8, care codifică cuvântul `okrivsuv`. Din populația inițială, acesta este cuvântul cel mai apropiat de cuvântul căutat. Calculând diferențele element cu element, în valori absolute, ele arată astfel:

97	108	98	97	115	116	114	117
-							
111	107	114	105	118	115	117	118
=							
14	1	16	8	3	1	3	1

Pe trei dintre cele opt dimensiuni, această soluție se află, numeric, foarte aproape de soluția optimă. Însă există trei dimensiuni îndepărtate. În ansamblu, suntem departe de soluția căutată.

Codul complet care realizează calculul de până acum, în care generarea aleatorie a populației inițiale a fost înlocuită cu scrierea directă a matricei populației pentru teste, este următorul:

```
% 1. Parametri initiali

% curatenie
clear;
clc;
```

```
% nr particule in populatie
np=10;

% dimensiunea cuvantului cautat (a unei particule)
dp=8;

% limite valori elemente individ
linf=97;
lsup=122;

% 2. Generare populatie
 % generare populatie
%pop=round(linf+rand(np,dp)*(lsup-linf));

% populatia initiala pentru teste
pop=[99,121,98,100,98,102,98,113;
    104,111,119,113,116,103,114,110;
    117,110,120,109,110,119,98,121;
    98,103,117,116,109,98,99,113;
    120,109,99,115,120,109,110,117;
    115,113,104,120,112,101,99,108;
    109,114,105,119,112,121,117,108;
    111,107,114,105,118,115,117,118;
    103,106,100,114,117,110,115,99;
    108,122,115,102,111,109,101,100];

% 3. Functia de adaptare
 % vector numeric cuvant cautat (vector de referinta)
vref=double('albastru');

% initializare vector functii de adaptare (coloana)
fadapt=zeros(np,1);

% calcul diferente
for i=1:np % pentru fiecare particula din populatie
    % extragere particula din populatie
    vact=pop(i,:);
    % calcul diferenta absoluta fata de vectorul de referinta
    df=sum(abs(vref-vact));
    % scriere diferenta in vectorul functiilor de adaptare
    fadapt(i)=df;
end;
```

Recapitulare

Am prezentat algoritmul general PSO și am implementat în program primii doi pași ai acestuia, generarea populației inițiale și calculul funcției de adaptare.

Va urma:

Particle Swarm Optimization – partea a II-a