Procesarea Cunoștințelor și Calcul Inteligent

Algoritmul Particle Swarm Optimization – partea a II-a

Completarea pașilor unei iterații a algoritmului

Data trecută, am început construirea unui algoritm Particle Swarm Optimization (PSO) pentru rezolvarea unei probleme simple: ghicirea unui cuvânt. Pentru transformarea soluției căutate într-un vector numeric, am adoptat metoda transformării literelor care compun respectivul cuvânt în codurile lor ASCII, cu valori cuprinse între 97 și 122. Din algoritmul general PSO, care are următoarea structură:

- Inițializează o populație 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 funcția de adaptare
  - Se compară funcția 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 funcția ei de adaptare
  - Se compară funcția 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 funcția ei de adaptare
  - Se actualizează viteza și poziția particulei, cu relațiile:

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

$$particula = particula + v$$

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

am rezolvat doi pași: generarea populației inițiale și calculul funcției de adaptare. Am prezentat o metodă de generare a unei populații inițiale aleatorii, după care am generat o asemenea populație pe care o vom folosi pentru explicarea funcționării algoritmului. Am calculat apoi funcția de adaptare a fiecărei particule din această populație, ca suma diferențelor absolute între elementele componente ale vectorului codurilor ASCII ale fiecărui cuvânt și vectorul codurilor ASCII ale cuvântului de ghicit. Am ales să lăsăm algoritmul să ghicească cuvântul „albastru”, ale cărui litere au codurile ASCII de mai jos:

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

Continuăm algoritmul cu următorii pași.

Într-o iterație, algoritmul PSO deplasează fiecare particulă concomitent în două direcții: înspre cea mai bună poziție obținută vreodată de particula respectivă și către cea mai bună poziție obținută vreodată de întregul roi. Pentru aceasta, aceste poziții trebuie cunoscute. Înainte de începerea căutării, cea mai bună poziție a fiecărei particule și cea mai bună poziție a roiului nu sunt cunoscute, pentru că nu au fost calculate niciodată. Ca să cunoaștem cât de bună este poziția fiecărei particule, trebuie calculată funcția ei de adaptare.

Iniţializarea pbest şi gbest

Trebuie reamintit că poziția unei particule este totuna cu particula, ea este dată de valoarea fiecărui element al vectorului care compune particula respectivă. Așadar, pozițiile `pbest` ale particulelor care formează la un moment dat roiul alcătuiesc o populație paralelă, iar funcțiile lor de adaptare formează un vector paralel cu cel al funcțiilor de adaptare ale particulelor din populația de lucru. În etapa de inițializare, este cel mai comod să alegem o populație `pbest` și, pentru ea, un vector `fpbest` de funcții de adaptare care sigur vor fi schimbate când vor fi calculate prima oară în pasul următor determinării funcțiilor de adaptare ale populației inițiale, `fadapt`. Pentru populația `pbest`, valorile alese pentru elementele particulelor nu contează. Pentru funcțiile de adaptare `fpbest`, trebuie aleasă o valoare mare, care să fie în mod sigur schimbată în prima iterație. Odată cu valorile funcțiilor de adaptare ale particulelor `pbest`, trebuie să inițializăm și o valoare pentru funcția de adaptare a celei mai bune particule din roi, funcție pe care o vom numi `fgbest`. Şi aceasta trebuie să aibă o valoare mare. Structura particulei lider, `gbest`, nu contează, pentru că va fi schimbată oricum în prima iterație. De aceea, nici n-o mai inițializăm, pentru că nu este necesar.

Viteza iniţială a particulelor

Concomitent cu una dintre populațiile `pop` sau `pbest`, inițializăm și vitezele particulelor, o matrice pe care o vom numi `vit`. Viteza fiecărei particule este un vector de lungime egală cu lungimea particulei. O particulă își schimbă poziția prin aplicarea peste poziția anterioară a vectorului vitezei. Cea mai bună poziție a unei particule înseamnă cea mai bună poziție a fiecărui element din acea particulă. De aceea, căutarea celei mai bune poziții se realizează în paralel pentru fiecare element din particulă, adică cu viteză diferită pentru fiecare element.

Pentru a nu influența în mod intenţionat șansele vreunei particule de a găsi soluția (poziția) optimă căutată de algoritm, vitezele inițiale ale particulelor se inițializează mici și aleatorii.

Așadar, înainte de a continua cu pasul 3, calculul `pbest` și `gbest`, modificăm programul scris până acum. În zona de inițializări, creăm populația inițială `pbest`, vectorul funcțiilor sale de adaptare, `fpbest`, funcția de adaptare a liderului roiului, `fgbest`, și vitezele inițiale `vit`.

Observați și rețineți convenția adoptată pentru notații: la particule:, pentru valori curente, optim personal și optim global: `pop` - `pbest` – `gbest`, respectiv, pentru funcțiile de adaptare, `fadapt` – `fbest` - `fgbest`.

```
% curatenie
clear;
clc;

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

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

% populatia initiala pbest
pbest=rand(np,dp);

% functiile de adaptare ale populatiei pbest
fpbest=10e6*ones(np,1);

% functia de adaptare initiala a liderului
fgbest=10e6;

% vitezele initiale
vit=0.01*rand(np,dp);

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

Reamintim că, în capitolul anterior, am generat o populație inițială pentru teste, căreia i-am calculat funcțiile de adaptare.

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

Calculul optimului personal și optimului global (pbest, gbest)

PSO actualizează în fiecare iteraţie populația paralelă `pbest`, vectorul funcțiilor sale de adaptare `fpbest`, liderul `gbest` și funcția sa de adaptare `fgbest`, conform logicii din algoritmul de principiu. Optimul personal al unei particule se evaluează după valoarea funcției sale de adaptare. De exemplu, într-o iteraţie oarecare, particula 3 din populația paralelă `pbest` va fi înlocuită cu particula 3 din populația curentă `pop` dacă funcția de adpatare `fadapt(3)` a particulei curente are o valoare mai bună decât valoarea existentă în `fpbest(3)`. Pentru problema noastră, o valoare mai bună înseamnă o valoare mai mică, deoarece dorim să minimizăm diferența dintre vectorul soluției optime și el al soluției găsite. Cea mai bună valoare a unei funcţii de adaptare poate fi 0, când nu există nicio diferenţă între cuvântul găsit de algoritm şi cel de referinţă.

Această verificare și înlocuire trebuie făcută pentru toate particulele din populație și pentru funcțiile lor de adaptare. În Matlab, putem face această operație simultan pentru toate particulele din

populație, cu ajutorul funcției `find`. Procedura de actualizare a populației `pbest` și a vectorului `fpbest` presupune doi pași:

- Găsește elementele din `fadapt` care au valori mai mici decât cele din `fpbest`

```
inloc=find(fadapt<fpbest);
```

- Înlocuiește în `pbest` și `fpbest` particulele, respectiv funcțiile lor de adaptare, cu valorile actualizate, mai bune

```
pbest(inloc,:)=pop(inloc,:);
fpbest(inloc,:)=fadapt(inloc,:);
```

Vectorul `inloc` va conține pozițiile din `fadapt` în care funcţiile de adaptare au valori mai mici decât în `fpbest`. În prima iterație, deoarece am inițializat vectorul `fpbest` cu valori de 10^6, iar valorile funcțiilor de adaptare calculate pentru populația inițială au valori mai mici de 100, condiția va fi respectată pentru toate elementele vectorului. Cu alte cuvinte, poziția inițială optimă sau personal best-ul inițial al fiecărei particule din populația inițială este chiar particula respectivă, deoarece aceea este prima poziție cunoscută a particulei. În iterațiile următoare, nu mai este obligatoriu să se întâmple așa.

După ce am actualizat `pbest`, putem găsi particula optim global `gbest` și funcția ei de adaptare `fgbest` căutând particula cu valoarea funcției de adaptare minime din vectorul `fpbest`. Unul dintre optimele personale ale particulelor este optimul global al roiului, liderul.

- se caută în vectorul `fpbest` poziția valorii minime a acestui vector:

```
lid=find(fpbest=min(fpbest));
```

Este posibil ca valoarea minimă să se întâlnească o dată sau de mai multe ori. Dacă minimul apare de mai multe ori, înseamnă că două particule diferite au găsit două soluţii optime egale ca performanţe. Aceste soluţii pot fi identice, sau nu. Variabila `lid` poate fi, așadar, un vector cu mai multe elemente. Alegem prima poziție din vectorul `lid`, ca să rămână un singur lider.

- se schimbă liderul și funcția sa de adaptare, doar dacă funcția de adaptare din poziția `lid(1)` este mai mica decât cea precedentă, adică dacă am găsit un lider mai bun.

```
if fpbest(lid(1))<fgbest
    fgbest=fpbest(lid(1));
    gbest=pbest(lid(1),:);
end;
```

Privind populația inițială, putem observa că primele două particule au valoarea funcției de adaptare foarte apropiată, dar particulele sunt foarte diferite. Modificând o singură valoare dintr-o particulă, funcțiile de adaptare ar deveni egale. Asta înseamnă că este posibil să existe lideri foarte diferiți ca structură, dar egali ca funcție de adaptare, atunci când încă nu am ajuns la soluția optima globală. Doar unul dintre ei conduce cel mai rapid la această soluție. Alegerea `lid(1)` nu este neapărat cea mai bună, dar este cea mai simplă.

În ceea ce privește liderul inițial, acesta este particula cu cea mai bună (în problema noastră, cea mai mică) valoare a funcției de adaptare. Este vorba despre particula de pe poziția 8, cea marcată cu galben în populația inițială.

Actualizarea vitezei și a poziției particulelor din populație

Folosind cele mai noi poziții `pbest` ale fiecărei particule și cel mai nou lider al roiului, algoritmul modifică fiecare particulă din populație în doi pași:

- Recalculează viteza particulei
- Modifică poziția particulei

Formula matematică de recalculare a vitezei este:

$$v_{particula} = v_{particula} + c_1 \cdot random \cdot (pbest_{particula} - particula) + c_2 \cdot random \cdot (gbest_{roi} - particula)$$

În această formulă, peste valoarea existentă a vitezei ($v_{particula}$) se suprapun două componente:

- o deplasare către optimul personal al particulei
$$c_1 \cdot random \cdot (pbest_{particula} - particula)$$
- o deplasare către optimul global al roiului (către lider)
$$c_2 \cdot random \cdot (gbest_{roi} - particula)$$

random sunt vectori generați aleatoriu, cu valori între 0 și 1, în vreme ce c_1 și c_2 sunt constante numerice, pentru care valoarea recomandată este $c_1=c_2=2$.

Recalcularea poziției înseamnă adunarea vectorilor viteză și poziție anterioară:

$$particula = particula + v_{particula}$$

În prima iterație, deoarece $particula = pbest_{particula}$, componenta de deplasare către optimul personal se anulează și putem urmări cum particula este „trasă” către optimul global. Vom exemplifica în Matlab acest comportament scriind formula desfăcută pe componente. Folosim un ciclu `for`, pentru a actualiza întreaga populație, de la prima la ultima particulă.

```
for i=1:np % pentru fiecare individ din populatie
   % extrage din populatie particula  si viteza ei curenta
   p=pop(i,:);
   v=vit(i,:);

   % extrage particula pbest
   pbp=pbest(i,:);

   % vectorii random
   r1=rand(1,dp);
   r2=rand(1,dp);

   % recalculeaza viteza particulei
   v=v+2*r1.*(pbp-p)+2*r2.*(gbest-p);

   % recalculeaza pozitie particula
   p=p+v;
 end;
```

Folosind o întrerupere plasată pe linia de extragere a particulei din populație, putem vedea valorile pe care le iau variabilele `p, v, pbp, r1, r2` și putem reproduce și verifica operațiile realizate în Matlab.

Deoarece folosim funcția `random`, dacă veți reproduce calculul de mai sus, elementele generate în vectorii `r1` și `r2` în calculul dumneavoastră vor fi altele, viteza inițială va fi alta, iar particula rezultantă va fi alta.

În exemplul rulat pentru prima particulă din populația inițială, s-a obținut:

particula	p=	99	121	98	100	98	102	98	113
viteza	v=	0.003774	0.00862	0.0079	0.0014761	5.22E-06	0.005747	0.008908	0.008449
pbest	pbp=	99	121	98	100	98	102	98	113
gbest	gbest=	111	107	114	105	118	115	117	118
	r1=	0.123084	0.205494	0.146515	0.1890722	0.042652	0.635198	0.281867	0.538597
	r2=	0.695163	0.499116	0.535801	0.4451832	0.123932	0.490357	0.852998	0.873927
	calcul								
v	0.003774	0.00862	0.0079	0.001476	0.000005	0.005747	0.008908	0.008449	
+	+	+	+	+	+	+	+	+	
2	2	2	2	2	2	2	2	2	
*	*	*	*	*	*	*	*	*	rezultat nul,
r1	0.123084	0.205494	0.146515	0.189072	0.0426524	0.635198	0.281867	0.538597	se face înmulțire cu 0
*	*	*	*	*	*	*	*	*	
pbest-p	0	0	0	0	0	0	0	0	
+	+	+	+	+	+	+	+	+	
2	2	2	2	2	2	2	2	2	
*	*	*	*	*	*	*	*	*	
r2	0.695163	0.499116	0.535801	0.445183	0.1239323	0.490357	0.852998	0.873927	
*	*	*	*	*	*	*	*	*	
gbest-p	12	-14	16	5	20	13	19	5	
$\mathbf{v_{noua}}$ =	16.68769	-13.9666	17.15353	4.453308	4.9572963	12.75504	32.42284	8.747723	
$\mathbf{p_{noua}=p+v_{noua}}$	115.6877	107.0334	115.1535	104.4533	102.9573	114.755	130.4228	121.7477	
după rotunjire	116	107	115	104	103	115	130	122	

S-a folosit operatorul (.*) pentru a face înmulțire element cu element. Înmulțirea normală ar fi generat eroare, deoarece, după regulile înmulțirii matriceale, dimensiunile matricelor înmulțite n-ar fi corespuns restricției cunoscute de la matematică.

După recalcularea particulei, este necesară rotunjirea valorilor obținute în numere întregi, deoarece codurile ASCII sunt numere întregi.

Dacă examinăm valorile inițiale și finale ale elementelor particulei și le comparăm cu valorile elementelor particulei `gbest`, se poate vedea cum particula s-a apropiat de particula `gbest`, adică s-a deplasat în direcția particulei `gbest`.

particula inițială	99	121	98	100	98	102	98	113
gbest	111	107	114	105	118	115	117	118
finală	116	107	115	104	103	115	**130**	122

Cu excepția primului și penultimului element, elementele particulei finale sunt mai aproape sau chiar coincid cu elementele particulei `gbest`. Însă, la penultimul element, există și o altă problemă: valoarea obţinută este în afara intervalului de coduri ASCII admis, 97-122. Prin urmare, particula obținută este invalidă.

Cu particulele invalide se poate proceda în două feluri:

- Pot fi eliminate, caz în care în populația modificată va trece particula inițială
- Pot fi modificate, astfel încât toate elementele componente să respecte restricțiile impuse.

Aceasta înseamnă că, pentru unele probleme, particulele obținute prin mecanismul de actualizare PSO trebuie *validate* înainte de a fi integrate în populația pentru iterația următoare.

Dacă alegem să modificăm particulele invalide, putem face asta în cel puțin trei feluri:

- Limitând valoarea care depășește una dintre limite la valoarea limită admisă (adică limitând valorile care depășesc maximul admis la maximul admis, respectiv valorile care sunt sub minimul admis, la valoarea minimului admis)
- "Ricoșând" valoarea care depășește limita către interiorul domeniului admisibil (de exemplu, pentru valoarea 130 care depășește limita admisă cu 3 (127+3=130), noua valoare va fi 127-3=124)
- Pentru orice valoare care încalcă una dintre limite, inferioară sau superioară, să regenerăm elementul respectiv cu un altul, obținut aleatoriu, între limitele admise.

Pentru exemplul nostru, vom alege prima metodă. Pentru a valida o particulă, o parcurgem cu un ciclu `for` de la un capăt la altul și, dacă detectăm încălcarea unei limite, aducem valoarea depășită la limita maximă, respectiv minimă încălcată.

Abia după validare putem introduce particula nouă în populație, înlocuind particula veche.

Am folosit în `for`-ul pentru validare contorul de numărare `j`, deoarece trebuie să fie independent de contorul `i`, care numără particulele din populație care trec prin schimbarea vitezei și a poziției. Contorul `j` ia valori de la `1` la `dp` pentru fecare particulă `i`.

```
% validare
for j=1:dp % pentru toate lementele particulei modificate
    % limita inferioara
    if p(j)<linf
        p(j)=linf;
    end;
    % limita superioara
    if p(j)>lsup
        p(j)=lsup;
    end;
end;
```

Programul scris până în acest moment arată astfel:

```
% curatenie
clear;
clc;
```

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

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

% populatia initiala pbest
pbest=rand(np,dp);

% functiile de adaptare ale populatiei pbest
fpbest=10e6*ones(np,1);

% functia de adaptare initiala a liderului
fgbest=10e6;

% vitezele initiale
vit=0.01*rand(np,dp);

% 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;

% 3. calcul pbest si gbest
```

```
 % gaseste particulele care trebuie inlocuite
inloc=find(fadapt<fpbest);

% actualizeaza matricea pbest si vectorul fpbest
pbest(inloc,:)=pop(inloc,:);
fpbest(inloc,:)=fadapt(inloc,:);

% verifica daca s-a gast un noul lider gbest
lid=find(fpbest==min(fpbest));

% daca am gasit altul mai bun, inlocuiesc liderul
if pbest(lid(1))<fgbest
    fgbest=fpbest(lid(1));
    gbest=pbest(lid(1),:);
end;

for i=1:np % pentru fiecare individ din populatie
   % extrage particula din populatie viteza curenta a particulei
   p=pop(i,:);
   v=vit(i,:);

   % extrage particular pbest
   pbp=pbest(i,:);

   % vectorii random
   r1=rand(1,dp);
   r2=rand(1,dp);

   % recalculeaza viteza particulei
   v=v+2*r1.*(pbp-p)+2*r2.*(gbest-p);

   % recalculeaza pozitie particula
   p=p+v;

   % rotunjeste valori elemente particula la numere intregi
   p=round(p);

   % validare
   for j=1:dp % pentru toate lementele particulei modificate
       % limita inferioara
       if p(j)<linf
           p(j)=linf;
       end;
       % limita superioara
       if p(j)>lsup
           p(j)=lsup;
       end;
   end;

   % inlocuieste particula veche din populatie cu particula noua
   % si viteza veche cu viteza noua
   pop(i,:)=p;
   vit(i,:)=v;
end;
```

Deocamdată, programul face o iteraţie completă a algoritmului PSO, calculând funcţia de adaptare, `pbest`, `gbest`, şi noile poziţii ale particulelor care sunt validate înainte de a fi acceptate în noua populaţie. Următoarea interaţie înseamnă repetarea acestor paşi, cu populația modificată.

Ca exemplu (dumneavoastră veţi obţine altceva, din cauza componentei aleatorii), populaţia modificată după prima iteraţie arată astfel:

114	108	113	107	122	111	122	117
108	110	110	103	118	118	117	120
112	107	116	103	120	113	122	115
108	108	113	105	115	103	120	116
108	106	112	115	119	114	114	117
112	110	120	97	116	120	115	122
110	108	118	97	121	117	117	110
111	107	114	105	118	115	117	118
111	108	116	99	118	116	116	122
114	105	115	103	122	115	122	108

Exerciţii

Încercaţi să realizaţi celelalte două variante propuse pentru validarea particulelor.

Recapitulare

Am completat programul cu iniţializări şi am dus până la capăt o iteraţie a algoritmului PSO.

Va urma

Vom completa algoritmul ca să devină iterativ, vom afişa rezultatele numeric, literal şi grafic şi vom urmări în detaliu evoluția populației vreme de două iteraţii.