Inteligenţă artificială
Laboratorul 9

Algoritmi evolutivi

1. Specificul calculului evolutiv

Un *algoritm evolutiv* este o metodă de căutare prin analogie cu evoluţia biologică (de tip darwinist). Pentru găsirea soluţiei se utilizează o populaţie de soluţii potenţiale care evoluează prin aplicarea iterativă a unor operatori stohastici. Elementele populaţiei reprezintă soluţii potenţiale ale problemei. Pentru a ghida căutarea către soluţia problemei asupra populaţiei se aplică transformări specifice evoluţiei naturale:

- *Selecţie*. Elementele populaţiei care se apropie de soluţia problemei sunt considerate adecvate şi sunt favorizate în sensul ca au mai multe şanse de a supravieţui în generaţia următoare precum şi de a participa la generarea de "urmaşi".
- *Încrucişare*. La fel ca la înmulţirea din natură pornind de la două sau mai multe elemente ale populaţiei (numite părinţi) se generează noi elemente (numite urmaşi). În funcţie de calitatea acestora (apropierea de soluţia problemei) urmaşii îşi pot înlocui părinţii.
- *Mutaţie*. Pentru a asigura variabilitatea populaţiei se aplică, la fel ca în natură, transformări cu caracter aleator asupra elementelor populaţiei permiţând apariţia unor trăsături (gene) care doar prin încrucişare şi selecţie nu ar fi apărut în cadrul populaţiei.

În funcţie de modul în care este construită populaţia şi de modul în care este implementată evoluţia, sistemele de calcul evolutiv se încadrează în una dintre următoarele categorii:

- *Algoritmi genetici*. Se folosesc în special pentru rezolvarea unor probleme de optimizare discretă (combinatorială). Populaţia este reprezentată de stări din spaţiul problemei codificate binar (un element al populaţiei este un şir de biţi) iar principalii operatori sunt cei de încrucişare şi selecţie, cel de mutaţie având probabilitate mică de aplicare.
- *Strategii evolutive*. Au fost concepute iniţial pentru a rezolva probleme de optimizare continuă. Populaţia este constituită din elemente din domeniul de definiţie al funcţiei obiectiv. Operatorul principal este cel de mutaţie dar şi recombinarea este folosită. Pentru strategiile evolutive au fost dezvoltate scheme de adaptare a parametrilor de control (auto-adaptare).
- *Programare genetică*. Scopul programării genetice este dezvoltarea unor "modele" de calcul (programe simple). Astfel, populaţia este reprezentată de programe care candidează la rezolvarea problemei. Există diferite reprezentări ale elementelor populaţiei, una dintre cele mai clasice fiind aceea în care se utilizeaza o structură arborescentă pentru reprezentarea programelor. În anumite aplicaţii, cum este regresia simbolică, programele sunt de fapt expresii.

De exemplu, "programul-expresie" "a+b*c" poate fi descris prin (+ a (* b c)). O astfel de structură este poate fi uşor codifiată în Clips sau Lisp. Într-o astfel de reprezentare încrucişarea este realizată selectând aleator subarbori din arborele asociat programelor părinte şi interschimbându-le. Ca şi în cazul algoritmilor genetici mutaţia are pondere mică.

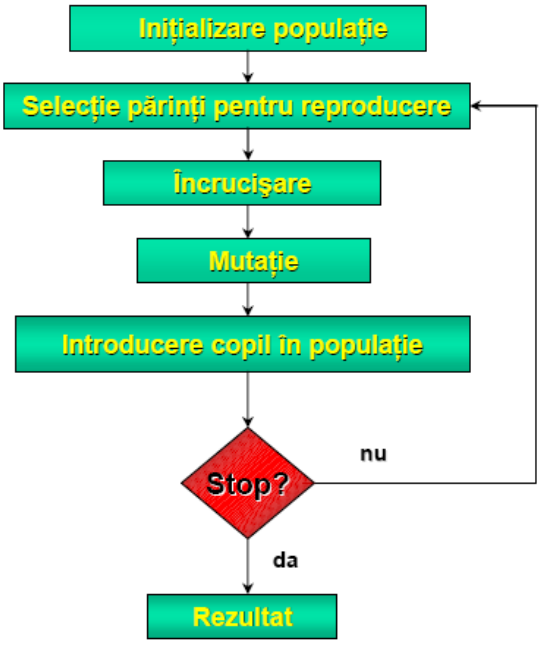
- *Programare evolutivă*. Iniţial programarea evolutivă a avut ca obiectiv dezvoltarea unor structuri de calcul (automate) printr-un proces de evoluţie în care operatorul principal este cel de mutaţie. Ulterior programarea evolutivă a fost orientată către rezolvarea problemelor de optimizare având aceeaşi sferă de aplicabilitate ca şi strategiile evolutive.

Toate aceste metode se bazează pe faptul că simulează evoluţia unei mulţimi (populaţie) de structuri informaţionale sub acţiunea unor procese similare celor din evoluţia naturală şi anume: selecţie, mutaţie şi încrucişare.

Acţiunea acestor procese este controlată prin intermediul unei funcţii de “fitness” care măsoară gradul de adaptare a fiecărui individ la mediul din care face parte. De exemplu, în cazul rezolvării unei probleme de optimizare (găsirea maximului sau minimului unei funcţii) funcţia de adaptare este chiar funcţia obiectiv a problemei.

2. Structura unui algoritm evolutiv

Majoritatea algoritmilor evolutivi au un caracter iterativ. Structura generală a unui astfel de algoritm (algoritm genetic, strategie evolutivă etc.) este:



Pseudocodul ar putea fi scris şi astfel:

```
t := 0; // iniţializarea indicatorului de iteraţie
init P(t); // iniţializarea (aleatoare) a populaţiei (mulţime de configuraţii a caror structură depinde de problemă)
evaluate P(t); // calculul funcţiei de adaptare pentru fiecare individ al populaţiei
repeat// proces iterativ de evoluţie
        t = t + 1// incrementarea indicatorului de iteraţie
        P1(t) = select P(t) // selectarea unei sub-populaţii în scopul reproduceri(părinţi)
        P2(t) = recombine P1(t) //încrucişarea "genelor" părinţilor având ca rezultat obţinerea unei noi populaţii
        P3(t) = mutate P2(t) // perturbarea aleatoare a noii populaţii (mutaţie)
        evaluate P3(t) // evaluarea noii populaţii(calculul funcţiei de adaptare pentru noua populaţie)
        P(t+1) = survive (P(t),P3(t)) // selecţia supravieţuitorilor din cele doua populaţii (pe baza valorii funcţiei de adaptare)
until <conditie de stop>
```

Codarea diferă de la problemă la problemă; cele mai des folosite metode fiind codarea binară, reală sau cu permutări.

Condiţia de oprire se poate referi la numărul de iteraţii (generaţii) sau la proprietăţile populaţiei curente (de exemplu, întreaga populaţie converge către o singură valoare).

3. Reguli de codificare. Exemple de probleme specifice

Modul în care o configuraţie este codificată într-un cromozom depinde de problema concretă. Există trei variante principale de codificare:

Codificare binară. Este varianta clasică. În acest caz cromozomii sunt vectori cu elemente din {0,1} iar spaţiul de căutare este $S = \{0, 1\}^n$, cu n numărul de gene (n este corelat cu dimensiunea problemei). Este adecvată pentru problemele de optimizare combinatorială în care configuraţiile pot fi specificate ca vectori binari.

Exemplul 1. Problema maximizării numărului de biţi egali cu 1 *(ONEMAX problem).* Se pune problema determinării şirului de biţi $(x_i,... ,x_n)$, $x_i \in \{0,1\}$ care maximizează funcţia

$$f : \{0, 1\}^n \longrightarrow N,\ f(x_1,... ,x_n) = \sum_{j=1}^{n} x_j$$

Problema este echivalentă cu a determina configuraţia de biţi care are cele mai multe elemente egale cu 1. Aceasta este evident (1, ..., 1) problema fiind una artificială folosită pentru a testa algoritmii genetici.

Exemplul 2. Problema submulţimii de sumă maximă limitată de un prag. Se consideră o mulţime $W = \{w_1,... ,w_n\}$ de valori întregi şi M o valoare întreagă. Se caută o submulţime $S \in W$ cu proprietatea că suma elementelor lui S este cât mai apropiată de M dar nu îl depăşeşte. Orice submulţime S poate fi reprezentată printr-un vector $(s_1,s_2,..., s_n)$ cu $s_i = 0$ dacă $w_i \notin S$ şi $s_i = 1$ dacă $w_i \in S$. Suma elementelor unei submulţimi S este în acest caz $\sum_{j=1}^{n} w_i s_j$

Exemplul 3. Problema rucsacului. Se consideră un set de n obiecte caracterizate prin greutăţile $(w_1,... ,w_n)$ şi prin valorile $(v_1,... ,v_n)$. Se pune problema determinării unui subset de obiecte pentru a fi introduse într-un rucsac de capacitate C astfel încât valoarea obiectelor selectate să fie maximă. O soluţie a acestei probleme poate fi codificată ca un şir de n valori binare în felul următor: $s_i = 1$ dacă obiectul i este selectat, respectiv $s_i = 0$ dacă obiectul nu este selectat.

Pentru 5 obiecte posibile de introdus în sac, o *genă* reprezintă un bit (x_i), în timp ce cromozomul este o soluţie potenţială (de exemplu 01101 înseamnă că articolele 2, 3 şi 5 vor fi incluse în sac).

Funcţia de adaptare / fitness indică cât de bună este o soluţie (cât este de adaptat un cromozom).

De exemplu, presupunem că la un moment dat avem combinaţia:

I	*1*	*2*	*3*	*4*	*5*
w_i	70	55	40	15	5
v_i	40	15	5	30	10

Atunci, pentru cromozomul: 01101, constrângerea este: $w = 55 + 40 + 5 = 100 \leq C = 100$ (ok), iar funcţia de adaptare: $v = 15 + 5 + 10 = 30$

Exemplul 4. Problema împachetării. Se consideră un set de *n* obiecte caracterizate prin dimensiunile (d_1, d_2,..., d_n) şi un set de *m* cutii având capacităţile (c_1, c_2,..., c_m). Se pune problema plasării obiectelor în cutii astfel încât capacitatea acestora să nu fie depăşită iar numărul de cutii utilizate să fie cât mai mic. O posibilă reprezentarea binară pentru această problemă este următoarea: se utilizează o matrice cu *n* linii şi *m* coloane iar elementul S_{ij} are valoarea 1 dacă obiectul *i* este plasat în cutia *j* şi 0 în caz contrar. Prin liniarizarea matricii se ajunge ca fiecare soluţie să fie reprezentată printr-un cromozom conţinând *mn* gene.

Exemplul 5. Optimizarea unei funcţii definite pe un domeniu continuu. Se consideră o funcţie $f : D = [a_1, b_1] \times [a_2, b_2] \times ... \times [a_n, b_n] \subset \mathbf{R}^n \rightarrow \mathbf{R}$ şi se caută $x^* = (x_1^*,..., x^*_n)$ care minimizează funcţia *f (f(x*) < f(x)* pentru orice $x \in D$). În acest caz codificarea binară nu este evidentă. Fiecare dintre componentele x_i ale lui *x* sunt transformate după cum urmează:

(i) se scalează pentru a fi aduse în intervalul [0,1): $v_i = (x_i - a_i) / (b_i - a_i)$;

(ii) se stabileşte numărul de biţi untilizaţi pentru reprezentare (*r*) şi se aduce valoarea v_i în mulţimea $\{0,1,..., 2^r - 1\}$: $u_i = [v_i * (2^r - 1)]$. Valoarea obţinută se reprezintă în baza 2. De exemplu, pentru $x = (1.25,2.3) \in [1,2] \times [2,3]$ şi $r = 5$ se obţine aproximativ $x = (0.25,0.3)$ iar cromozomul asociat va avea 2r = 10 componente binare: (0,0,1,1,1,0,1,0,0,1). Este evident că această codificare are caracter aproximativ, motiv pentru care în cazul variabilelor reale se preferă utilizarea codificării reale.

Utilizarea reprezentării binare în cazul în care configuraţiile corespunzătoare problemei sunt vectori de valori reale prezintă dezavantajul că valori reale aflate la distanţă mică au asociate reprezentări binare aflate la distanţă mare (diferă în multe poziţii binare). De exemplu reprezentarea lui 7 pe 4 biţi este 0111 iar reprezentarea lui 8 este 1000. Se remarcă faptul că trecerea de la reprezentarea lui 7 la cea a lui 8 necesită nu mai puţin de 5 complementari de biţi. O altă variantă de codificare care evită acest dezavantaj este codificarea de tip Gray caracterizată prin faptul că valori întregi succesive au asociate şiruri de biţi care diferă într-o singură poziţie.

Codificare reală. Este adecvată pentru problemele de optimizare pe domenii continue (vezi exemplul 5 de mai sus). In acest caz cromozomii sunt vectori cu elemente reale, fiind chiar elementele domeniului de definiţie al funcţiei (pentru exemplul de mai sus cromozomul este chiar $x = (1.25,2.3)$. Avantajul acestei reprezentări este faptul că este naturală şi nu necesită proceduri speciale de codificare/decodificare.

Codificare specifică. Se alege o variantă cât mai apropiată de specificul problemei.

Exemplul 6. Problema împachetării. În varianta de codificare binară apare dezavantajul că pot fi generate configuraţii care nu sunt fezabile. Acestea sunt de exemplu matricile care conţin mai multe valori egale cu 1 pe o linie (aceasta ar însemna că un obiect este simultan inclus în mai multe cutii). Pentru a evita astfel de situaţii se poate utiliza un alt tip de reprezentare: un vector cu n componente (s_1,..., s_n) în care $s_i \in \{1,..., m\}$ reprezintă cutia în care este inclus obiectul *i*.

Exemplul 7. Problema comis-voiajorului. Se consideră un set de n oraşe şi se pune problema găsirii unui traseu care să treacă o singură dată prin fiecare oraş şi care să aibă costul minim (dacă costul este proporţional cu lungimea traseului atunci se caută trasee de lungime minimă). O codificare naturală a unei configuraţii (traseu) este: (s_1,..., s_n) unde $s_i \in \{1, ... , n\}$ reprezintă numărul de ordine al oraşului vizitat la etapa *i* (la fiecare etapa comis-voiajorul se află într-un oraş). Pentru a fi respectată restricţia ca fiecare oraş să fie vizitat o singură dată este necesar ca elementele vectorului *s* să fie distincte ($s_i \neq s_j$ pentru orice $i \neq j$). Astfel fiecare configuraţie poate fi interpretată ca o permutare de ordin n, motiv pentru care această codificare este numită şi codificare de tip permutare.

4. Reguli de decodificare

Decodificarea asigură pregătirea evaluării configuraţiei.

Atenţie specială ridică decodificarea doar în cazul codificării binare a unor vectori cu valori reale (vezi exemplul 5 de la codificarea binară). În cazul codificării binare clasice a unei valori din intervalul [a, b] pentru a obţine valoarea decodificată pornind de la şirul de *r* biţi obţinut prin codificare se foloseşte relaţia:

$$\delta(s_1, \ldots, s_r) = a + \frac{b-a}{2^r - 1} \sum_{j=1}^{r} s_j 2^{j-1}$$

5. Construirea funcţiei de adaptare

Pentru a folosi analogia dintre procesele de căutare şi cele de evoluţie din natură este util să se reformuleze problemele de optimizare ca probleme de maximizare (orice problemă de minimizare poate fi transformată într-una de maximizare prin schimbarea semnului funcţiei obiectiv). Pentru o problemă de minimizare de forma: să se determine $x^* \in D$ cu proprietatea că $f(x^*) < f(x)$ pentru orice $x \in D$, funcţia de adaptare are fi $F(x) = -f(x)$. În cazul unei probleme de maximizare atunci funcţia de adaptare poate fi chiar funcţia obiectiv.

Exemplul 8. Problema ONEMAX. În acest caz funcţia de adaptare coincide cu funcţia obiectiv a problemei, calitatea unei soluţii fiind reflectată de numărul de componente egale cu 1.

Exemplul 9. Problema comis-voiajorului. În acest caz, dacă se foloseşte reprezentarea de tip permutare restricţia problemei (trecerea o singură dată prin fiecare oraş) este implicit satisfăcută. Dacă matricea *C* conţine costurile *(c(i, j)* reprezintă costul trecerii de la oraşul *i* la oraşul *j)* atunci funcţia de cost poate fi descrisă prin:

$$f(s_1, \ldots, s_n) = \sum_{i=1}^{n-1} c(s_i, s_{i+1}) + c(s_n, s_1).$$

În condiţiile în care costul trebuie minimizat, funcţia de adaptare va fi opusa funcţiei cost.

6. Selecţia

Selecţia are ca scop determinarea populaţiei intermediare ce conţine părinţii care vor fi supuşi operatorilor genetici de încrucişare şi mutaţie precum şi determinarea elementelor ce vor face parte din generaţia următoare. Criteriul de selecţie se bazează pe gradul de adaptare al configuraţiei la cerinţele problemei, exprimat prin valoarea funcţiei de adaptare.

Nu este obligatoriu ca atât părinţii cât şi supravieţuitorii se fie determinaţi prin selecţie, fiind posibil ca aceasta să fie folosită doar într-o singură etapă. De exemplu, toţi indivizii populaţiei curente sunt potenţiali părinţi dar după încrucişare şi mutaţie doar cei determinaţi prin procesul de selecţie vor supravieţui. Pe de altă parte este posibil ca părinţi să fie doar indivizii selectaţi iar toţi cei generaţi prin încrucişare şi mutaţie să fie transferaţi în noua generaţie.

Procesul de selecţie nu depinde de modul de codificare a elementelor populaţiei fiind însă legat de funcţia de adaptare.

Există mai multe metode de selecţie.

6.1. Elitismul

Această modalitate de selecţie presupune că cel mai adaptat individ este copiat direct în noua populaţie (mai rar primii câţiva cei mai adaptaţi). Aceasta asigură faptul că niciodată nu se va pierde soluţia cea mai bună.

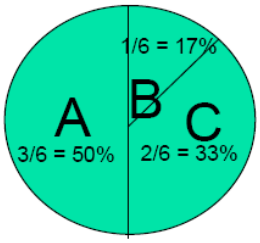
6.2. Selecţia de tip ruletă

Ideea de bază a acestui tip de selecţie este că indivizii mai adaptaţi au şanse mai mari. „Ruleta" se „învârte" de n ori pentru a se alege n indivizi.

fitness(A) = 3

fitness(B) = 1

fitness(C) = 2



Fie S suma tuturor funcţiilor de adaptare (ale tuturor indivizilor din populaţie).
-Se generează un număr aleatoriu N între 1 şi S
-Se returnează cromozomul a cărui funcţie de adaptare adăugată la suma parţială este ≥ N

-Cromozom:	1	2	3	4	5	6
-Fitness:	8	2	17	7	4	11
-Suma parţială:	8	10	27	34	38	49

-N ($1 \leq N \leq 49$): 23
- Selectat: 3

6.3. Selecţia pe baza rangurilor

Se ordonează crescător valorile funcţiei de adaptare pentru toate elementele populaţiei. Se reţin valorile distincte şi li se asociază câte un rang (cea mai mică valoare are rangul 1, iar cea mai mare are rangul maxim). Se partiţionează populaţia în grupuri de elemente care au aceeaşi valoare a funcţiei de adaptare şi li se asociază rangul corespunzător valorii. Presupunând că sunt *k* grupuri se construieşte distribuţia de probabilitate:

$$\begin{pmatrix} 1 & 2 & \dots & i & \dots & k \\ p_1 & p_2 & \dots & p_i & \dots & p_k \end{pmatrix} \qquad \text{cu } p_i = \frac{i}{\sum_{j=1}^{k} j}$$

Se generează câte un rang aleator în conformitatea cu distribuţia de mai sus (folosind metoda ruletei de exemplu) după care se selectează uniform aleator un element din grupul corespunzător rangului.

O altă modalitate de stabilire a probabilităţilor de selecţie pornind de la rangul elementelor este următoarea. Se ordonează crescător cele *m* elemente ale populaţiei şi i se asociază câte un rang (0 pentru elementul cu cel mai mic grad de adaptare şi *m* - 1 pentru elementul cu cel mai mare grad de adaptare). Probabilitatea de selecţie a elementului *i* se calculează astfel:

$$p_i = \frac{\alpha + (rang(i)/(m-1)) * (\beta - \alpha)}{m}$$

cu $\alpha < \beta$ valori alese astfel încât $\sum_{i=1}^{m} p_i = 1$, ceea ce implică $a + \beta = 2$. α poate fi interpretat ca numărul mediu de copii ale celui mai slab element iar β ca numărul mediu de copii ale celui mai bun element.

6.4. Selecţia de tip turnir

În acest caz se aleg aleatoriu *k* membri din populaţie şi se determină cel mai adaptat dintre aceştia. Procedura se repetă pentru a selecta mai mulţi părinţi. Această metodă este mai rapidă decât selecţiile prin ruletă sau ranguri.

Probabilitatea selectării individului *i* depinde de mărimea funcţiei de adaptare a lui *i*, precum şi de dimensiunea eşantionului *k*. Turnirul poate fi determinist sau probabilistic.

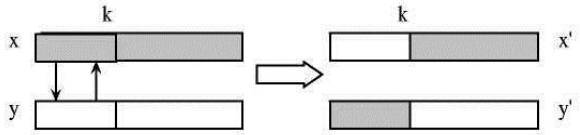
7. Încrucişarea

Încrucişarea permite combinarea informaţiilor provenite de la doi sau mai mulţi părinţi pentru generarea unuia sau mai multor urmaşi. Vom considera doar cazul a doi părinţi (notaţi cu *x* şi *y*) care generează doi urmaşi (notaţi cu *x'* şi y'). încrucişarea (numită uneori recombinare) depinde de modul de codificare a datelor aplicându-se direct asupra cromozomilor.

Variante specifice *codificării binare* (cromozomul este o succesiune de *n* cifre binare):

*Încrucişare cu un punct de tăietură***.** Se alege (aleator) un $k \in \{1,..., n-1\}$ numit punct de tăietură (încrucişare) şi se construiesc urmaşii în modul următor:

$$x' = (x_1,...,x_k,y_{k+1},...y_n) \quad \text{şi} \quad y' = (y_u \ldots ,y_k,x_{k+1},.. .x_n).$$



Încrucişare cu două sau mai multe puncte de tăietură se realizează asemănător cu cea de mai sus, doar că pot exista mai multe puncte de taiere.

*Încrucişare uniformă***.** La construirea fiecărui urmaş se selectează cu probabilitatea *p* o genă din primul părinte şi cu probabilitatea 1-p o genă din al doilea părinte. Cazul cel mai natural este cel în care $p = 0.5$.

*Încrucişare specifică codificării de tip permutare***.** În cazul codificărilor specifice şi operatorii de încrucişare şi mutaţie sunt specifici fiind de regulă bazaţi pe transformări cu caracter euristic sau pe transformări întâlnite în alte tehnici de rezolvare a problemei. De exemplu în cazul codificării de tip permutare trebuie ca încrucişarea să conserve specificul codificării.

Exemplul 10. Să considerăm problema comis voiajorului cu 7 oraşe (identificate prin A, B, C, D, E, F şi G) şi două configuraţii cu rol de părinţi: *(A,B,C,D,E,F,G)* şi *(D,C,F,E,A,B,G).* în acest caz încrucişarea cu un punct *(k* = 3) conduce la *(D,C,F,A,B,E,G)* respectiv *(A,B,C,D,F,E,G).* Primul fiu este obţinut preluând secvenţa formată din primele *k* = 3 oraşe din al doilea părinte, iar celelalte oraşe sunt tot din al doilea părinte însă amplasate în ordinea în care ele apar în primul părinte. în acest fel se asigură faptul că urmaşii conţin aceleaşi elemente însă amplasate în altă ordine decât cea din cadrul părinţilor.

Nu e necesar ca încrucişarea să se aplice întotdeauna, modul de aplicare putând fi controlat prin intermediul unei probabilităţi (numită probabilitate de încrucişare şi notată cu p_c). Valori uzuale pentru p_c sunt cuprinse între 0.2 şi 0.9.

8. Mutaţia

Asigură alterarea valorii unor gene pentru a evita situaţiile în care o anumită alelă nu apare în populaţie ca urmare a faptului că nu a fost generată de la început. În felul acesta este asigurată diversitatea populaţiei.

În cazul codificării binare cel mai simplu şi frecvent operator de mutaţie este cel în care se selectează aleator un cromozom, în cadrul acestuia se selectează o genă iar valoarea acesteia este modificată (0 devine 1 iar 1 devine 0). în funcţie de modul în care se selectează cromozomii şi genele există diverse variante.

De exemplu pentru fiecare cromozom se decide cu o anumită probabilitate (p_m, numită probabilitate de mutaţie) dacă el va fi supus mutaţiei sau nu. în caz afirmativ se selectează (uniform aleator) o genă şi valoarea acesteia se modifică. În felul acesta într-un cromozom poate fi modificată o singură genă.

O altă variantă este aceea în care toate genele se consideră ca făcând parte din aceeaşi structură şi pentru fiecare dintre ele se decide dacă va fi modificată sau nu. In felul acesta este posibil ca mai multe gene din cadrul unui cromozom să fie modificate.

În cazul codificării de tip permutare mutaţia cea mai simplă constă în schimbarea poziţiei a două elemente. De exemplu în cazul TSP de la configuraţia *(D, C, F,A,B, E, G)* se ajunge la configuraţia *(D, C, E,A,B, F, G)* prin interschimbarea elementelor de pe poziţiile 3 şi 6.

9. Domenii de aplicabilitate

Algoritmii evolutivi se utilizează în general pentru probleme de optimizare, atunci când nu există altă strategie de rezolvare a problemei şi este acceptat un răspuns aproximativ.

Câteva dintre domeniile de aplicabilitate sunt:

- *Planificare*. Majoritatea problemelor de planificare (alegerea traseelor optime ale unor vehicule, rutarea mesajelor într-o reţea, planificarea unor activităti etc.) pot fi formulate ca probleme de optimizare. Multe dintre acestea sunt probleme din clasa "NP-hard" necunoscându-se algoritmi de rezolvare care să aibă complexitate polinomială. Pentru astfel de probleme algoritmii evolutivi oferă posibilitatea obţinerii în timp rezonabil a unor soluţii sub-optimale de calitate acceptabilă.

- *Proiectare*. Algoritmii evolutivi sunt aplicaţi cu succes în proiectarea circuitelor digitale, a filtrelor dar şi a unor structuri de calcul cum sunt reţelele neuronale. Ca metode de estimare a parametrilor unor sisteme care optimizează anumite criterii se aplică în diverse domenii din inginerie cum ar fi: proiectarea avioanelor, proiectarea reactoarelor chimice, proiectarea structurilor în construcţii etc.

- *Simulare şi identificare*. Simularea presupune să se determine modul de comportare a unui sistem pornind de la un model al acestuia. Identificarea este sarcina inversă a identificării structurii sistemului pornind de la modul de comportare. Algoritmii evolutivi sunt utilizaţi atât în simularea unor sisteme din inginerie dar şi din economie (de exemplu pentru modelarea proceselor de competiţie în marketing). Identificarea unui model este utilă în special în efectuarea de predicţii în diverse domenii (economie, finanţe, medicină, ştiinţele mediului etc.)

- *Control*. Algoritmii evolutivi pot fi utilizaţi pentru a implementa controlere on-line asociate sistemelor dinamice (de exemplu pentru a controla roboţii mobili).

- *Clasificare*. Un sistem de clasificare se bazează pe o populaţie de reguli de asociere (reguli de producţie) care evoluează pentru a se adapta problemei de rezolvat (calitatea unei reguli se stabileşte pe baza unor exemple). Algoritmii evolutivi au fost utilizaţi cu succes în clasificarea imaginilor, în biologie (pentru determinarea structurii proteinelor) sau în medicină (pentru clasificarea electrocardiogramelor).

10. Aplicaţii

1. Să se minimizeze funcţia următoare (funcţia Rastrigin generalizată), pentru $n = 5$:

$$f(x) = nA + \sum_{i=1}^{n} x_i^2 - A\cos(\omega x_i).$$

Pentru $A = 10$ şi $\omega = 2\pi$, funcţia are minimul $f(0)=0$. Domeniul variabilelor este: $-5.12 < x_i < 5.12$.

Populaţia poate conţine 100 de indivizi iniţializaţi aleatoriu din domeniul de definiţie al variabilelor. Cromozomii vor fi vectori de 5 numere reale. Se va utiliza selecţia prin turnir cu 2 indivizi. Probabilitatea de încrucişare se va considera 90% iar probabilitatea de mutaţie 2%. Pentru mutaţie, se va reiniţializa gena cu o valoare din domeniul de definiţie. Se va utiliza elitismul.

Având în vedere că un algoritm evolutiv este o metodă euristică de optimizare, se va considera soluţie o aproximare suficient de fină a soluţiei exacte, de exemplu, cu o eroare maximă de 10^{-4}.

2. Să se rezolve problema rucsacului folosind codificarea binară.

Referinţe bibliografice

1. DUMITRACHE, I., N. CONSTANTIN, M, DRAGOICEA: Reţele neurale. Identificarea şi conducerea proceselor, Ed. Matrix Rom, 1999.
2. DUMITRESCU, D. Algoritmi genetici şi strategii evolutive – aplicaţii în inteligenţa artificială şi în domenii conexe, Edit. Albastră, 2000
3. ŞERBAN, S.: Modelare şi simulare, Ed. Printech Rom, 2007.
4. http://www.ici.ro/RRIA/ria2007_4/art02.html
5. http://web.info.uvt.ro/~dzaharie/cursnn1.pdf
6. http://web-ng.info.uvt.ro/~dzaharie/cdsa5.pdf
7. http://www.ipe.ro/RePEc/WorkingPapers/cs14-2.pdf