Inteligenţă artificială
Laboratorul 8

Algoritmi evolutivi

1. Obiective

Obiectivele acestui laborator sunt următoarele:

- prezentarea structurii generale a unui algoritm evolutiv, împreună cu operatorii specifici;
- rezolvarea a două probleme de optimizare bazate pe valori reale, folosind același algoritm.

2. Structura unui algoritm evolutiv

Probleme precum alegerea traseelor optime ale unor vehicule, rutarea mesajelor într-o rețea, planificarea unor activități, problema orarului etc., pot fi formulate ca probleme de optimizare. Multe dintre acestea sunt probleme NP-dificile, necunoscându-se algoritmi de rezolvare care să aibă complexitate polinomială. Pentru astfel de probleme, algoritmii evolutivi oferă posibilitatea obținerii în timp rezonabil a unor soluții sub-optimale de calitate acceptabilă.

Algoritmii evolutivi sunt aplicați cu succes în proiectarea circuitelor digitale, a filtrelor dar și a unor structuri de calcul precum rețelele 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 construcții etc. Uneori nici nu se cunoaște o expresie analitică a funcțiilor care trebuie optimizate, ci acestea se estimează din simulări. În aceste cazuri, nu pot fi aplicate ușor metodele de optimizare diferențială.

Un algoritm evolutiv este o metodă de optimizare prin analogie cu evoluția biologică. Pentru găsirea soluției, se utilizează o populație de soluții potențiale, care evoluează prin aplicarea iterativă a unor operatori stohastici. Elementele populației reprezintă soluții potențiale ale problemei.

Pentru a ghida căutarea către soluția problemei, asupra populației se aplică transformări specifice evoluției naturale:

- *Selecția*. Determină părinții care se vor reproduce pentru a forma următoarea generație. Indivizii mai adaptați din populație (care se apropie mai mult de soluția problemei) sunt favorizați, în sensul ca au mai multe șanse de reproducere;
- *Încrucișarea*. Pornind de la doi părinți, se generează copii;
- *Mutația*. Pentru a asigura diversitatea populației, se aplică transformări cu caracter aleatoriu asupra copiilor nou generați, permițând apariția unor trăsături care nu ar fi apărut în cadrul populației doar prin selecție și încrucișare.

Structura generală a unui astfel de algoritm este cea din figura 1.

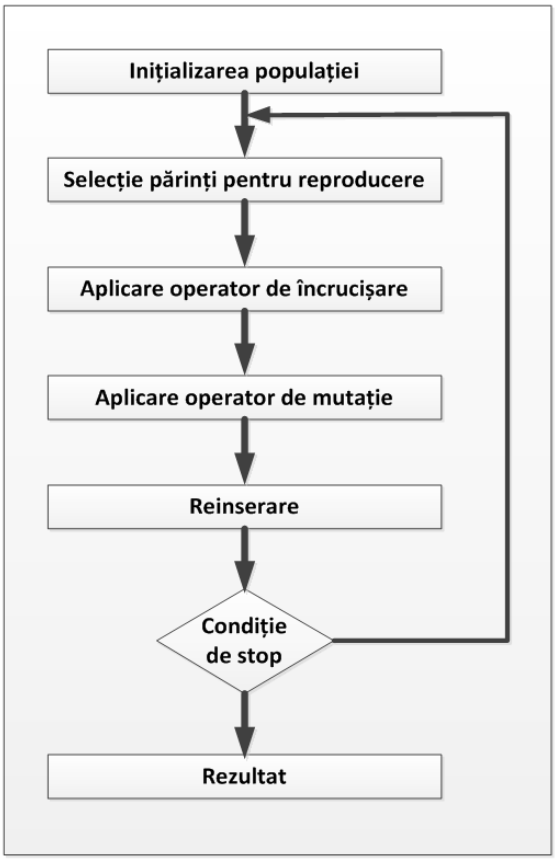


Figura 1. **Structura generală a unui algoritm evolutiv**

Codificarea diferă de la problemă la problemă, cele mai des folosite metode fiind codificările binară și reală. Condiția de stop se poate referi la evoluția pentru un anumit număr de generații sau la proprietățile populației curente, de exemplu convergența populației către o anumită valoare, pierderea diversității indivizilor din populație etc.

Dimensiunea populației poate fi în jur de 50. Pentru probleme simple poate fi mai mic (30), iar pentru probleme dificile poate fi mai mare (100). Dacă sunt prea puțini cromozomi, algoritmul nu are diversitatea necesară găsirii soluției. Dacă sunt prea mulți, algoritmul va fi mai lent, fără a se îmbunătăți însă calitatea soluției.

Numărul maxim de generații variază de obicei între 100-1000.

3. Tipuri de codificare

Modul în care o posibilă soluție este codificată într-un cromozom depinde de problema concretă care trebuie rezolvată.

Codificarea binară

Este varianta clasică a algoritmilor genetici. În acest caz, cromozomii sunt șiruri de *n* biți, *n* fiind numărul de gene. Este adecvată pentru problemele de optimizare combinatorială în care configurațiile pot fi specificate ca vectori binari.

Exemplul 3.1. Problema rucsacului

Se consideră o mulțime de n obiecte caracterizate prin greutățile (w_1, ... , w_n) și prin valorile (v_1, ... , v_n). Se pune problema determinării unei submulțimi de obiecte pentru a fi introduse într-un rucsac de capacitate C astfel încât valoarea obiectelor selectate să fie maximă. O soluție a acestei probleme poate fi codificată ca un șir 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 soluție potențială (de exemplu 01101 înseamnă că articolele 2, 3 și 5 vor fi incluse în sac).

Funcția de adaptare (engl. "fitness") arată cât de bună este o soluție, cât este de adaptat un cromozom.

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

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 satisfăcută: $w = 55 + 40 + 5 = 100 \leq C = 100$, iar funcția de adaptare este: $F = 15 + 5 + 10 = 30$.

Exemplul 3.2. Problema împachetării

Se consideră un set de n obiecte caracterizate prin dimensiunile (d_1, d_2, ..., d_n) și o mulțime de m cutii având capacitățile (c_1, c_2, ..., c_m). Se pune problema plasării obiectelor în cutii astfel încât capacitatea acestora să nu fie depășită, 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 și m coloane iar elementul S_{ij} are valoarea 1 dacă obiectul i este plasat în cutia j și 0 în caz contrar. Prin liniarizarea matricii se ajunge ca fiecare soluție potențială să fie reprezentată printr-un cromozom cu $m \cdot n$ gene.

Codificarea reală

Este adecvată pentru problemele de optimizare pe domenii continue. În acest caz, cromozomii sunt vectori cu elemente reale din domeniul de definiție al funcției. Avantajul acestei reprezentări este faptul că este naturală și nu necesită proceduri speciale de codificare/decodificare.

Exemplul 3.3. Funcția Rastrigin generalizată

Presupune minimizarea funcției următoare:

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

Pentru $A = 10$ și $\omega = 2\pi$, funcția are minimul $f(\mathbf{0}) = 0$.

Domeniul variabilelor este: $-5.12 < x_i < 5.12$.

În acest caz, cromozomii vor fi vectori cu n elemente din acest interval.

4. Construirea funcţiei de adaptare

Funcția de adaptare definește problema de optimizare. În formularea standard, scopul algoritmilor evolutivi este maximizarea acestei funcții.

Orice problemă de minimizare poate fi transformată într-una de maximizare prin schimbarea semnului funcției obiectiv. Pentru o problemă de minimizare în care dorim minimizarea unei funcții $g(x)$, funcția de adaptare va fi $F(x) = -g(x)$.

5. Selecţia

Selecția are ca scop determinarea părinților care vor genera elementele populației din următoarea generație. Criteriul de selecție se bazează pe valoarea funcției de adaptare. Procesul de selecție nu depinde de modul de codificare a elementelor populației (binar, real etc.).

Selecţia de tip ruletă

Ideea de bază a acestui tip de selecție este că probabilitatea unui individ de a fi selectat este proporțională cu funcția sa de adaptare. Ruleta se învârte de n ori pentru a se alege n indivizi.

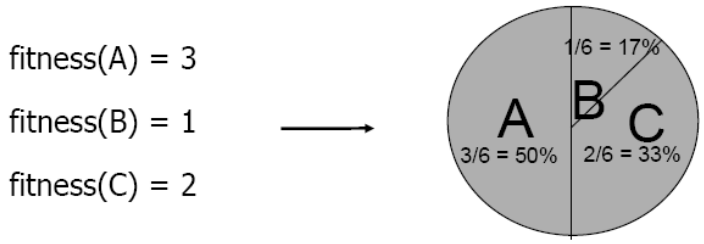


Figura 2. **Selecţia de tip ruletă**

Selecţia bazată pe ranguri

Se ordonează crescător valorile funcției de adaptare pentru toate elementele populației. Se reţin valorile distincte şi li se asociază câte un rang: cea mai mică valoare are rangul 0 sau 1, iar cea mai mare are rangul maxim, n sau $n - 1$, unde n este dimensiunea populației. Probabilitatea de selecție este în acest caz proporțională cu rangul individului.

Selecţia de tip competiţie

Se aleg aleatoriu k membri din populație și se determină cel mai adaptat dintre aceștia. De obicei, $k = 2$. Procedura se repetă pentru a selecta mai mulți părinți. Această metodă este mai rapidă decât selecțiile prin ruletă sau ranguri.

Elitismul

Cel mai adaptat individ este copiat direct în noua populație. Asigură faptul că niciodată nu se pierde soluția cea mai bună.

6. Încrucișarea

Încrucișarea (engl. "crossover") permite combinarea informațiilor provenite de la doi părinți pentru generarea a 1 sau 2 copii. Vom considera doar cazul a doi părinți (notați cu **m** și **f**, de la "mother" și "father") care generează doi copii (notați cu $\mathbf{c}_1$ și $\mathbf{c}_2$). Încrucișarea (numită uneori și recombinare) depinde de modul de codificare al datelor, aplicându-se direct asupra cromozomilor.

Încrucișarea binară

Cea mai des folosită este *încrucișarea cu un punct*. Reamintim că aici cromozomul este un șir de n biți. Se alege în mod aleatoriu un punct de tăietură $k \in \{ 1, ..., n-1 \}$ și se construiesc copiii astfel:

$$\mathbf{c}_1 = (m_1, ..., m_k, f_{k+1}, ..., f_n)$$
$$\mathbf{c}_2 = (f_1 ... , f_k, m_{k+1}, ..., m_n)$$

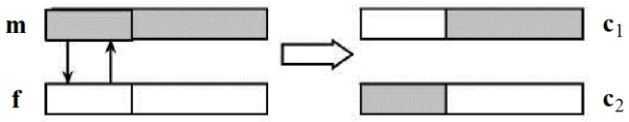


Figura 3. **Încrucișarea binară cu un punct**

Încrucișarea reală

Încrucișarea aritmetică combină materialul genetic al părinților **m** și **f** (vectori de numere reale) aplicând următoarele reguli pentru a obține, de exemplu, un copil **c**:

$$c_i = a \cdot m_i + (1 - a) \cdot f_i$$

În mod analog cu încrucișarea binară, se pot obține simultan doi copii în loc de unul singur.

Valoarea a este un număr aleatoriu din intervalul (0, 1) sau (–0.25, 1.25). În ultimul caz, copiii se vor găsi în interiorul unui hipercub puțin mai mare decât hipercubul delimitat de valorile genelor părinților. Această metodă este utilă în cazul în care domeniul de definiție al genelor nu poate fi estimat foarte bine și trebuie extins în mod dinamic pe parcursul execuției algoritmului.

Astfel, copilul va fi plasat pe un segment de dreaptă determinat de genele părinților.

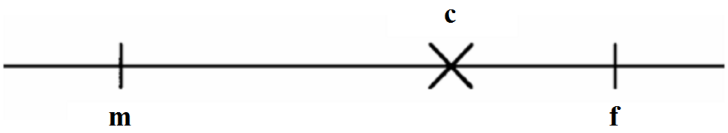


Figura 4. **Încrucișarea reală aritmetică**

Încrucișarea are loc cu o anumită probabilitate, numită rată de încrucișare și notată cu p_c. Aceasta trebuie să fie în general mare, de exemplu 0.9.

Din punct de vedere al implementării, se generează un număr aleatoriu r între 0 și 1. Dacă $r < p_c$, se aplică încrucișarea. Dacă nu, se generează un copil egal cu un parinte.

7. Mutația

Mutația asigură modificarea valorilor unor gene pentru a evita situațiile în care o anumită alelă (valoare posibilă a unei gene) nu apare în populație deoarece nu a fost generată de la început. În felul acesta poate fi crescută diversitatea populației și se poate evita convergența într-un optim local.

Mutația se efectuează asupra copiilor generați după aplicarea operatorului de încrucișare.

Mutația binară

Cel mai simplu și des utilizat tip de mutație binară este cel în care se selectează o genă iar valoarea acesteia este negată (0 devine 1 iar 1 devine 0) cu o anumită probabilitate, numită rată de mutație și notată cu p_m. Aceasta trebuie să fie în general mică, de exemplu 0.02.

Mutația reală

Pentru codificarea reală, se folosește de obicei resetarea valorii unei gene la un număr aleatoriu din domeniul ei de definiție. Rata de mutație reală poate fi mai mare decât la mutația binară, deoarece numărul valorilor reale este mai mic decât numărul de biți, de exemplu 0.1.

Pentru explicații suplimentare, consultați cursul 4.

8. Aplicații

8.1. Determinați soluția reală a ecuației: $x^5 - 5x + 5 = 0$. (Soluția reală este: $x = -1.680494$.)

Se va acorda un bonus de 0.25 p pentru determinarea în plus și a soluțiilor complexe. În acest scop, trebuie modificată corespunzător funcția de adaptare. (Soluțiile complexe sunt: $1.05 \pm 0.3i$ și $-0.2 \pm 1.57i$.)

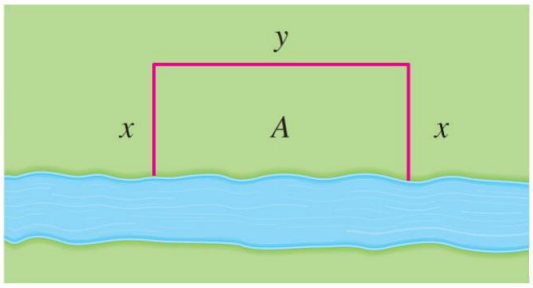
Indicații. Trebuie să exprimăm cerința sub forma unei probleme de optimizare, adică de găsire a maximului (sau minimului) unei funcții. Dacă trebuie să determinăm x astfel încât $f(x) = y$, înseamnă că trebuie să minimizăm diferența dintre $f(x)$ și y: $\min |f(x) - y|$. Minimul așteptat ar trebui să fie 0. Pentru ecuația noastră, va trebui să determinăm: $\min |x^5 - 5x + 5|$. Deoarece algoritmul evolutiv standard găsește maximul unei funcții, nu minimul ei, trebuie să transformăm problema într-una de maximizare: $\max -|x^5 - 5x + 5|$. Prin urmare, funcția de adaptare corespunzătoare problemei este: $F(x) = -|x^5 - 5x + 5|$.

Pentru că toți operatorii genetici presupun lucrul cu numere aleatorii, reamintim două metode utile din clasa `Random`. Fie obiectul `Random rand = new Random()`. Atunci:

- `rand.Next(max)` returnează un număr natural din mulțimea $\{ 0, ..., max - 1 \}$;
- `rand.NextDouble()` returnează un număr real din intervalul $[0, 1)$.

8.2. Un fermier are 100 m de gard cu care vrea să împrejmuiască un teren dreptunghiular care se învecinează cu un râu. Latura vecină cu râul nu are nevoie de gard. Care sunt dimensiunile terenului cu arie maximă care poate fi împrejmuit în acest fel?

Problema de optimizare formalizată este maximizarea ariei $A = x \cdot y$, respectând constrângerea $2x + y = 100$. (Soluția este: $x = 25$ și $y = 50$, deci $A = 1250$).



Indicații. Aici funcția de adaptare este direct cea exprimată de problemă: $F(x, y) = x \cdot y$. Dificultatea apare din existența constrângerii. Întrucât foarte puțini indivizi ar putea respecta constrângerea, o soluție inteligentă este „repararea" acestora, adică forțarea respectării constrângerii. De exemplu, un individ are genele x și y, care nu respectă constrângerea. Genele sale se pot scala proporțional cu un factor r, păstrând informațiile utile pentru căutare și diversitate genetică:

$$r = 100 / (2x + y)$$
$$x' = x \cdot r$$
$$y' = y \cdot r$$

Astfel, $2x' + y'$ va fi întotdeauna 100 (constrângerea va fi satisfăcută de toți indivizii), iar optimizarea se va face după funcția de adaptare $F(x', y') = x' \cdot y'$.

Repararea genelor presupune faptul ca în metoda corespunzătoare funcției de adaptare să se modifice efectiv genele unui cromozom din vechile valori x și y în noile valori scalate x' și y'.

Se dă un prototip de aplicație, în care sunt implementate toate clasele necesare pentru rezolvare. Folosind variantele *reale* ale operatorilor genetici, implementați următoarele metode:

Metoda Tournament(Chromosome[] population) din clasa Selection

Alege în mod aleatoriu doi indivizi/cromozomi diferiți din populație, le compară valorile funcției de adaptare și îl returnează pe cel mai adaptat.

Metoda GetBest(Chromosome[] population) din clasa Selection

Returnează individul din populație cu funcția de adaptare maximă. Nu returnați referința la cel mai bun individ din populație, ci creați un obiect nou, copie a celui mai bun individ.

Metoda Arithmetic(Chromosome m, Chromosome f, double rate) din clasa Crossover

Aplică încrucișarea artimetică între cei doi părinți, cu probabilitatea rate.

Metoda Reset(Chromosome child, double rate) din clasa Mutation

Aplică asupra copilului child mutația prin resetare, cu probabilitatea rate.

Metoda Solve(...) din clasa EvolutionaryAlgorithm

Trebuie completată bucla principală, cu apelul operatorilor genetici.

Metoda ComputeFitness(Chromosome c) din clasa Equation

Funcția de adaptare pentru prima problemă.

Metoda ComputeFitness(Chromosome c) din clasa Fence

Funcția de adaptare pentru a doua problemă. Aici trebuie satisfăcută și constrângerea. Fezabilitatea soluțiilor se poate asigura prin repararea cromozomilor.

Metoda Main() din clasa Program

Pentru ambele probleme, trebuie să găsiți, prin încercări repetate, valorile parametrilor algoritmului (dimensiunea populației, numărul maxim de generații, ratele de încrucișare și mutație) care dau cele mai bune rezultate.

Diagrama UML de clase a aplicației complete este următoarea: