

Programare evolutivă și programare genetică

noiembrie 2003

1 Introducere

Programarea evolutivă (PE) și cea genetică (PG) sunt ramuri ale calculului evolutiv al căror scop este evoluția unor "structuri de calcul" (o structura de calcul poate fi un program, un automat finit, o rețea neuronală etc). Principala caracteristică a acestora este faptul că elementele populației nu mai sunt șiruri de valori (binare sau reale) cum se întâmplă la algoritmi genetici (AG) și strategiile evolutive (SE) ci structuri mai complicate (dependente de problema de rezolvat) care necesită operatori adecvați.

Din punct de vedere cronologic, inițial a fost dezvoltată programarea evolutivă (în anii 1960-1970), programarea genetică fiind dezvoltată abia începând cu 1990. Din punct de vedere al afinității cu celelalte categorii de algoritmi evolutivi, programarea evolutivă este mai apropiată de strategiile evolutive (utilizează mutația a operator primar, recombinația fiind rar sau deloc folosită) pe când programarea genetică este mai apropiată de algoritmi genetici (operatorul primar este cel de încrucișare).

Din punct de vedere al structurii generale, în PE și PG se efectuează aceleași prelucrări ca la AG și SE:

- inițializarea populației;
- proces iterativ în care se aplică succesiv: evaluarea populației, selecția părinților, aplicarea operatorilor de variație (incrușare și/sau mutație).

2 Programare evolutivă

A fost inițiată în 1960 de către L. Fogel ca o tehnică de generare automată a unui comportament inteligent, văzut ca abilitatea unui sistem artificial de a realiza predicții privind mediul informațional în care se află sistemul. Sistemele sunt modelate prin automate finite (tranziția către a nouă stare a automatului și simbolul pe care îl emite fiind determinată de starea curentă și un simbol de intrare) iar mediul informațional este reprezentat de o succesiune de simboluri de intrare. Comportamentul automatului este considerat inteligent dacă poate prezice simbolul următor. Elementele populației sunt diagramele de tranziție iar gradul de adecvare a unui element este cu atât mai mare cu cât șirul de simboluri produs de automat este mai apropiat de un șir țintă.

O altă direcție de dezvoltare a PE o reprezintă cea orientată către rezolvarea problemelor de optimizare numerică. Această direcție este foarte apropiată de strategiile evolutive cu excepția următoarelor diferențe [2]:

1. Nu se aplică recombinație.

2. Numărul de descendenți coincide cu numărul de părinți ($\mu = \lambda$).
3. În scopul selecției se calculează pentru fiecare element, x^i , un grad de adecvare $F(x_i) > 0$ obținut prin scalarea funcției obiectiv $f(x^i)$ și eventual prin perturbarea acesteia cu o valoare aleatoare: $F(x^i) = G(f(x_i) + \nu_i)$ unde ν_i este un vector care realizează perturbarea iar G este o funcție ce asigură scalarea.
4. Determinarea celor μ supraviețuitori dintre cele 2μ elemente ale populațiilor concatenate se face prin selecție de tip turneu: pentru fiecare element x_i al populației reunite se selectează aleator r elemente dintre cele 2μ și se calculează contorul $0 \leq q_i \leq r$ care reprezintă numărul de elemente selectate ce sunt mai puțin bune decât x^i ; Se ordonează elementele populațiilor descrescător după valoarea contorului și se rețin pentru generația viitoare primele μ .
5. Mutația unui element, x^i , se bazează pe adăugarea unui vector aleator ale cărui componente sunt variabile aleatoare normal repartizate cu medie 0 și abatere standard $\sigma_j = \sqrt{\beta_j F(x^i) + \gamma_j}$ ($j = \overline{1, n}$) cu β_j și γ_j parametri specifici fiecărei componente.
6. Varianta auto-adaptivă a PE presupune utilizarea unor elemente extinse, $(x_1, \dots, x_n, \sigma_1, \dots, \sigma_n)$, și modificarea parametrului σ_j prin: $(\sigma'_j)^2 = (\sigma_j)^2 + \sqrt{\zeta} \sigma_j z_j$ cu z_j variabilă aleatoare cu repartiția normală standard iar ζ este un parametru de control. Spre deosebire de auto-adaptarea de la SE unde perturbarea log-normală asigură pozitivitatea parametrilor σ_i auto-adaptarea de mai sus nu mai prezintă acest avantaj, motiv pentru care trebuie utilizată o regulă de "reparare" (dacă se obțin valori negative pentru σ_i atunci se înlocuiește cu o valoare pozitivă suficient de mică).

În continuare sunt prezentate particularități ale PE utilizate pentru generarea de automate finite deterministe.

2.1 Reprezentarea elementelor populației

Diagrama de tranziții a unui automat finit determinist (AFD) este de fapt un multi-graf orientat etichetat în care nodurile sunt etichetate cu stările automatului iar arcele marchează tranzițiile și sunt etichetate cu simbolul de intrare și cel de ieșire corespunzătoare.

Să considerăm un AFD simplu [3] care permite verificarea dacă un șir de biți conține un număr par/impar de poziții egale cu 1 (problema parității). Alfabetul de intrare va conține simbolurile $\{0, 1\}$ iar simbolul de ieșire (produs la fiecare tranziție) va fi 0 (număr par de cifre egale cu 1) respectiv 1 (număr impar de cifre egale cu 1). Mulțimea stărilor va fi constituită din două elemente: $\{par, impar\}$ iar starea inițială este cea cu *par*. Rezultatul furnizat de AFD este dat atât de starea finală cât și de ultimul simbol transmis. Diagrama de tranziții asociată este prezentată în figura 1.

Diagrama de tranziții poate fi reprezentată printr-o structură de date adecvată unui graf, de exemplu prin liste de adiacență.

2.2 Funcția de adecvare

Gradul de adecvare al unui element al populației se determină prin simularea funcționării automatului pentru un set de șiruri binare de test și calculul ratei de succes (raportul dintre numărul de simulări corecte și numărul total de simulări). Un element este cu atât mai bun cu cât rata de succes este mai mare. O problemă importantă o reprezintă alegerea setului de exemple de test (nu întotdeauna este posibil să se testeze toate variantele posibile).

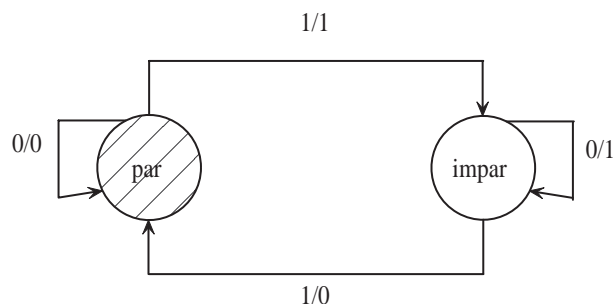


Figura 1: Automat finit determinist pentru problema parității

2.3 Operatori genetici

PE folosește doar mutație care este aplicată asupra fiecărui element al populației curente generând o populație de urmași (numărul de urmași este egal cu numărul de părinți). *Selecția* supraviețuitorilor constă în reținerea celor mai bune elemente dintre părinți și urmași.

Mutația trebuie să genereze pornind de la o diagramă de tranziție, tot o diagramă de tranziție, astfel că operatori adecvați sunt: modificarea unui simbol de ieșire, adăugarea/eliminarea unei stări, modificarea unei tranziții (prin schimbarea stării finale), modificarea stării inițiale. Pentru fiecare element al populației se alege uniform aleator unul dintre cei cinci operatori. Este însă posibil ca asupra aceluiași element să se aplice mai mulți operatori de mutație.

Variante ale AFD din fig. 1, obținute prin mutație, sunt ilustrate în fig. 2.

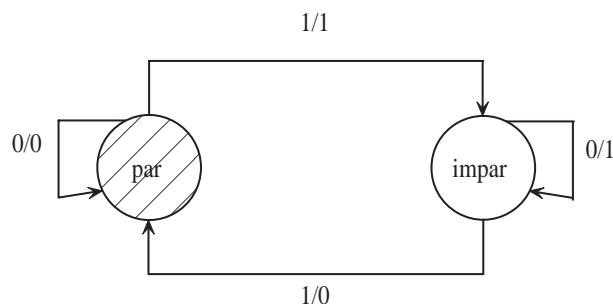


Figura 2: (a) Automatul obținut prin modificarea simbolurilor de ieșire la trecerea de la starea *par* la starea *impar* și invers; (b) Automatul obținut prin modificarea tranziției din starea *par* la primirea simbolului 0

Aplicând mutația asupra unei diagrame de tranziție se obține tot o diagramă de tranziție însă nu neapărat una minimală (pot exista stări care nu pot fi atinse prin nici o tranziție). Efectul structural al mutației depinde gradul de modificare a structurii diagramei de tranziție: modificarea unui simbol de ieșire are un efect mai scăzut decât adăugarea sau eliminarea unei stări.

2.4 Aplicații

Programarea evolutivă este folosită pentru optimizare numerică continuă (clasă de probleme abordabilă și prin strategii evolutive), dezvoltarea sistemelor de clasificare, antrenarea rețelelor neuronale, proiectarea unor sisteme de control ce pot fi modelate prin automate finite, controlul deplasării roboților, dezvoltarea unor strategii.

3 Programare genetică

Reprezintă cea mai nouă direcție dezvoltată în cadrul calculului evolutiv de către Koza (în jurul anilor 1990). Programarea genetică este de fapt o variantă a algoritmilor genetici care operează cu populații constituite din "structuri de calcul" (din acest punct de vedere fiind similară programării evolutive).

A fost dezvoltată inițial cu scopul de a genera automat programe care să rezolve (aproximativ) anumite probleme. Ulterior aria de aplicații s-a extins către *proiectarea evolutivă*[1] un domeniu aflat la intersecția dintre calculul evolutiv, proiectarea asistată de calculator și biomimetism (subdomeniu al biologiei care studiază procesele imitative din natură). Programarea genetică urmează structura generală a unui algoritm genetic folosind încrucișarea ca operator principal iar mutația ca operator secundar.

Particularitățile PG sunt legate de modul de reprezentare a indivizilor, fapt ce necesită și alegerea adecvată a operatorilor.

3.1 Reprezentarea elementelor populației

Considerăm cazul clasic al evoluției unor "programe" cu rol de structuri de calcul. Acestea nu sunt văzute ca succesiuni de linii de cod ci ca arbori de derivare asociați "cuvântului" pe care îl reprezintă în limbajul formal asociat limbajului de programare utilizat. În practică se lucrează cu limbaje restrânse bazate pe un set precizat de simboluri asociate variabilelor și un set de operatori (asociați atât operațiilor aritmetice, logice cât și prelucrărilor de decizie sau de ciclare, dacă este necesar). În aceste condiții orice program este de fapt o expresie în sens general. Partea care este strâns legată de problema de rezolvat este cea a stabilirii setului de simboluri și de operatori. Această alegere determină esențial rezultatele ce se pot obține prin programare genetică, însă nu există reguli generale care să stabilească legătura dintre o problemă și un set de simboluri și operatori, rolul cel mai important revenindu-i programatorului.

Koza a propus ca modalitate de reprezentare scrierea prefixată a expresiilor (cum este cea folosită în limbajul LISP unde o expresie descrisă infixat prin $(a + b) * c$ poate fi specificată prin $(*(+ab)c)$). Este ușor de observat că scrierea prefixată corespunde parcurgerii în preordine a arborelui de structură al expresiei.

Pentru a simplifica descrierea considerăm că se operează cu "programe" ce sunt expresii ce conțin operatori aritmetici, relaționali și logici precum și apeluri ale unor funcții matematice (trigonometrice, exponențială, logaritmică, extragere rădăcină pătrată etc.). În acest caz, limbajul formal asociat este independent de context și fiecărui cuvânt (expresie) i se poate asocia un arbore de derivare. Nodurile interne ale arborelui sunt etichetate cu operatori sau nume de funcții iar cele terminale sunt etichetate cu nume de variabile sau valori constante.

Operarea cu expresii este adecvată pentru rezolvarea problemelor de *regresie simbolică*. În aceste probleme se pornește de la cunoașterea câtorva valori pentru două mărimi X și Y și se caută o *expresie* care să descrie dependența analitică dintre cele două mărimi. Este de fapt o problemă de aproximare, numai că spre deosebire de *regresia numerică* unde se presupune cunoscută *forma*

dependenței și se determină doar valorile parametrilor, în regresia simbolică se prestabilește doar un set de operatori și funcții ce vor interveni eventual în expresie.

În momentul în care se alege reprezentarea indivizilor pentru o problemă concretă trebuie să se stabilească:

- Simbolurile ce vor fi utilizate pentru a eticheta nodurile neterminale. Acestea sunt de fapt operatorii și funcțiile ce vor fi folosite în construirea expresiilor. De exemplu pentru regresia simbolică ar fi utili operatorii aritmetici (+, -, *, /, **) și funcțiile elementare (sqrt, exp, log, sin, cos).
- Simbolurile ce vor fi utilizate pentru a eticheta nodurile terminale. Acestea sunt de regulă variabilele ce intervin în problemă, constante matematice (π , e) sau constante aleatoare (specificate printr-o funcție de generare a numerelor aleatoare)

Dacă setul de simboluri folosite pentru specificarea variabilelor este $\{a, b, c\}$, operatorii sunt $\{+, *\}$ și se utilizează funcția SIN atunci un exemplu de arbore asociat unei expresii este cel din fig. 3

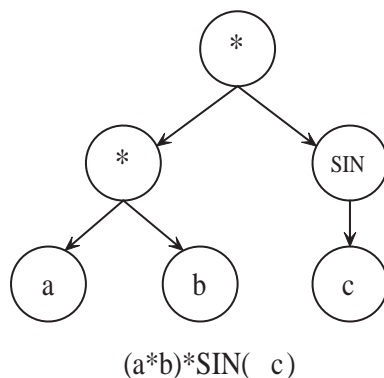


Figura 3: Arbore asociat unei expresii aritmetice

Din punct de vedere al implementării o variantă o reprezintă utilizarea limbajului LISP care prezintă avantajul că expresiile sunt descrise prin liste asupra cărora pot fi efectuate simplu transformările impuse de operatorii genetici. O alegere mai bună din punct de vedere al eficienței o reprezintă limbajele C și C++ cu structuri de date adecvate pentru reprezentarea expresiilor.

Spre deosebire de AG și SE unde inițializarea populației este o prelucrare simplă în cazul programării genetice generarea aleatoare a unei expresii aritmetice este ceva mai dificilă. Considerând că *terminale* reprezintă setul de simboluri terminale, *neterminale* reprezintă setul de simboluri neterminale iar *adâncime* reprezintă adâncimea maximă a arborelui sintactic asociat, *alege* este o funcție care selectează un element arbitrar din mulțimea transmisă ca parametru, secvența de generare a unei expresii poate fi descrisă în pseudocod astfel:

```

generare (terminale, neterminale, adâncime)
  IF (adâncime=0) OR "stopare dezvoltare arbore" THEN expresie := alege(terminale)
  ELSE
    functie:=alege(neterminale)
    IF (functie=operator_unar)
      THEN arg:=generare(terminale, neterminale, adâncime-1)
      expresie=(functie,arg)
    ELSE arg1:=generare(terminale, neterminale, adâncime-1)
      arg2:=generare(terminale, neterminale, adâncime-1)
      expresie=(functie,arg1,arg2)
  RETURN expresie

```

3.2 Funcția de adecvare

Evaluarea unui element presupune "execuția programului" respectiv pentru date de intrare și compararea cu ieșirile pe care ar trebui să le producă. Un "program" este cu atât mai bun cu cât rezultatul pe care îl produce este mai apropiat de cel așteptat.

În cazul regresiei simbolice se *evaluează* expresia pentru argumentele din setul de date aflat la dispoziție și se compară cu rezultatul corect fie stabilindu-se un procent al răspunsurilor corecte fie calculându-se o funcție de eroare (valoarea acesteia va fi invers proporțională cu cea a funcției de adecvare).

3.3 Operatori genetici

Încrucișare. Se realizează interschimbând subarbori ai celor doi părinți astfel încât arborii obținuți să respecte aceleași reguli de sintaxă. Deși structurile arborescente pot fi liniarizate (de exemplu printr-o parcurgere de tip preordine) vectorii rezultați nu au aceeași dimensiune (o expresie poate fi mai simplă iar alta mai complexă) motiv pentru care nu se pot aplica direct tehnicile de încrucișare de la algoritmi genetici. Totuși încrucișarea constă în alegerea unui punct de încrucișare (nod sau arc în arbore). Este indicat ca punctele de încrucișare să nu fie selectate uniform aleator ci să fie favorizate nodurile interne (de exemplu în 90% din cazuri se aleg noduri interne pentru încrucișare). Este posibil ca încrucișarea să se aplice doar cu o anumită probabilitate (de exemplu $p_c = 0.8$).

O ilustrare a încrucișării pentru cazul unor expresii simple este prezentată în fig. 4.

Mutație. Deși este un operator secundar mutația permite modificarea structurilor arborescente în moduri în care încrucișarea nu poate să o facă. În funcție de efectul pe care îl are asupra structurii există trei variante de mutație:

- *Schimbare simplă.* Modifică eticheta unui nod selectat aleator (terminal sau neterminal) prin înlocuirea unui operator sau a unei variabile (vezi fig. 5).
- *Cu dezvoltarea structurii.* Constă în înlocuirea unui nod terminal cu un întreg subarbore construit după aceleași reguli (dar care nu face neapărat parte dintr-un element al populației curente, cum se întâmplă în cazul încrucișării) (vezi fig. 6).
- *Cu reducerea structurii.* Constă în înlocuirea unui întreg subarbore cu un nod terminal (vezi fig. 7).

În general se poate alege un nod al arborelui și întreg subarborul ce "pornește" din acel nod este înlocuit cu un altul, generat aleator. Astfel mutația poate fi văzută ca încrucișare cu un arbore generat aleator.

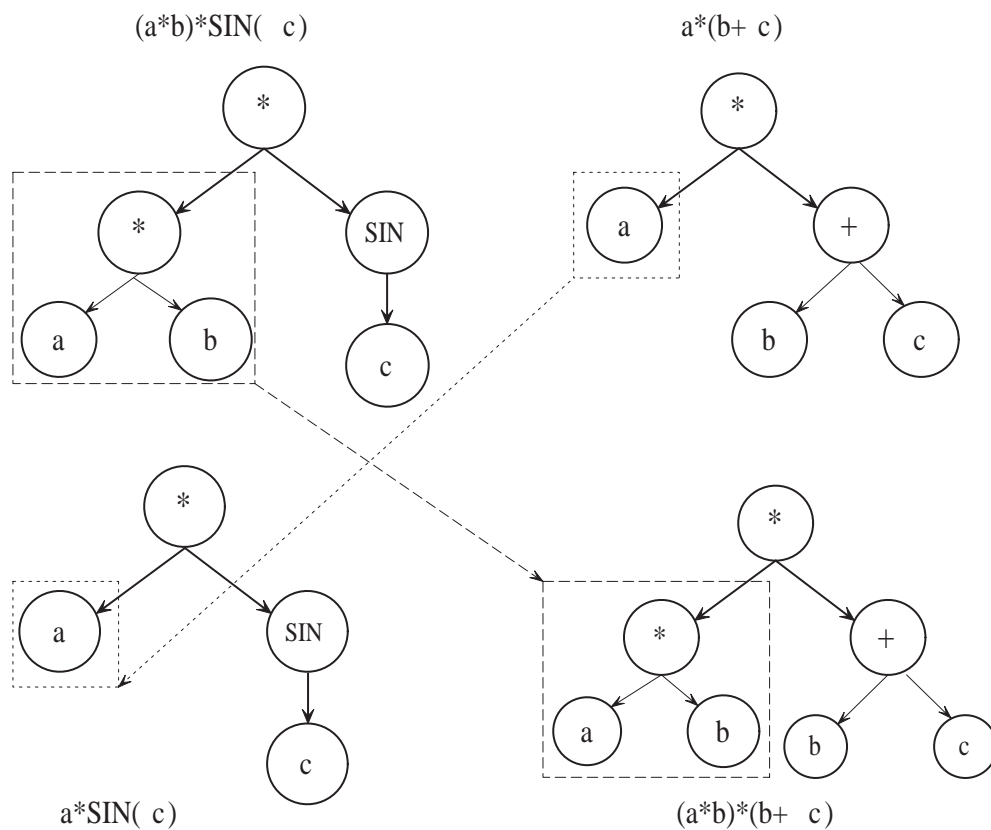


Figura 4: Încrucișarea a doi arbori și expresiile nou generate

3.4 Aplicații

PG se folosește în regresia simbolică, construirea modelelor unor procese economice, proiectarea structurilor de calcul (rețele neuronale, automate celulare, rețele de sortare), prelucrarea imaginilor, paralelizarea automată a algoritmilor.

Referințe

- [1] P. Bentley; Evolutionary Design by Computers, Morgan Kaufmann, 1999.
- [2] T.Bäck, G. Rudolph, H.P. Schwefel; Evolutionary Programming and Evolution Strategies: Similarities and Differences, 1993.
- [3] Z. Michalewicz, R. Hinterding, M. Michalewicz; Evolutionary Algorithms

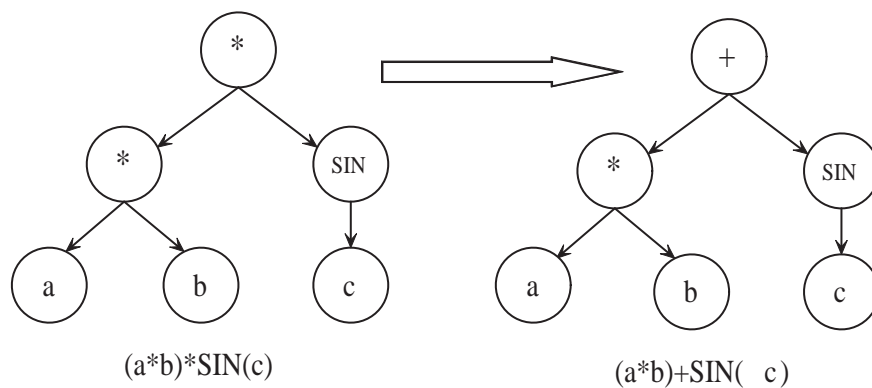


Figura 5: Mutație prin înlocuirea etichetei unui nod ("switch mutation")

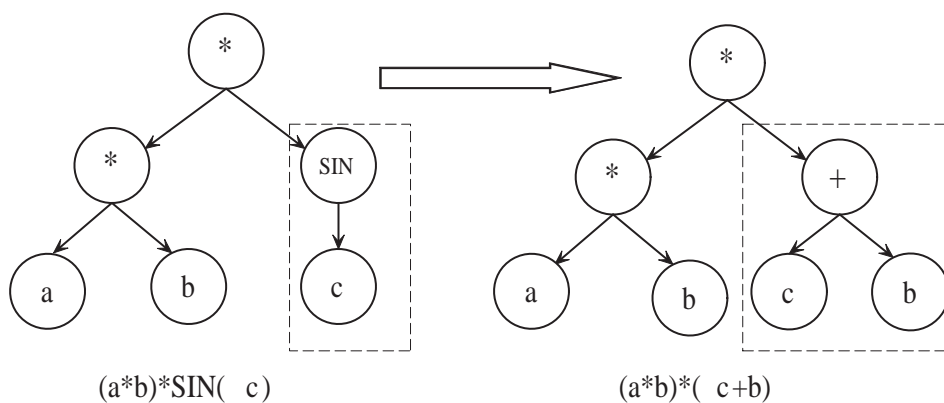


Figura 6: Mutație prin înlocuirea unui nod terminal cu un subarbore ("expanding mutation")

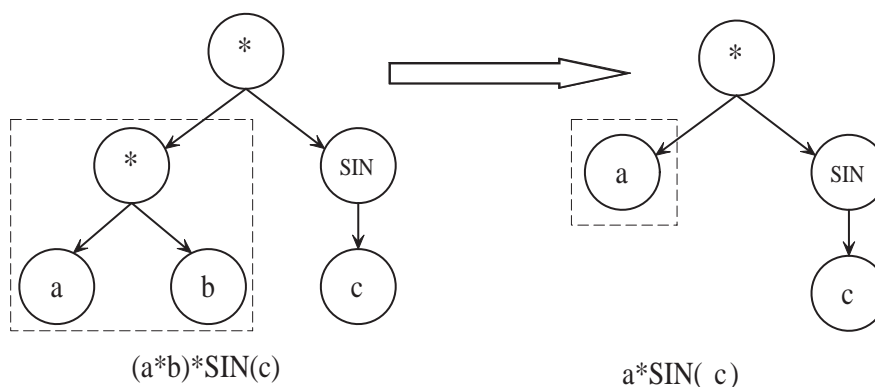


Figura 7: Mutație prin înlocuirea unui subarbore cu un nod ("shrinking mutation")