Laborator 6

Introducere în programarea genetică

1. Obiectivele laboratorului

- Recapitularea conceptelor legate de programare genetică
- Însușirea unei metode de calcul a unei expresii matematice neliniare folosind programarea genetică
- Însușirea unei metode de determinare a structurii unui sistem ETPN folosind programarea genetică
- Exerciții

2. Recapitularea conceptelor de programare genetică

Pentru algoritmii genetici, cromozomul este un vector cu număr fix de gene, cu valori aparținând unui anumit genotip. *Programarea genetică* este o tehnică din domeniul sistemelor evolutive, în care cromozomul are structură și mărime variabile, fiind reprezentat de obicei sub formă de arbore, așa cum se vede în exemplul de mai jos. Astfel, sunt favorizate limbajele care folosesc structuri de arbori în mod natural (de exemplu, Lisp sau alte limbaje funcționale).

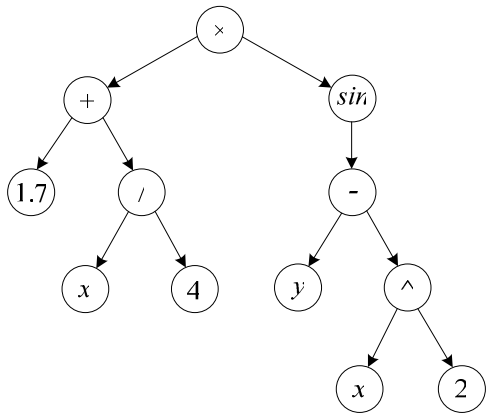

Figura 18. Exemplu de cromozom sub formă de arbore

Nodurile interne (neterminale) conțin operatori sau funcții, deci genotipul aferent este de exemplu: G_{nod} = {+, -, ×, /, ^, √, *sin*, *cos*, *log*, etc.} sau, folosind operatori logici: G_{nod}' = {*if-then-(else), or, not, and*, >, ≤, etc.}. Aici, trebuie ținut cont că numărul de copii să fie potrivit cu natura funcției, de exemplu doi copii pentru +, și unul pentru *sin*.

Nodurile terminale (frunze) conțin valori (operanzi): fie constante, fie variabile de intrare: $G_{terminal} = \{x, y\} + R_+$.

Așadar, un cromozom poate codifica programe software, rezultatul algoritmului fiind un program care rezolvă o anumită sarcină (*task*), conform funcției de performanță.

Operatorii genetici trebuie adaptați la structura cromozomului: *încrucișarea* se face prin tăierea unui subarbore și înlocuirea lui cu un subarbore de la alt cromozom, iar pentru *mutația* unui nod trebuie aleasă o valoare din mulțimea genotip corespunzătoare. Încrucișarea modifică structura cromozomului, deci numărul total de noduri ar putea să crească excesiv, devenind nepractic din punct de vedere computațional.

Se pot introduce totuși diferite constrângeri, de exemplu o adâncime maximă de ordinul $d_{max} = 5$ nivele sau limitarea numărului total de noduri.

Pentru a evalua cromozomul sub formă de arbore, se face mai întâi transformarea arborelui în individul corespunzător, prin *mapping*. Individul este în acest caz, expresia echivalentă sub forma aleasă (expresie matematică, program, expresie Lisp, etc). Construirea expresiei se face cel mai ușor parcurgând arborele în mod recursiv. De exemplu, rezultatul *mapping*-ului pentru cromozomul dat mai sus este expresia matematică $E(x,y) = (1.7 + x / 4) \times sin(y - x^2)$.

Evalurea efectivă a expresiei echivalente (individul) se face într-un mod adecvat, în funcție de specificul aplicației. De exemplu, se pot înlocui operanzii x și y cu valori numerice, se poate evalua structura arborelui sau performanța computațională a expresiei.

Observație

Pentru o viteză computațională superioară, se poate opta pentru implementarea tuturor funcțiilor (inițializare, operatori genetici, evaluare) direct pentru reprezentarea sub formă Lisp a expresiei. Aceasta evită alocarea de memorie (foarte costisitoare temporal) pentru crearea fiecărui nod și, în schimb, efectuează toate procesările pe șiruri de caractere (*string*-uri).

3. Studiu de caz I: determinarea unei expresii matematice / logice

Problema generală este aceea de a determina cea mai bună expresie matematică sau logică, care produce niște rezultatele specificate. Expresia cerută are una sau mai multe variabile de intrare care modifică valoarea/valorile de ieșire. În exemplul următor se dă un tabel de valori (*engl*. *lookup table*) și se dorește determinarea unei expresii logice (program) cu intrarea x care returnează valorile date y.

3.1. Formularea problemei

Se dă tabelul de valori de mai jos:

x	-5	-4	-3	-2	-1	0	1	2	3	4	5
y	0	0	-3	0	0	0	1	2	3	4	5

Se cere să se determine o expresie logică (program) care primește ca intrare o valoare x și generează ca rezultat valoarea y corespunzătoare. Domeniul pentru valoarea de intrare este $x \in [-5, 5]$. Se consideră ca o expresie este cu atât mai performantă cu cât numărul total de noduri este mai mic.

3.2. Rezolvarea problemei

Pentru că problema vizează determinarea unei expresii logice, vom considera mulțimile genotip următoare:

G_{nod} = {*if-then-else, or, and, >, ==*} și $G_{terminal}$ = {x, -5, -4,…, 0, …, 5}

Se observă că din G_{nod} am eliminat funcția ≥, deoarece aceasta poate fi obținută prin operația (> *or* ==), toate cele 3 funcții fiind disponibile. În alte cazuri similare, se poate elimina de exemplu *tg* = *sin* / *cos* (dacă *sin* și *cos* sunt disponibile). Nu există o regulă clară referitor la ce funcții sau valori terminale trebuie să fie incluse în mulțimile genotip, aceasta ține mai mult de problemă și experiența programatorului.

Vom considera adâncimea maximă a arborelui d_{max} = 5, iar funcția de evaluare:

$$J(chr) = \sum_{x=-5}^{5} \left(y(x) - eval\_chr(x) \right) + nr\_nodes$$

, unde *eval_chr*(*x*) reprezintă rezultatul evaluării cromozomului *chr* pentru valoarea lui *x* dată ($x \in [-5, 5]$), și *nr_nodes* reprezintă penalizarea aferentă numărului total de noduri. La nevoie, fiecare dintre aceste două contribuții pot fi scalate cu un factor proporțional.

3.3. Implementare

În pachetul *jgap* există programul *SimpleExample* care implementează o problemă similară. Din acest motiv, nu reproducem aici codul implementat. Este necesară modificarea doar a mulțimilor genotip și a funcției de performanță.

4. Studiu de caz II: determinarea unei expresii TPNL

În cadrul acestui exemplu, vom folosi programarea genetică pentru a determina structura unei rețele ETPN (Enhanced Time Petri Net) pe baza expresiei TPNL (Time Petri Net Language) echivalente.

4.1. Rețele ETPN și limbajul TPNL

Rețelele ETPN sunt rețele Petri temporizate cu porturi de intrare (*reacție*) și ieșire (*control*) care permit interconectarea cu alte rețele similare. Porturile sunt, de fapt, locații care sunt partajate (folosite în comun) cu celelalte rețele conectate.

Notația *TPNL* este o variantă compactă de reprezentare a unei rețele ETPN. În acest limbaj, o tranziție este notată $t_i[r_j, c_k]$, unde:

- r_j reprezintă un *canal de reacție* (condiția de activare a tranziției), adică un arc de la un port de intrare (Pr_j) sau o temporizare τ, care poate fi și τ=0.
- c_k reprezintă un *canal de control* (acțiunea facută de tranziție), adică un arc spre un port de ieșire (Pc_k). Lipsa arcului de control se notează cu φ.

Există mai multe variante de execuție a tranzițiilor sau a secvențelor de tranziții, care pot fi descrise identic în limbajul TPNL:

Operand	Semnificație	Notație TPNL
*	execuție secvențială	exp_1*exp_2
+	execuție alternativă	exp_1 + exp_2
&	execuție concurentă	exp_1 & exp_2
#	execuție repetitivă	exp_1 # exp_2

Expresiile TPNL descriu rețele ETPN echivalente, într-un limbaj compact. Prin concatenarea succesivă a mai multe expresii TPNL, se pot obține rețele ETPN complexe. Se oferă spre exemplu expresia următoare:

$$\sigma_1 = ((t_1[r_1,\varphi] * t_2[4,c_1]) \,\&\, (t_3[r_2,\varphi] * (t_4[3,c_2] + (t_5[r3,\varphi] * t_6[1,c_3]))) \,\#\, t_7[10,\varphi]$$

Conversia din expresie TPNL în reprezentarea sub formă de arbore se rezolvă recursiv și este imediată. Devine evident, deci, că problema de a genera structura ETPN potrivită pentru un anumit scop (de exemplu de control al unui sistem) poate fi abordată prin programare genetică.

4.2. Formularea problemei

În figura de mai jos este prezentat un model ETPN pentru trecerea la nivel cu calea ferată. Rețeaua are porturi de intrare (P_{c1}, P_{c2}) prin care este controlată poziția barierei și porturi de ieșire (P_{r1}, P_{r2}) care sunt activate de apropierea, respectiv de îndepartarea trenului. Trenul parcurge un segment în $\tau = 5$ unități de timp, iar trecerea vehiculelor peste calea ferată este controlată de barieră.

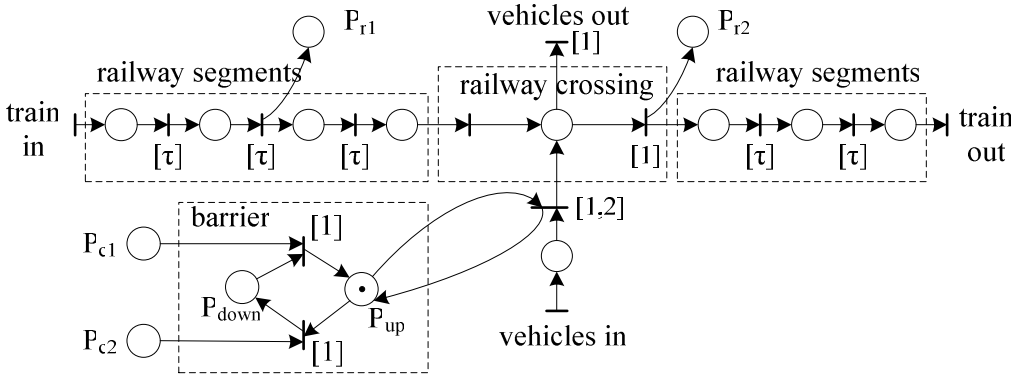

Figura 19. Model ETPN al trecerii la nivel cu calea ferată

Se cere să se construiască un sistem de control ETPN care să controleze trecerea la nivel cu calea ferată în siguranță (nu se permit ciocniri) și în mod cât mai eficient (numărul de vehicule care pot trece într-o perioadă de simulare T_{sim} să fie maxim). Se va considera un flux constant de vehicule (1 vehicul / secundă) și două sau mai multe trenuri succesive la intervale date.

4.3. Rezolvarea problemei

Vom considera mulțimile genotip următoare:

$G_{nod} = \{*, +, \&, \#\}$, $G_{reacție} = \{r_1, r_2, 0, \ldots, 5\}$, $G_{control} = \{c_1, c_2, \varphi\}$

Se observă că în acest caz, $G_{terminal} = G_{reacție} + G_{control}$, cu mențiunea că pentru un nod terminal (tranziție) se vor alege doi parametri: canalul de reacție (din $G_{reacție}$), și canalul de control (din $G_{control}$). La operația de mutație (în cazul unui nod terminal) se va considera posibilitatea mutației numai a reacției sau numai a controlului. În consecință, trebuie rescris operatorul de mutație folosit de algoritm cu unul personalizat.

Funcția de performanță va lua în considerare mai multe contribuții: existența accidentelor (un tren și un vehicul se află simultan în intersecția dată) și numărul total de vehicule care reușesc să treacă. Pentru fiecare dintre aceste contribuții considerăm un factor de scalare potrivit. De asemenea, un sistem de control mai simplu este considerat mai bun decât unul complex, deci se va penaliza și numărul total de noduri.

$J(chr) = c_1 \cdot nr\_accidents - c_2 \cdot nr\_vehicles\_pass + nr\_nodes$

4.4. Implementare

Pentru un arbore de adâncime d_{max} = 5, mărimea spațiului de căutare în această configurație devine:

$$|S| > |G_{terminal}|^{N_{terminal}}(d_{max}) \cdot |G_{nod}|^{N_{nod}}(d_{max})$$

, unde ^ este operația de ridicare la putere, iar $N_{terminal}$ și N_{nod} sunt numărul de noduri terminale respectiv neterminale pentru adâncimea dată a arborelui. În condițiile problemei,

$$|S| > (7 \cdot 3)^{2^5} \cdot 4^{(2^3+2^2+2^1+2^0)} \approx 2.19 \cdot 10^{51}$$

Din cauza spațiului de căutare extrem de vast, algoritmul standard care tratează fiecare nod ca obiect (care trebuie alocat/dealocat la fiecare operație) devine foarte încet și ineficient în găsirea unei soluții utile. În consecință, se optează pentru implementarea tuturor operațiilor (crossover, mutație, evaluare, generare de cromozom inițial) direct pe expresia de tip TPNL a cromozomului, folosind șiruri de caractere. În acest fel, se reduce considerabil nevoia de a instanția obiecte noi. În limbajul Java, se pot folosi obiecte de tip `StringBuilder`, care permit operații directe pe șirul de caractere. Algoritmul implementat este disponibil la adresa: [link git !], împreună cu un exemplu de utilizare.

4.5. Testare

Vom considera adâncimea maximă a arborelui d_{max} = 5, iar pentru evaluarea unui cromozom (sistem de control) vom realiza o simulare cu T_{sim} = 50 secunde. Vehiculele vor intra în sistem cu perioada dată (1 vehicul / secundă), și vom introduce jetoane reprezentând trenuri în partea stângă la momentele T_1 = 5s, T_2 = 20s, T_3 = 25s. Sistemul se va simula cu ajutorul rețelei ETPN descrisă în figura de mai sus.

5. Exerciții

1. Folosind exemplul *SimpleExampleGP* din pachetul *jgap* ca punct de pornire, construiți programul pentru studiul de caz I. Pentru aceasta, setați mulțimile genotip G_{nod} si $G_{terminal}$ și implementați funcția de performanță așa cum sunt acestea descrise în capitolul 3.2. Rulați programul și notați expresia rezultată.
2. Studiați influența parametrilor cromozomului: adâncimea maximă, numărul total de noduri, factori de scalare, etc. asupra performanței de a găsi soluția optimă.
3. Construiți grafic rețeaua ETPN pentru expresia lui σ_1 dată în capitolul 4.1. Pentru expresia TPNL dată ca `string`, construiți operatorul de mutație așa cum este el descris în capitolul 4.3. Se vor folosi probabilități configurabile pentru mutația unui operator, nod terminal – reacție sau nod terminal – control.
4. Construiți programul pentru studiul de caz II. Se va folosi simulatorul pentru sistemul dat disponibil la următoarea adresă: [link git2 !] și pachetul de programare genetică care lucrează pe șiruri de caractere [link git !]. Folosiți operatorul de mutație implementat la exercițiul 3 sau cel deja disponibil. Implementați funcția de performanță și scenariul de simulare. Rulați programul și interpretați fizic sistemul de control generat.

5. Construiți manual, pe hârtie, cel mai bun sistem de control ETPN care respectă specificațiile problemei. Calculați performanța acestuia și opriți algoritmul dacă ajunge la o diferență de performanță de sub 5% față de cel mai bun sistem construit. Cât timp rulează algoritmul acum ? Poate fi îmbunătățit sistemul ?