Laborator 4

Determinarea parametrilor pentru controlul cu logica fuzzy

1. Obiectivele laboratorului

- Recapitularea conceptelor: control fuzzy, Fuzzy Logic Enhanced Time Petri Nets (FLETPN)
- Însușirea unei metode de generare a regulilor pentru un sistem de control cu logica fuzzy, cu ajutorul algoritmilor genetici
- Însușirea unei metode de generare a tabelelor de reguli fuzzy logic pentru un sistem FLETPN cu specificații date
- Studiul efectului pe care îl are modul de evaluare asupra regulilor fuzzy generate.

2. Recapitularea conceptelor legate de logica fuzzy

2.1. Controlul fuzzy

Ființele umane folosesc o logică intuitivă bazată pe valori discrete (*fuzzy*) și reguli fuzzy de forma dată mai jos, unde x_1, x_2, y_1, y_2 sunt valori fuzzy care aparțin mulțimilor finite X_1, X_2, Y_1, Y_2.

IF (x_1 IS X_1 AND x_2 IS X_2 …) THEN (y_1 IS Y_1 AND y_2 IS Y_2 …)

Pentru a putea folosi reguli fuzzy la controlul unui sistem automat vom folosi structura de mai jos, unde sunt reprezentate operațiile necesare: *fuzzyficare* (transformare din valori numerice în valori fuzzy), *aplicarea regulilor* de control *FLR* pe valori fuzzy, *defuzzyficare* (transformare din valori fuzzy în valori numerice).

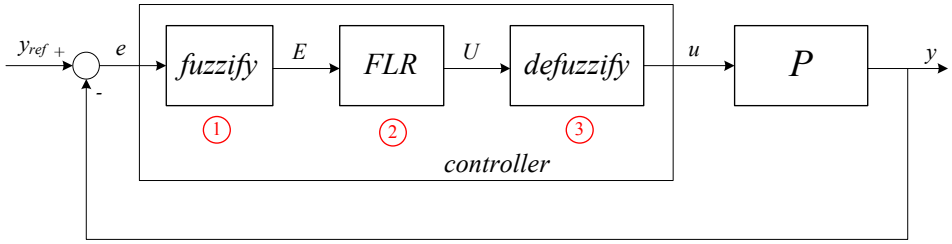

Figura 7. Controlul fuzzy al unui proces

Pentru *fuzzyficare* și *defuzzyficare* vom folosi *funcțiile de apartenență* date mai jos (*NL* – negative large, *NM* – negative medium, *ZR* – zero, *PM* – positive medium, *PL* – positive large), cu valorile numerice corespunzătoare, în ordine: -1, -0.5, 0, 0.5, 1. Valorile fuzzy vor fi reprezentate ca vectori ce conțin 5 grade de apartenență, de exemplu x_1' = [0, 0, μ_{ZR}', μ_{PM}', 0].

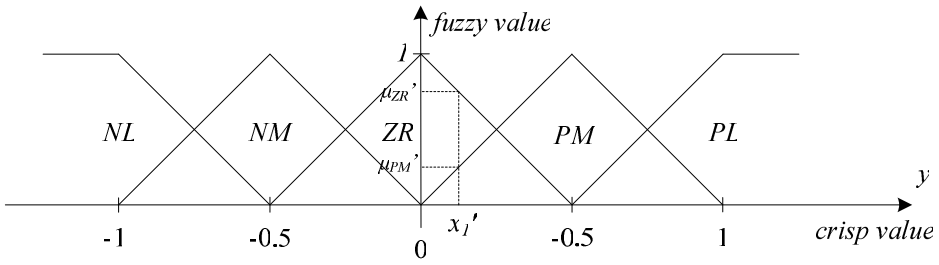

Figura 8. Funcții de apartenență (membership functions)

În figura de mai jos sunt reprezentate regulile fuzzy pentru un sistem cu 2 intrări. Pentru x_1 între *ZR* si *PM*, iar x_2 între *NM* și *ZR*, avem 4 reguli active, indicate în tabel. Pe baza gradului de îndeplinire a fiecărei reguli, se calculează mărimile de ieșire. Pentru un sistem cu o singură intrare, tabelul de reguli devine un simplu vector.

x_2 \ x_1	*NL*	*NM*	*ZR*	*PM*	*PL*
NL	(*PL*, *PL*)	(*PL*, *PM*)	(*PL*, *ZR*)	(*PL*, *NM*)	(*PL*, *NL*)
NM	(*PM*, *PL*)	(*PM*, *PM*)	(*PM*, *ZR*)	(*PL*, *NM*)	(*PM*, *NL*)
ZR	(*ZR*, *PL*)	(*ZR*, *PM*)	(*ZR*, *NL*)	(*ZR*, *NM*)	(*ZR*, *NL*)
PM	(*NM*, *PL*)	(*NM*, *PM*)	(*NM*, *ZR*)	(*NM*, *NM*)	(*NM*, *NL*)
PL	(*NL*, *PL*)	(*NL*, *PM*)	(*NL*, *ZR*)	(*NL*, *NM*)	(*NL*, *NL*)

Figura 9. Reguli fuzzy reprezentate ca tabel (două intrări: x_1, x_2)

2.2. Fuzzy Logic Enhanced Time Petri Nets (FLETPN)

Structurile FLETPN sunt rețele Petri modificate în care jetoanele au în componență o valoare fuzzy cu 5 grade de apartenență, ca mai sus. Tranziția de bază a unei rețele FLETPN implementează un set de reguli fuzzy (FLRS). Tranziția devine executabilă atunci când cel puțin o regulă din tabelul aferent este activă. Factorii de amplificare w_1, $w_2 \in$ [-10, 10] scalează mărimile de intrare.

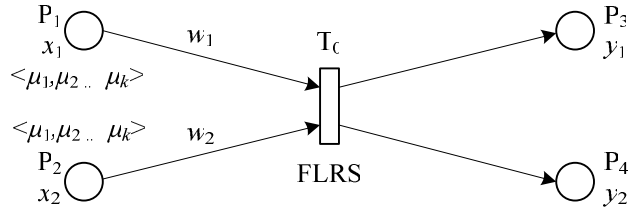

Figura 10. Exemplu de tranziție FLETPN

În plus față de logica fuzzy standard, modelul FLETPN introduce o nouă valoare fuzzy, „φ” (*FF*), care semnifică lipsa jetonului, fie la intrare, fie la ieșire. Cu ajutorul acestei valori, se poate descrie de exemplu un arc inhibitor. Valoarea fuzzy FF este tratată identic ca și celelalte valori în tabelul de reguli.

3. Studiu de caz I: stabilirea regulilor fuzzy logic de control pentru un sistem simplu

Determinarea regulilor de control fuzzy care oferă performanțe ridicate este o problemă în sine. Sunt necesare cunoștințe ample despre sistemul care trebuie controlat și implică de obicei experiența experților care descriu tabelele de reguli. O abordare mai simplă și mai directă o constituie algoritmii genetici, care implică cunoștințe minime despre sistem.

3.1. Formularea problemei

Se dă modelul matematic al unui sistem cu intrări (u_i) și ieșiri (y_i). Se cere stabilirea regulilor de control fuzzy pentru care sistemul respectă o referință dată.

De exemplu, vom folosi modelul unui sistem de gradul II în spațiul stărilor care modelează un cuptor cu gaz. Stările sistemului reprezintă concentrația de oxigen din camera de ardere (x_1), respectiv temperatura (x_2), iar intrările sunt debitul de gaz (u_1), respectiv aportul de oxigen (u_2). Ieșirea reprezintă temperatura în cuptor.

$$\begin{cases} \begin{bmatrix} x_1(k+1) \\ x_2(k+1) \end{bmatrix} = \begin{bmatrix} 0.2 & -0.4 \\ -0.1 & 0.9 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ -0.1 & -0.1 \end{bmatrix} \cdot \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix} \\ y(k) = \begin{bmatrix} 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1(k) \\ x_2(k) \end{bmatrix} \end{cases}$$

Vom calcula regulile de control pentru un sistem de control fuzzy care comandă sistemul pentru a respecta o referință dată. Starea inițială este $x_1(0) = 1$, $x_2(0) = 0$ și se consideră intrările, stările și ieșirea sistemului limitate în intervalul [-1, 1] (în caz contrar, ele pot fi scalate folosind niște factori constanți).

3.2. Rezolvarea problemei

În cele ce urmează vom considera funcțiile de apartenență fuzzy din Figura 10, deci vom folosi 5 nivele fuzzy: *NL*, *NM*, *ZR*, *PM*, *PL*. Vom folosi ca și arhitectură a sistemului cea din figura de mai jos, observând că mărimea de control are două componente, u_1 și u_2. Convenția de notație este că valorile numerice sunt reprezentate cu litere mici, iar cele fuzzy cu majuscule.

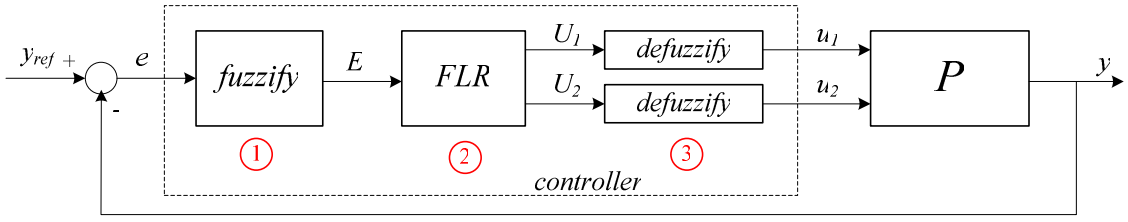

Figura 11. Arhitectura sistemului cu control fuzzy

Tabelul de reguli implementate de blocul *FLR* este de fapt un vector de 5 perechi, pentru că avem o singură intrare, *E*, și două ieșiri, U_1 și U_2. Un exemplu valid de reguli este indicat mai jos. Necunoscutele problemei sunt regulile de control înscrise în tabel, deci acestea vor fi codificate în cromozom. Tabelul de reguli este forma soluției căutate (individul).

E	NL	NM	ZR	PM	PL
(U_1, U_2)	(PL, PL)	(ZR, PM)	(NL, ZR)	(PL, NM)	(PL, PL)

Pentru a stabili structura cromozomului vom partitiona tabelul de perechi în două tabele distincte (pentru U_1, respectiv U_2), rezultând deci un vector de 10 gene (5 pentru U_1 și 5 pentru U_2). Genele conțin numere întregi în intervalul [1, …, 5], care codifică cele 5 nivele fuzzy: *NL*, *NM*, *ZR*, *PM*, *PL*. O alternativă este se implementeze o nouă clasă specială pentru acest tip de gene (`FuzzyGene`).

Funcția de performanță practic simulează sistemul și penalizează eroarea totală a ieșirii *y* față de referință y_{ref}. Cerința problemei este ca sistemul să respecte orice referință dată, deci funcția de performanță ar trebui să testeze toate valorile posibile pentru referință în intervalul [-1, 1].

3.3. Implementare

Oferim mai jos secvența de cod care implementează funcția de performanță.

Secvența de cod 1: Funcția de performanță pentru sistemul fuzzy

```
public class FuzzFitnessFunction extends FitnessFunction {
  private static final double[] yrefValues = { -1, -0.5, 0, 0.5, 1};
  private static final int optimizationHorizon = 20;

  public double evaluate(IChromosome chr) {
    OneXTwoTable FLRS = Mapping(chr);
    double errorSum = 0;
    //should use every possible setpoint
    for (double yref : yrefValues){
      double[] y = GetY(FLRS, yref);
      for (int i=0; i < optimizationHorizon; i++)
        errorSum += Math.abs(y[i]-yref);
    }
    return errorSum;
  }
}
```

În secvența de cod dată se folosește pachetul JGAP, precum și utilitarul FuzzyP, disponibil la adresa: https://github.com/AttilaOrs/FuzzP/tree/master/fatJar, care implementează logica fuzzy. Fișierul *.jar* trebuie adăugat ca referință în proiect. Diagrama claselor pentru cele mai importante componente ale acestei biblioteci este prezentată mai jos. Clasa indicată cu roșu (`FuzzFitnessFunction`) este cea dată în secvența de cod de mai sus și trebuie implementată astfel: metoda `Mapping(chr)` preia din cromozom regulile de control și construiește un obiect `OneXTwoTable`, iar metoda `GetY(FLRS, yref)` calculează ieșirea *y* prin simularea sistemului (figura 11) pe întreg orizontul de optimizare.

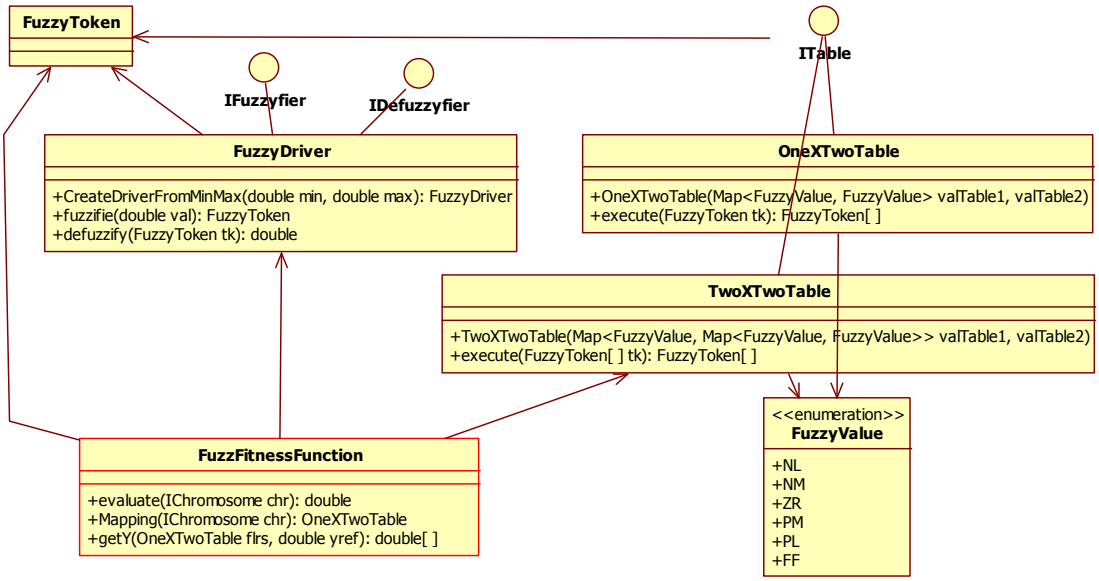

Figura 12. Diagrama claselor pentru pachetul FuzzyP

3.4. Testare și concluzii

Precizia soluției

Algoritmul genetic poate stabili valoarea corectă numai pentru acele reguli care au fost activate la un moment dat în cadrul evaluării performanței. Regulile care nu au fost activate niciodată nu au nici un efect asupra performanței, deci valoarea lor va fi aleatoare. Din acest motiv s-a recurs la simularea repetată a sistemului pentru mai multe valori ale referinței. Pentru simplitate, în funcția de performanță au fost testate, ca și referință, doar valorile care corespund lui *NL*, *NM*, *ZR*, *PM* și *PL*. Aceste valori acoperă intervalul specificat [-1, 1].

Convergența și eficiența algoritmului

În acest caz, spațiul de căutare are mărimea $|S| = 5^{10}$ (5 posibilități pentru fiecare dintre cele 10 gene). Mărimea populației trebuie să fie suficient de mare pentru a oferi varietate bazinului genetic, iar numărul de generații trebuie să fie suficient de mare încât fiecare genă (statistic vorbind) să poată fi atinsă de mutație sau încrucișare. Totuși, trebuie ținut cont că pentru fiecare evaluare de cromozom se fac practic 5 simulări ale sistemului, pe orizontul de optimizare. Recomandăm stabilirea mărimii populației și a numărului de generații astfel încât să se evalueze în total aproximativ 2-3% din spațiul de căutare.

Considerații despre timpul de execuție

Folosind această abordare, nici nu este necesară cunoașterea modelului matematic al sistemului. De exemplu, în cazul unui sistem software de tip *blackbox*, se poate recurge în loc de simulare la activarea efectivă a sistemului (prin apelarea metodelor unei aplicații externe). În mod similar se poate proceda în cazul unor sisteme fizice rapide. În ambele cazuri, întârzierile de comunicație și ale sistemului pot fi relevante și trebuie luate în considerare.

4. Studiu de caz II: stabilirea regulilor pentru un sistem FLETPN

În cazul unor sisteme mai complexe, de exemplu atunci când sunt necesare mai multe tabele de reguli, aceste tabele pot fi liniarizate și concatenate pentru a forma cromozomul. Prezentăm mai jos un exemplu de componentă FLETPN pentru care trebuie stabilite regulile de inferență fuzzy din tranziții.

4.1. Formularea problemei

Se dă componenta FLETPN din figura de mai jos, cu 3 porturi de intrare (P_1, P_2, P_3) și 2 porturi de ieșire (T_2, T_3). Valorile jetoanelor corespunzătoare porturilor sunt x_1, x_2, δ, pentru intrări și y_1, y_2 pentru porturile de ieșire. Toate jetoanele sunt de tip fuzzy cu mulțimea valorilor S = {*NL*, *NM*, *ZR*, *PM*, *PL*, *FF*}. Componenta reprezintă un comparator cu histerezis, și funcționează astfel:

- dacă $x_1 - x_2 > \delta$, atunci $y_1 = 1$ și $y_2 = \varphi$ (deci doar T_2 este activată)
- dacă $x_2 - x_1 > \delta$, atunci $y_1 = \varphi$ și $y_2 = -1$ (deci doar T_3 este activată)
- dacă $|x_1 - x_2| \leq \delta$, atunci $y_1 = 1$ și $y_2 = -1$ (ambele tranziții de ieșire sunt activate)
- dacă lipsește una din mărimile x_1, x_2 (au valoare *FF*), atunci $y_1 = \varphi$ și $y_2 = \varphi$ (nici o tranziție de ieșire nu este activată), iar dacă lipsește δ, acesta va fi considerat $\delta = 0$

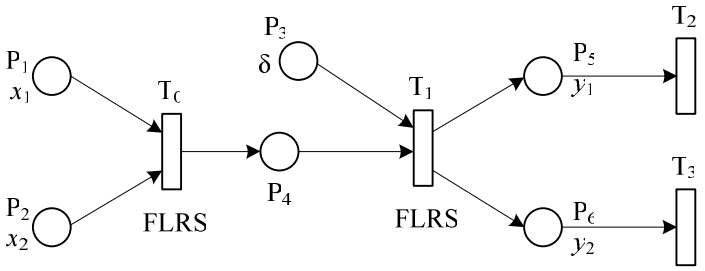

Figura 13. Modelul componentei ca rețea de tip FLETPN

Se cere să se găsească tabelele de reguli FLRS pentru tranzițiile interne ale componentei (T_0 și T_1) astfel încât să se respecte specificațiile comportamentale date.

4.2. Rezolvarea problemei

Pentru mulțimea valorilor fuzzy S dată, tabelul de reguli FLRS al tranziției T_0 conține $6 \times 6 = 36$ de reguli (pentru o singură ieșire), iar pentru tranziția T_1 care are două ieșiri, $2 \times 36 = 72$ de reguli, deci în total 108 reguli. În consecință vom considera un cromozom cu 108 gene de tip număr întreg (`IntegerGene`, cu intervalul [1, …, 5], similar cu exemplul anterior).

Funcția de performanță penalizează, ca de obicei, diferența dintre comportamentul real al sistemului și cel referință. Ca și în exemplul anterior, ar fi necesar să testăm în funcția de performanță toate combinațiile posibile ale valorilor de intrare x_1, x_2, δ, pentru a activa fiecare regulă cel puțin o dată. În cazul nostru, vom parcurge doar cazurile aferente valorilor fuzzy și vom penaliza diferența dintre valorile y_1, y_2 și referințele lor din formularea problemei.

4.3. Implementare

Oferim mai jos funcția de performanță recomandată.

Secvența de cod 2: Funcția de performanță pentru componenta FLETPN

```
public class FLETPNFitnessFunction extends FitnessFunction {
  private static final double[] fuzValues = { -1, -0.5, 0, 0.5, 1};

  public double evaluate(IChromosome chr) {
    TwoXOneTable FLRS1 = Mapping1(chr);
    TwoXTwoTable FLRS2 = Mapping2(chr);

    double errorSum = 0;
    //should use every possible combination of inputs
    for (double x1 : yrefValues)
      for (double x2 : yrefValues)
        for (double d : yrefValues){
          FuzzyToken[] yref = GetYRef(x1, x2, d);
          FuzzyToken[] y = GetY(FLRS1, FLRS2, x1, x2, d);
          errorSum += GetDiff(yref[0],y[0]) + GetDiff(yref[1],y[1]);
        }
    return errorSum;
  }
}
```

Metoda `GetYRef` returnează valorile fuzzy ale lui y_1 și y_2 ce vor fi folosite ca referință, conform specificațiilor problemei. Metoda `GetY` calculează valorile fuzzy ale lui y_1 și y_2 aplicând intrările `x1`, `x2` și `d` fuzzyficate, pe tabelele `FLRS1` și apoi `FLRS2`.

Metoda `GetDiff` compară valorile jetoanelor date ca argumente și returnează diferența absolută dintre ele (le defuzzyfică, apoi calculează diferența în modul).

5. Exerciții

1. Implementați un program Java pentru problema prezentată în capitolul 3. Stabiliți mărimea populației și numărul de generații conform indicațiilor. Testați programul și observați precizia soluției.
2. Eliminați unele dintre valorile referinței (`yrefValues`). Care reguli nu vor fi activate acum ? Rulați programul de mai multe ori și verificați faptul că regulile neactivate niciodată sunt generate aleator.
3. Implementați programul pentru studiul de caz II, care stabilește regulile de inferență fuzzy aferente tranzițiilor pentru componenta dată. Testați programul și evaluați performanța stabilirii regulilor.
4. Notați tabloul optim de reguli găsit la exercițiul 1. Până acum, am folosit pentru simplitate aceleași funcții de apartenență pentru toate mărimile, dar aceasta nu este întotdeauna varianta reală. Implementați acum un program Java care va găsi cele mai bune funcții de apartenență (în locul celor din Figura 8). În esență, trebuie găsite valori numerice pentru nivelele fuzzy *NL*, *NM*, *ZR*, *PM*, *PL* pentru fiecare dintre mărimile de intrare și de ieșire: $e = y_{ref} - y$, u_1, u_2 (deci în total 15 valori).
   Atenție ! Valorile pentru nivelele fuzzy trebuie să fie în ordine, deci nu este permis de exemplu ZR > PM. Pentru construirea funcțiilor de apartenență se va folosi clasa `TriangleFuzzifier`.