Státnice 2000, otázka 4/3

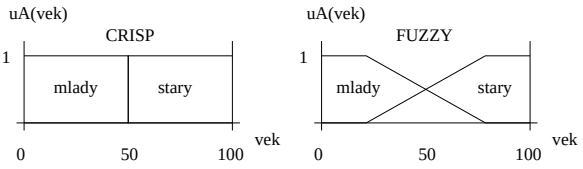
Tomáš Hlavatý, xhlavaty@fel.cvut.cz

Leden 2000

Softcomputing: Základy fuzzy množin, fuzzy řízení a rozhodování. Hrubé (rough) množiny. Neurofuzzy učící se algoritmy.

Evoluční a genetické algoritmy.

Bohužel jsem nesehnal žádné materiály o hrubých (rough) množinách. Pokud někdo víte, co to je, dejte mi vědět.



Obrázek 1: Hodnoty ve fuzzy logice

1 Fuzzy řízení a rozhodování

Klasické metody řízení a rozhodování vyžadují přesné vyjádření vztahů mezi vstupy a výstupy systému. **Fuzzy logika** (Zadeh, 1965) místo přesných matematických formulací vyjadřuje vztahy mezi vstupy a výstupy pomocí množiny **pravidel** tvaru „jestliže — pak“, založených na zkušenostech a znalostech. Fuzzy logika pracuje s vícehodnotovou, gradovanou pravdou a s vágními lingvistickými výrazy.

Fuzzy množiny poskytují intuitivně přijatelný způsob reprezentace neurčitosti typu vágnosti. **Fuzzy množina** je množina uspořádaných dvojic elementů x a jejich stupně (míry) příslušnosti (MFV — Membership Function Value) $A = \{(x, \mu_A(x)); x \in X\}$. Je charakterizována funkcí příslušnosti (MF — Membership Function) $\mu_A \in \langle 0, 1 \rangle$, která udává, jak silně jsme přesvědčeni, že uvažovaný prvek patří do dané množiny [1].

V praktických úlohách se užívá funkcí příslušnosti ve tvaru Gaussovy křivky, může být lichoběžníková, trojúhelníková atd.

1.1 Hodnoty

Hodnota **CRISP** je ostrá (ano/ne, přesné číslo), hodnota **FUZZY** je neurčitá s nějakou mírou jistoty (obr. 1). Proměnná x se nazývá **lingvistická proměnná**.

1.2 Operace

Fuzzy logika zavádí operace konjunkce, disjunkce, komplement a fuzzy implikaci. Mohou být definovány různě, např. podle Zadeha:

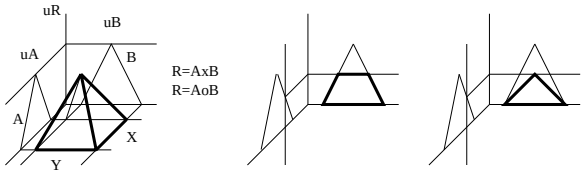
$$\begin{aligned} \mu_{A \wedge B} &= \min\{\mu_A, \mu_B\} \\ \mu_{A \vee B} &= \max\{\mu_A, \mu_B\} \\ \mu_{\neg A} &= 1 - \mu_A \end{aligned}$$

nebo pravděpodobnostně:

$$\begin{aligned} \mu_{A \wedge B} &= \mu_A \mu_B \\ \mu_{A \vee B} &= \mu_A + \mu_B - \mu_A \mu_B \\ \mu_{\neg A} &= 1 - \mu_A. \end{aligned}$$

Fuzzy implikace svazuje dvě fuzzy proměnné (obr. 2) a může být opět definována různě, např.

$$\mu_R(x, y) = \min\{\mu_A(x), \mu_B(y)\}.$$

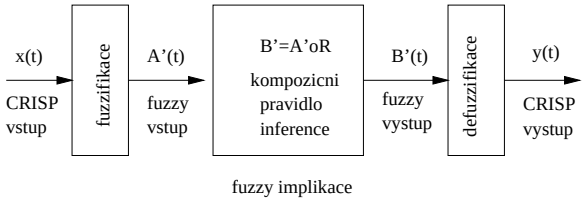


Obrázek 2: Fuzzy implikace

1.3 Fuzzifikace a defuzzifikace

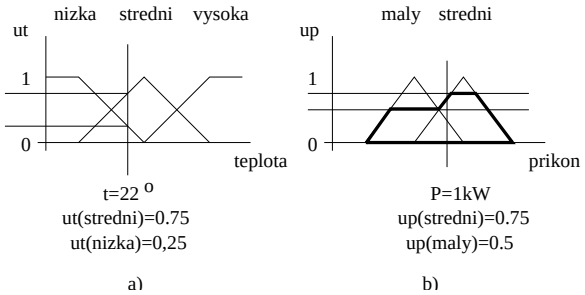
Kompletní schema fuzzy systému je na obr. 3.

Při **fuzzifikaci** (obr. 4a) se převádí vstupní CRISP hodnoty na FUZZY hodnoty (lingvistickou proměnnou a míru příslušnosti).



Obrázek 3: Kompozice fuzzy systému

Při **defuzifikaci** (obr. 4b) se převádí FUZZY hodnoty na výstupní CRISP hodnoty. Používají se různé metody, např. výpočet těžiště celé plochy nebo střední úsek s max. hodnotou.



Obrázek 4: Fuzzifikace a) a defuzzifikace b)

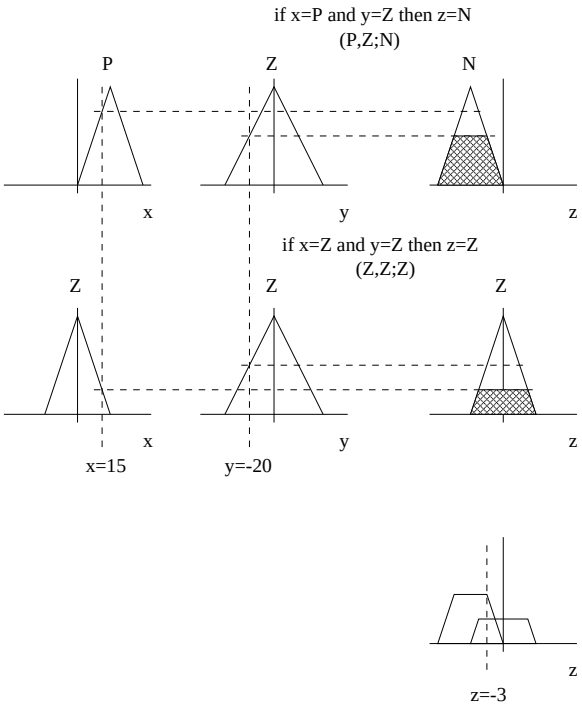
1.4 Pravidla

Pravidla pro fuzzy řízení mají tvar „jestliže — pak“ (fuzzy implikace). Ve fuzzy logice jde o zobecněný modus ponens. Předpoklad popisuje vstupní stav, závěr akci. Např. „jestliže teplota je nízká, pak příkon je vysoký“. V podmínkové části může být více podmínek, např. „if x is A and y is B then z is C“.

Na obr. 5 je znázorněn průběh inferenční procedury [3] pro dvě vstupní proměnné x a y a jednu výstupní proměnnou z. Nejprve jsou CRISP hodnoty $x = 15$ a $y = -20$ převedeny na FUZZY hodnoty (fuzzifikace). Poté se vyhodnotí výraz „and“ $(\min)$ a provede fuzzy implikace (řez). Výsledky obou pravidel se poté složí a metodou těžiště se spočte výstupní CRISP hodnota $z = -3$ (defuzzifikace).

1.5 Fuzzy řízení

V regulační technice se nejčastěji používá Mandamiho fuzzy implikace. Implementuje se pomocí tabulky (look-up table), která se předem vytvoří pro každé pravidlo a při řízení se výstup určuje již pouze vyhledáváním a interpolací. Výhodou je možnost popsání silně nelineárních vztahů. Tento přístup je velmi



Obrázek 5: Příklad inferenční procedury

robustní. Problém nastává při větším počtu vstupních proměnných, neboť roste i dimenze tabulky.

1.6 Fuzzy Logic Neural Network

Místo vyhledávací tabulky je v systému pětivrstvá neuronová síť [4]. 1. vrstva provádí fuzzifikaci, 2. podmínku if, 3. část and — then, 4. část also (or) a 5. defuzifikaci. Tato síť je schopna popsat libovolný fuzzy systém.

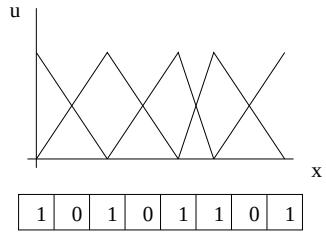
$\oplus$ Je do sítě explicitně vidět.

$\oplus$ Síť se dá trénovat (neuro-fuzzy učící se systém)!

V [2] jsou popsané i třívrstvé sítě. Učit se dají různými způsoby, ať se znalostí nebo bez ní (síť sama objeví shluky). Nejjednodušší je učení pomocí delta-pravidla. Klasické metody učení neuronových sítí však neumožňují během učení měnit počet fuzzy množin a tvar a počet pravidel. Při učení se nastavují jen parametry fuzzy množin. S výhodou se dá k učení použít genetických algoritmů. Chromozóm pak popisuje fuzzy množiny podle obr. 6.

2 Evoluční a genetické algoritmy

Evoluční algoritmy (EA) jsou algoritmy **prohledávací**, které pracují nad populací možných řešení.



Obrázek 6: Učení neuronových fuzzy sítí genetickými algoritmy

Populace těchto řešení se s časem vyvíjí podle pravidel evoluce. Používají se hlavně při řešení NP-úplných úloh, které nemůžeme tradičními algoritmy řešit v rozumném (polynomiálním) čase. Při řešení těchto úloh se použije nějaká nedeterministická metoda (např. EA) k nalezení možného řešení a toto řešení se v polynomiálním čase otestuje.

Genetické algoritmy (GA) jsou částí algoritmů evolučních. Mohou být použity v prohledávacích a optimalizačních úlohách. Jsou založeny na genetických procesech biologických organismů. Jsou nedeterministické a v principu paralelní. Pracují s populací řetězců (obdoba chromozómů z biologie), které reprezentují množinu parametrů řešené úlohy. V populaci dochází k reprodukci, křížení a mutaci řetězců a vznikají tak populace nové a kvalitnější podle zadaného kritéria.

Prostor všech možných řešení se nazývá **prohledávací prostor**. Každý bod tohoto prostoru je ohodnocen **fitness funkcí**. Hledaní řešení znamená hledání extrému fitness funkce v prohledávacím prostoru. Tuto úlohu řeší mnohé algoritmy, např. náhodné prohledávání, gradientní metody (Hill Climbing), iterativní gradientní metody (Iterated Hill Climbing) nebo simulované žíhání. Oproti klasickým metodám se GA liší v několika bodech:

Přímá manipulace se zakódovanými parametry — GA nepracuje přímo s parametry, ale s jejich reprezentací konečné délky a abecedy. Prohledávání je bez omezení a není nutná existence derivace fitness funkce.

Populace řešení — Prohledávání začíná z několika míst zároveň. Genetické operátory zajišťují diverzitu v populaci a brání uvíznutí v lokálním extrému.

Stochastické operátory — GA používá nedeterministické operátory. Nejedná se o náhodnou procházku, ale díky náhodné volbě se dosahuje dobrých prohledávacích schopností.

GA jsou podepřeny teorií o **schématech**. Na schématech se dokazuje činnost genetických operátorů. Obsahují symboly 0, 1 a *. Symbol * (don't care symbol) reprezentuje buď 0 nebo 1. Např. operátor křížení lze schémata popsat jako [11**0*] + [01*11*] → [11**1*] a [01*10*] při křížení za čvrtým znakem.

2.1 SGA

SGA je jednoduchý GA s následujícím algoritmem:

1. Náhodně se vygeneruje **počáteční populace** n chromozómů (možných řešení problému).
2. Chromozómy se ohodnotí **fitness funkcí** $f(x)$.
3. Vytvoří se **nová populace** chromozómů:
   (a) **Selekce** — Vyberou se dva chromozómy (rodiče) ze současné populace.
   (b) **Křížení** — Na vybrané chromozómy se aplikuje operátor křížení s ohledem na jeho pravděpodobnost. Pokud nebyl operátor křížení aplikován, jsou nově vzniklé chromozómy (potomci) přesné kopie rodičů.
   (c) **Mutace** — Na potomky se aplikuje operátor mutace s ohledem na jeho pravděpodobnost.
   (d) **Umístění** — Potomci se umístí do nově vytvářené populace.
4. Nově vzniklá populace **nahradí** populaci původní.
5. Pokud není splněna ukončující podmínka, pokračuje se bodem 2, jinak běh algoritmu **končí**.

2.2 Kódování chromozómů

Chromozóm musí obsahovat informaci o řešení úlohy, které reprezentuje. Výběr kódování chromozómů je značně závislý na konkrétní úloze.

Binární řetězec — Nejpoužívanější reprezentace, kde každý bit (nebo množina bitů) může reprezentovat nějakou charakteristiku řešení (např. [11011001]).

Pokud chceme např. do batohu určité kapacity vložit co nejvíce předmětů různé velikosti a ohodnocení (Knapsack problem), může každý bit chromozómu indikovat přítomnost odpovídajícího předmětu v batohu.

Permutation Encoding — Užívá se v úlohách, kde je důležité pořadí. Každý chromozóm je řetězec čísel, které reprezentují pořadí v sekvenci (např. [1 5 2 4 3]). Pro zachování konzistence chromozómu jsou nutné speciální operátory křížení a mutace.

Pokud např. chceme projít všemi danými městy a urazit co nejmenší vzdálenost (TSP — Travelling Salesman Problem, nejlevnější Hamiltonův cyklus v úplném grafu), popisuje chromozóm pořadí měst, které budem navštěvovat.

Value Encoding — Když je kódování do binárních řetězců komplikované, lze vytvořit chromozóm přímo z potřebných hodnot (např. [1.23 2.34 3.45] nebo [(back), (back), (right), (forward)]). Pro daný problém je nutné vytvořit speciální operátory křížení a mutace.

Pokud chceme např. nastavit váhy neuronové sítě určité architektury podle daných vstupů a výstupů, použijeme chromozóm reálných čísel reprezentujících danou váhu.

Tree Encoding — Používá se hlavně k hledání programů nebo výrazů pro **genetické programování**. Každý chromozóm je strom objektů, jako jsou funkce operátory, čísla, příkazy programovacího jazyka apod. (obr. 7).

Pokud např. chceme nalézt funkci, která nejlépe vyhoví zadaným vstupním a výstupním hodnotám, bude chromozóm reprezentovat určitou funkci.

2.3 Fitness funkce

Fitness funkce $f(x)$ přiřazuje chromozómu číslo (ohodnocení), které odpovídá jeho kvalitě. Navrhuje se pro každou úlohu. Pokud např. hledáme optimum funkce, bude fitness funkce udávat přímo její hodnotu. V mnoha případech je ale složitější. Neměla by pokud možno obsahovat příliš mnoho lokálních extrémů, velmi izolované globální optimum a rozumná řešení by se v prohledávaném prostoru měla nacházet poblíž sebe.

2.4 Selekce

Výběr rodičů provádí operátor selekce. Podle Darwina vzniknou kvalitní potomci z nejlepších jedinců. Existuje mnoho metod výběru rodičů:

Ruletové kolo — Rodiče se vybírají podle jejich ohodnocení fitness funkcí. Nejlepší jedinci mají větší šanci být vybráni. Chromozómy jsou seřazeny podle fitness funkce do kruhu a každý zabírá poměrově část podle své fitness funkce. Náhodně se ukáže na místo na kruhu a vybere se odpovídající jedinec.

Rank Selection — Ruletové kolo nepracuje dobře při velmi odlišných fitness funkcích chromozómů. Proto zde dochází k rovnoměrnějšímu rozdělení chromozómů do kruhu. Nejhorší jedinec bude zabírat poměrnou část 1, další 2 a nejlepší N. Po takovém přepočtu mají všechny chromozómy šanci být vybrány. Může to ovšem vést k pomalejší konvergenci, protože se nejlepší jedinci od ostatních moc neliší.

Přepočet fitness funkce pro selekci se nazývá škálování. Existuje několik **škálovacích metod**, např. lineární, mocninová nebo Linear Ranking. Cílem je udržet konstantní průměrnou hodnotu fitness funkce v populaci a dát šanci i špatným jedincům k výběru. Mohou totiž obsahovat část kvalitního chromozómu, třeba i z oblasti, která ještě nebyla v populaci moc zkoumána.

Steady-State Selection — V každé generaci se vybere několik chromozómů s vysokou hodnotou fitness funkce pro vytváření potomků. Chromozómy s nízkou hodnotou fitness funkce jsou nahrazeny těmito potomky. Zbytek populace se zkopíruje do populace nové.

Elitizmus — Při vytváření nové populace křížením a mutací je velká pravděpodobnost, že ztratíme nejlepší řešení. Elitizmus nejprve zkopíruje do nové populace jeden či více nejlepších chromozómů a zbytek populace se vytvoří jako obvykle. Elitizmus může značně zefektivnit GA, protože nedochází k zapomínání nejlepšího nalezeného řešení.

2.5 Křížení

Při křížení se z rodičů vytváří potomci. Způsob křížení závisí značně na kódování chromozómu. Pro zefektivnění GA jsou pro speciální úlohy specifické operátory křížení.

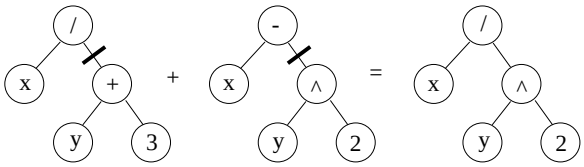
1-point — Náhodně se vybere křížící bod a takto vzniklé části rodičů se zkombinují (např. [11011001] + [01010110] → [11010110] a [01011001] při křížení za čtvrtým bitem). Při Permutation Encoding je např. [1 2 3 4 5] + [4 2 3 5 1] → [1 2 4 3 5] při křížení za druhou cifrou.

2-point — Křížící body jsou dva. Potomci vznikají kombinací částí rodičů jako u křížení 1-point.

Uniformní křížení — Bity se náhodně kopírují střídavě z jednoho či druhého rodiče (např. [110011] + [001100] → [010110]).

Aritmetické křížení — Na rodiče se aplikuje aritmetická operace (např. and).

Řezy — Při Tree Encoding se provádí řezy podstromů a potomci vzniknou prohozením těchto podstromů (obr. 7).



Obrázek 7: Křížení v genetickém programování (Tree Encoding)

2.6 Mutace

Mutace je prevencí před uvíznutím algoritmu v lokálním optimu. Neměla by nastávat příliš často, protože by se GA změnil v náhodné prohledávání. Mutace náhodně mění potomky.

Mutace závisí na kódování chromozómů. Při binárním kódování chromozómů náhodně invertuje bity. Při Permutation Encoding prohazuje dvě vybrané cifry (např. [1 2 3 4 5] → [1 5 3 4 2]. Při Value Encoding s reálnými čísly na dané pozici přičítá nebo odčítá malé náhodné číslo. Při Tree Encoding mutace mění vybrané uzly za jiné objekty.

2.7 Parametry

Doporučené nastavení parametrů je jen přibližné a neexistuje žádná teorie univerzální pro jakýkoliv řešený problém. Doporučení jsou založena na empirických studiích GA.

Pravděpodobnost křížení — říká, jak často bude operátor křížení aplikován. Měl by být větší (80%–90%, někdy i 60%).

Pravděpodobnost mutace — říká, jak často bude operátor mutace aplikován (nebo jak často budou části chromozómu mutovány). Měla by být malá (0.1%–1%). Je možné jí i dynamicky měnit pro překlenutí lokálního extrému.

Velikost populace — udává počet chromozómů v populaci. Jestliže je příliš malá, má GA malou šanci prohledat dostatečnou část prohledávaného prostoru. Pokud je příliš velká, je GA pomalý. Bývá okolo 20–30 nebo i 50–100. Závisí na kódování chromozómů a jejich velikosti.

2.8 Ošetření omezení

Omezení představují pro GA velký problém. Chromozóm reprezentuje velký prohledávací prostor, který je přitom z části neplatný. Omezení lze ošetřit několika způsoby:

Penalty — Zakázané chromozómy jsou označeny za horší ohodnocením fitness funkcí. Je použitelné jen pro velmi malá, jednoduchá omezení. Problémem je volba pokutové funkce a síly omezení.

Dekodéry a opravovací algoritmy — Speciální procedury, které dekódují chromozóm vždy jako platné řešení nebo opraví zakázanou reprezentaci na nejbližší platnou. Obecně může být velmi drahé.

Speciální reprezentace a operátory — V závislosti na úloze je nutné navrhnout reprezentaci a odpovídající operátory. Problémem je nalezení takové reprezentace (např. *ad hoc*).

Příkladem je řešení TSP. Klasický GA by prohledával prostor N^N, ale je $N!$ řešení je správných. Proto je navržena speciální reprezentace (Permutation Encoding) a operátory.

2.9 Paralelní GA

GA jsou z principu paralelní. Jejich implementace může mít různý stupeň paralelizmu.

Globální — Jedinci se volně kříží v jediné populaci, paralelně se počítá pouze jejich fitness funkce, mohou se paralelně aplikovat i genetické operátory.

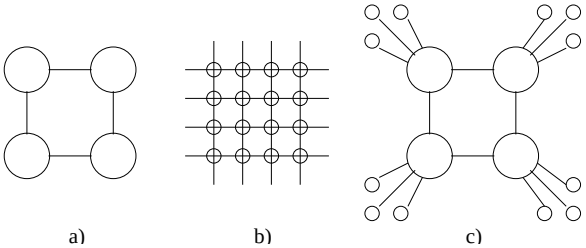
Hrubozrnný — Výpočet je rozdělen do subpopulací (obr. 8a). Většina výpočtů probíhá v rámci těchto subpopulací, čas od času se však vymění jedinci mezi jednotlivými subpopulacemi.

Jemnozrnný — Populace je tvořena jedinci, kteří běží každý ve svém procesu (obr. 8b). Vznikne síť těchto jedinců, kteří mezi sebou komunikují a množí se. Použitelné pro masívně paralelní počítače, jinak drahé. Odlišná koncepce od klasického GA.

Hybridní — kombinace globálního, hrubozrnného a jemnozrného GA (obr. 8c). V praxi dosahuje nejvyššího výkonu, je ale pracnější na implementaci.

Problémem u paralelních GA je konvergence. Když nám pro jednu úlohu konverguje klasický GA, nemusí ještě konvergovat paralelní GA.

Dalším problémem je **synchronizace** na úrovni populací. Buď může výpočet probíhat asynchronně, nebo se provádí synchronizace, kdy se čeká na dokončení všech výpočtů, nebo se počká jen např. na 70% nových jedinců a pokračuje se v další iteraci.



Obrázek 8: Paralelní GA: a) hrubozrnný, b) jemnozrnný a c) hybridní

Literatura

[1] Mařík, Štěpánková, Lažanský a kolektiv: Umělá inteligence 2. Academia, 1993.

[2] C. H. Chen: Fuzzy Logic and Neural Network Handbook. ISBN 0-07-011189-9, McGraw-Hill, 1996.

[3] Bart Kosko: Neural Networks and Fuzzy Systems. ISBN 0-13-612334-1, Prentice-Hall, 1992.

[4] Přednášky a cvičení z předmětu Softcomputing, 1999.