1 Agentní systémy

Agent je entita, která podle svých vlastností (potřeby, cíle, preference) působí na prostředí kolem sebe a reaguje na podněty.

Autonomie je hlavní vlastností agenta (jedná nezávisle a sám kontroluje svůj stav a akce).

Triviální agenti mohou být i chemikálie, přístroje nebo softwarové procesy (daemon).

Inteligentní agent je systém, který vykonává akce flexibilně, je reaktivní, proaktivní a sociální.

Reaktivní agent bezprostředně reaguje na změny v prostředí a podněty vhodnou akcí.

Proaktivní (deliberativní) systém má (nebo si stanoví) cíle a provádí akce k jejich dosažení (nemusí být závislý na prostředí).

Čistě reaktivní agent reaguje pouze na okolí a současnost, nebere v potaz minulost (zkušenosti). Jednoduchý, snadno realizovatelný, méně chybový.

Reaktivní agent se stavem udržuje informaci o stavu (pamatuje si historii, učí se apod.)

Reaktivní a proaktivní požadavky jsou v konfliktu, agent musí najít rovnováhu.

Hybridní systém hledá optimální použití reaktivnosti a proaktivnosti. Reaktivní část ma většinou vyšší prioritu, vrstvená architektura.

Sociálnost agenta je přihlédnutí ke skutečnosti, že prostředí obsahuje agentů více, někdy je potřeba kooperace. Agent potřebuje schopnost komunikace a spolupráce pomocí *agentního komunikačního jazyka*.

Další vlastnosti agentů:

- *Racionalita* – výběr nejvhodnější akce pro dosažení cíle
- *Adaptivita/učení* – zlepšování vlastností v čase
- *Mobilita* – schopnost přemisťovat se
- *Důvěryhodnost* – neposkytnutí nepravdivé informace
- *Benevolence* – pokud nekonfliktuje s cíli, vykonává požadavky

Typy agentních prostředí:

- *Dostupné vs. nedostupné* – Lze získat úplnou informaci o stavu prostředí? (reálný svět → ne)
- *Deterministické vs. nedeterministické* – Lze určit výsledek akce na 100%? (reálný svět → ne)
- *Epizodické vs. neepizodické* – Lze dělit na nezávislé úseky? (lze restartovat?)
- *Statické vs. dynamické* – Je ovlivněno pouze agentem nebo i nezávisle? (multiagentní je dynamické)
- *Diskrétní vs. spojité* – Existuje konečný počet vjemů/akcí za jednotku času?

Intenční systém (BDI architecture – beliefs, desires, intentions) něčemu věří, po něčem touží a má nějaké záměry. Druhý stupeň intenčního systému pak i věří a touží po víře a tužbách atd.

2 Modelování systémů

Inteligentní systémy jsou složité a dynamické. Pro důkladné testování je potřeba vývoje pomocí *simulace*.

Postupy modelování a simulace jsou formálně definovány, existují nástroje.

Postup: Máme reálný systém, vytvoří se model, jehož výstupy odpovídají výstupům reálného systému. Simulace provádí instrukce modelu a generuje chování podobné reálnému systému.

Systém s diskrétními událostmi je reprezentován časovou posloupností událostí (vstup dat, konec výpočtu, apod.). Nejčastěji je specifikován pomocí DEVS.

DEVS – Discrete Event System Specification – matematický formalismus pro specifikaci modelů. Existují simulátory, distribuovaná simulace, mapování na jiné modely (statecharts, FSM, Petriho sítě), varianty pro spojitou simulaci a softcomputing (FuzzyDEVS), dynamický DEVS, ...

Model systému: Má výstupy (generovány samovolně nebo v reakci na vstup) a vstupy (na některé nereaguje). Uvnitř dochází ke změnám stavu (samovolně nebo v reakci). Může být sestaven ze subsystémů.

Formalismus DEVS: $M = (X, Y, S, s_0, \tau, \delta_{ext}, \delta_{int}, \lambda)$ – vstupy, výstupy, stavy a počáteční stav, časová funkce, přechodová funkce pro podněty, přechodová funkce stavů, výstupní funkce

Interpretace formalismu: Systém je v nějakém stavu $s \in S$. Pokud nedostane vstup, zůstává ve stavu s po dobu $\tau(s)$ a pak generuje výstup $\lambda(s)$ a přejde do $\delta_{int}(s)$. Při vstupu v čase e (od minulé události) ihned přejde do $\delta_{ext}(s, e, x)$. Při konfliktu interního a externího přechodu má přednost externí. Na vstupní a výstupní proměnné lze mapovat porty.

Hierarchie modelování: atomické a spojované modely (atomic and coupled). Atomický model odpovídá základní definici DEVS.

Spojovaný model definuje propojení submodelů, obsahuje tedy množinu modelů (komponent), množinu vlastních vstupních a výstupních portů (vnější obal) a propojení portů (ať už komponent nebo vázání na své porty).

Spojovaný DEVS: $CM = (X, Y, D, \{M_{i \in D}\}, EIC, EOC, IC, select)$ – vstupy, výstupy, množina jmen komponent, modely komponent, vazby vnějších vstupních portů, vazby vnějších výstupních portů, vnitřní vazby, funkce výběru při konfliktu $2^D \rightarrow D$ (dvě simultánní události v modelech)

Pro každý spojovaný DEVS lze vytvořit ekvivalentní atomický DEVS. (věta)

3 Strojové učení

Učením se agent přizpůsobuje prostředí a zlepšuje své parametry v čase.

Učící element systému komunikuje s prováděcím (stará se o senzorická data a efektory a poskytuje informace o výkonu apod.) a také dostává zpětnou vazbu, podle které se může řídit.

Typy zpětné vazby:

Supervised learning – učitel poskytuje správné odpovědi/řešení

Unsupervised learning – vazba je založena na změně parametrů samotného systému

Reinforcement learning – někdy je agent odměněn (např. až při dosažení cíle, ale to není po každé změně)

4 Zpětnovazební učení, Markovský systém s odměnami, dynamické programování

Zpětnovazební učení (reinforcement learning) je učení na základě interakce s prostředím. Prostředí je nedostupné a nedeterministické a vazba může být zpožděna, není neustálá (odměna/trest nepřijde vždy). Např. učení backgammonu odmění/potrestá až na konci hry; inverzní kyvadlo – trest při pádu.

Postup: agent zvolí akci a provede ji, pak sleduje stav a vyhodnocuje případnou odměnu. Na základě informací sestavuje strategii, která přinese největší odměnu, učí se tak z vlastních pokusů a omylů.

Markovský rozhodovací proces s odměnami je sestaven z množiny stavů a akcí. Následující odměna závisí *pouze* na aktuálním stavu a zvolené akci. Agent pak podle stavu vybere akci a je okamžitě odměněn, podle čehož se dále řídí.

Učení v Markovských procesech: agent se snaží najít optimální strategii s maximální odměnou vycházejíc z libovolného stavu. Data jsou tvaru (stav, akce, odměna). Nelze použít učitele, protože optimální akce pro stav není známa, je známa pouze odměna, ale v lokálním kontextu.

Ohodnocení stavu – při dané strategii se stav hodnotí podle budoucích odměn, které přijdou při použití daného stavu.

Ohodnocení akce – při dané strategii se akce ohodnotí podle odměny, která při použití akce v daném stavu přijde.

Dynamické programování řeší problém po částech a sjednocením výsledků získává nejlepší postup.

Metoda Monte Carlo používá náhodné vzorkování ke zisku statistiky – v agentním prostředí se náhodně nastaví parametry a pustí se epizoda. Po analýze výsledků se vyberou nejlepší parametry (nejlépe ohodnocené stavy stavového prostoru).

Q-learning je učení, které nepotřebuje model prostředí, rozhoduje se podle současného stavu a celkového plánu.

TD-learning (Temporal Difference) oproti Monte-Carlo nehodnotí po konci epizody všechny projité stavy, ale za běhu pouze stav minulý. To ale znamená, že stejnou cestou je potřeba jít několikrát, aby se odměna propagovala až do počátečního stavu (MC to udělá ihned). Proto se někdy používá TD s n zpětnými kroky ohodnocení.

5 Neuronové sítě

Neuronová síť je propojení několika neuronů pomocí synapsí. Je snaha vyrobit takové sítě (ANN) v programovém prostředí a dosažení podobných vlastností jako má mozek.

Vlastnosti ANN: Vysoce paralelní a distribuované, váhovaná propojení jednotek, jednotky s prahovým vyhodnocováním, samostatná změna vah, ...

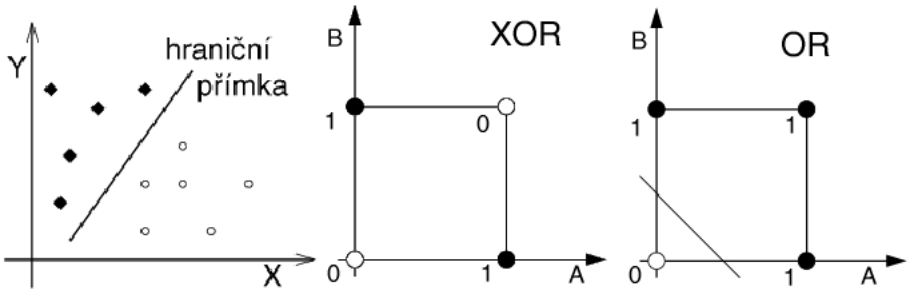
Neuron má několik vstupů nad nimiž je definována *bázová funkce* (uvažuje i váhy vstupů) a také obsahuje *aktivační funkci* – skoková nebo spojitá (její vstup je výsledek bázové funkce).

Architektury:

- *Plně propojená* – každý má vazbu s každým (v obou směrech)
- *Plně propojená symetrická síť* – váhy opačných vazeb jsou stejné
- *Vsrtvová* – síť obsahuje několik vrstev (vstupní, skryté, výstupní)
- *Acyklická síť* – vstvová bez cyklů vazeb
- *Dopředná síť* – vstvová, kde vstupy ovlivňují i další vrstvy

Perceptron je nejjednodušší síť (je to vlastně jeden neuron), která dle vstupů vytváří výstupní hodnotu a provádí lineární separaci prostoru hodnot (dá se říci, že na základě vstupů odpoví ANO/NE). Perceptrony lze použít pouze na lineárně separabilní prostory hodnot.

Lineární separabilita je vlastnost problémů – zda se prostor výstupních hodnot dá rozdělit přímkou na dvě části. Například funkce OR je takto separabilní, ale XOR není.



Učení perceptronu spočívá v úpravách vah vstupních hodnot tak dlouho, až směr přímky je vhodný ke správnému rozdělení prostoru.

Backpropagation je systém zpětného šíření chyby mezi vrstvami neuronových sítí. Pokud je některá výstupní hodnota označena za špatnou, nejen poslední neuron si upraví váhy, ale také zpětně všechny neurony k němu napojené v nižších vrstvách.

6 Fuzzy logika

Fuzzy množiny umožňují částečnou příslušnost prvku z intervalu $\langle 0, 1\rangle$ (oproti $\{0, 1\}$ u klasických množin). Píšeme, že prvek x patří do fuzzy množiny X s 80% pravděpodobností, nebo že jeho příslušnost k množině X je 0.8.

Funkce příslušnosti mapuje jednotlivé příslušnosti k prvkům množiny. Může být dána diskrétně – $m_X = \{(a, 1), (b, 0.5), (c, 0)\}$ nebo spojitě $m_X = max(0, q - x)$ (q je nějaká konstanta).

Vlastnosti fuzzy množin:

$A = B \Leftrightarrow \forall x : \ m_A(x) = m_B(x)$
$A \subseteq B \Leftrightarrow \forall x : \ m_A(x) \leq m_B(x)$
$m_{A\cup B}(x) = max(m_A(x), m_B(x))$
$m_{A\cap B}(x) = min(m_A(x), m_B(x))$
$m_{A'}(x) = 1 - m_A(x)$
$support(A) = \{x \mid m_A(x) > 0\}$
Grafická znázornění mluví sama za sebe.

Singleton je fuzzy množina, jejíž support je tvořen jediným prvkem.

Odlišnosti (zákony, které nejsou stejné jako u ostrých množin): Neplatí zákon protikladu ani vyloučení třetího – $A \cap A' \neq \emptyset$ a $A \cup A' \neq U$ kde U je univerzum (všechny možné prvky).

Fuzzy operace:

- *Koncentrace* – $CON(A) = \{(x, m_A^2(x)) \mid m_A^2(x) = m_A(x) \cdot m_A(x)\}$ (zůžení rozložení)
- *Dilatace* – $DIL(A) = \{(x, \sqrt{m_A(x)}\}$ (rozšíření rozložení)
- *Normalizace* – $NORM(A) = \{(x, m_A(x)/MAX)\}$ (srovnání maxima na hodnotu 1)
- *Kartézský součin* – $A \times B = \{(x, m_{A\times B}(x) \mid x = (a, b) \wedge m_{A\times B}(a, b) = min(m_A(a), m_B(b))\}$
- *Relace* – $R = \{(x, m_R(x)) \mid x = (a, b) \wedge m_R(a, b) \leq m_A(a) \wedge m_R(a, b) \leq m_B(b)\}$ (jelikož je to podmnožina kart. součinu, takže platí $A \subseteq B \Leftrightarrow \forall x : \ m_A(x) \leq m_B(x)$)

7 Fuzzy řízení

Postup: fuzzifikace, fuzzy inference, defuzzifikace. Ostré hodnoty převedeny na fuzzy, na tyto jsou aplikována pravidla a výsledné hodnoty se po sloučení převedou na ostrou hodnotu.

Pravidla jsou ve formě znalostí experta `if x is big then a is small` apod. Je častý zápis do tabulek pro konjunkční výrazy.

Používané fuzzy hodnoty: large negative, small negative, zero, small positive, large positive. Nekdy jsou large hodnoty vynechány. Převod mezi fuzzy a ostrými hodnotami se provádí podle míry příslušnosti.

Fuzzy inference pracuje se všemi pravidly najednou. Každé pravidlo má svou váhu (jak moc se má ve výsledném sloučení uplatnit). Pravidlo se také uplatní do té míry, do jaké se uplatnily jeho předpoklady (vstupy). Ručně se tento postup nejlépe dělá graficky.

Defuzzifikace se provádí nalezením buďto těžiště singletonů nebo těžiště plochy.

8 Genetické algoritmy

Genetické algoritmy jsou založeny na evoluční teorii z reálného světa.

Vytvářejí se *generace* hypotéz a z nich se vyberou ty nejlepší k dalšímu křížení.

Výběr se provádí na základně klasifikace a ohodnocení (fitness function), ta je různá podle aplikace.

Mutace je uměle zavedena aby nedocházelo k degeneraci výsledků (např. křížením příliš podobných jedinců) – inverze náhodného bitu.

Crowding (nakupení) – pokud se vyberou ke křížení jedinci blízko u sebe, pak se populace koncentrují a je třeba je znovu diverzifikovat.

Křížení vybere z rodičů jednotlivé části a jejich spojením vznikají noví jedinci:

- *One-point crossover* – v jednom bodě se rozdělí a prohodí si stejně dlouhé části
- *Two-point crossover* – stejné, ale ve dvou bodech

- *Cut and splice* – části nejsou stejně dlouhé

Genetické programování je programování za využití postupů genetických algoritmů. Program je reprezentován stromem a křížení se provádí prohozením podstromů (mutace pak mění informaci – konstanty nebo generuje kód). Každý takový program se spustí a ohodnotí na základě testovacích dat a vypočtených výsledků.

9 Plánování

Plán je posloupnost akcí vedoucích k dosažení cíle. Oproti řešení je dán předem a nemusí být pak přesně sledován.

Jazyk při plánování je predikátová logika, aktuální stav je dán množinou formulí (např. `is_in(car, garage)`).

Plánovač volí v daném stavu akci tak, aby nejvíce vyhovovala cíli, takto postupně sestaví plán – od počátku (progrese) nebo od konce (regrese).

Lineární plánovač produkuje posloupnost akcí s linárním uspořádáním (otevři dveře, projdi, zavři dveře).

Nelineární (parciální) plánovač produkuje posloupnost akcí s částečným uspořádáním (nazuj si levou botu nebo pravou botu, následně jdi). Je potřeba linearizace, tedy množina možných lineárních plánů.

STRIPS je regresní lineární plánovač. Jeho jazyk obsahuje akce, předpoklady (predikáty) a efekty. Instance akce je pak akce s dosazenými hodnotami proměnných. Instance uzavřená (grounded) neobsahuje proměnné, kdežto otevřená (lifted) ano.

Příklad STRIPS:

```
scheme:
   PRECOND: car_at(car,X)
   ACTION:  drive(car,X,Y)
   EFFECT:  car_at(car,Y) & !car_at(car,X)
grounded instance (X/home, Y/work):
   PRECOND: car_at(car,home)
   ACTION:  drive(car,home,work)
   EFFECT:  car_at(car,work) & !car_at(car,home)
```

Regresní hledání plánu bere pravidla opačně a vybere se taková akce, která aktuální stav zajišťuje, stav se změní na podmínky dané akce. Výhodou je, že se řešení nalezne vždy, pokud existuje.

Progresní hledání plánu vybírá takové akce, které je možné spustit. Také nalezne řešení, pokud existuje, ale více se větví a jsou aplikovány i akce které nevedou k cíli.

Heuristika je ulehčení plánování pomocí odhadu výsledku.

Sussmanova anomálie: při lineárním programování může efekt jedné akce zrušit splnění podcíle akce jiné.

Nelineární plánovače prohledávají prostor plánů, používaný nástoj je GRAPHPLAN. Mnohem složitější systémy, řeší konflikty podcílů, vzájemné vyloučení apod.

Nedeterministické prostředí klade další překážky, plánovač musí sestavit plán pro dosažení cíle z každého stavu.

10 Distribuovaná umělá inteligence

DAI se soustředí na paralelní řešení problémů pomocí AI. Nejčastěji používáno ve smyslu miltiagentních systémů nebo multiagentních prostředí.

Multiagentní prostředí obsahuje více agentů a ti spolu musejí spolupracovat nebo vycházet, případně spolu soupeří.

Multiagentní systém využívá multiagentní prostředí k řešení problému (tedy ostatní agenti už nejsou jen prostředí, ale součástí systému).

Vlastnosti MAS:

- Agenti jsou alespoň částěčně autonomní.
- Agenti nemají globální znalosti, pouze lokální.
- Decentralizovaná architektura, neexistuje řídicí agent.

Komunikace agentů:

- *Přímá* – agenti spolu komunikují přímo pomocí konkrétního protokolu
- *Nepřímá* – buď skrz agenta (prostředníka), nebo skrz prostředí (nástěnky, semafory)

Typy konfliktů:

- *Fyzické* – o data, služby, kanály, kritické sekce, ...
- *Mentální* – agenti se vzájemně blokují v dosažení svých záměrů a cílů

Řešení fyzických konfliktů:

- *Dražba* – dražba o médium
- *Vyjednávání* – pro dva agenty
  - Sekvence návrhů, které končí buď přijetím, nebo odmítnutím, každý agent navrhuje svoje
  - Je možné se dohodnout na jednom návrhu, na kompromisu, odmítnout
  - Agent má funkci, která ohodnotí jednotlivá řešení
- *Volba* – pro tři a více agentů
  - Volby se učástní všichni agenti, pouze někteří jsou kandidáti
  - Každý agent volí na základě svých preferencí
  - Vítěz voleb rozhoduje konflikt

Výsledek dohody je závazný, pokud se ho některý agent nedrží, je penalizován, tzn. izolace agenta, odmítání komunikace, ztráta důvěryhodnosti, nepřidělování prostředků.

Řešení mentálních konfliktů:

- Komunikace – prosazování svých záměrů, agenti argumentují, proč je jejich návrh lepší
- Argumentace – dokazování rozporu dvou teorií (na základě odvozování)

11 Robotické systémy

Android je robot podobný člověku.

Zooid je robot podobný zvířeti.

Mobot (mobilní robot) je využíván v nedostupném nebo nebezpečném prostředí nebo k ulehčení (mobilní vysavač).

Základní schopnosti:

- *Ovlivňovat okolí* – motorický subsytém (efektory), reproduktor, světlo, ...
- *Vnímat okolí* – senzorický subsystém (kamera, mikrofon, jiná čidla), mj. vybírá důležitá data
- *Myslet* – kognitivní subsystém (plánování a řešení úloh, vnímání senzorických dat, vyhodnocení cílů, učení, ...)
- *Zpětné vazby* mezi subsystémy (kontrola vykonání, stavu, apod.)

Robotický systém je skupina robotů, která pracuje na zadaném úkolu a je řízena operátorem (nebo řídicím systémem).

Inteligentní robotický systém obsahuje vlastní řídicí systém, nezávislý na vnějším řízení.

Míra nezávislosti RS:

- *Úplná* – nezávislé RS (vesmírný průzkum planet)
- *Částečná* – řízení koriguje, kontroluje (autopilot apod.)
- *Žádná* – řídicí systém je plně externí (výrobní linky, lékařský robot, mixér apod.)

Efektory jsou buďto k pohybu – *lokomoce* nebo k přesunu předmětů – *manipulace*.

Stabilita nohových robotů je *statická*, pokud se může kdykoli zastavit a neztratí rovnováhu, *dynamická*, pokud je stabilní pouze při pohybu, pak může být stabilní i na jedné noze.

Stupeň volnosti je počet možných pohybů (v prostoru jich je 6 - pohyb podél os a otočení kolem os; v ploše 3 - x,y,otočení).

Neholonomní robot je robot, který má méně stupňů volnosti než možný počet. V ploše se většinou přesto používá (3 kola).