Universitatea
Politehnica
Bucuresti
Anul universitar
2021-2022

Inteligență Artificială

Adina Magda Florea

Continut curs

- **Introducere în IA**
- **Rezolvarea problemelor prin căutare**
  - Căutări informate
  - Problema satisfacerii restricțiilor
  - Căutare adversarială și jocuri
- **Raționament incert**
  - Cunoștințe incerte și raționament probabilistic
  - Rețele Bayesiene
- **Introducere în învățare automată**
  - Arbori de decizie
  - Regresie liniară
  - Regresile logistică
  - Naive Bayes
- **Agenți și sisteme multi-agent**
- **Prelucrarea limbajului natural**
- **Reprezentarea cunoștințelor**
- **Planificare automată**

Materiale curs

- A. Florea. Slide-uri curs
- S. Russell, P. Norvig. Artificial Intelligence: A Modern Approach, Prentice Hall, 2021 (Global Edition), http://aima.cs.berkeley.edu/global-index.html
- D. Poole, A. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*, Cambridge University Press, 2010 – complete book online

  http://artint.info/html/ArtInt.html

Resurse AI on-line

- http://aima.cs.berkeley.edu
- Laboratorul de Inteligenta Artificiala si Sisteme Multi-agent (AI-MAS)
  http://aimas.cs.pub.ro/links
- Asociația Română de Inteligență Artificială
  http://www.aria-romania.org/
- Tools for learning AI
  http://www.aispace.org/index.shtml
- Direcțiile de cercetare din IA:
  http://aitopics.org/

Cursuri AI on-line

MIT Open Courseware

http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/

Udacity

https://www.udacity.com/course/intro-to-artificial-intelligence--cs271

Diferite subuiecte de IA – Coursera

https://www.coursera.org/search?query=artificial+intelligence&categories=cs-ai

Cerințe pentru promovare

- Participare la laborator: minimum 7 ședințe de laborator
- Minim 50% din punctajul de parcurs (laborator+teme de cas) și minim 50% din punctajul de la examenul final
- Activitate și teme de laborator
- Rezolvarea temelor de casă
- Parcurgerea materialelor obligatorii
- Examen final

Notare

- Examen final: 40%
- Laborator 25%
- Teme de casă 25%
- Teste curs 10%

Curs nr. 1

Introducere în IA

Intrebări cheie

Este posibilă simularea comportamentului inteligent pe calculator?

Care este criteriul pe baza căruia se apreciaza inteligența unui program?

La ce nivel se încearca modelarea comportamentului inteligent?

Care sunt reprezentările și tehnicile utilizate în rezolvarea problemelor de inteligența artificială?

1. Ce este inteligența artificială?

- Alan Turing - “Computing Machinery and Intelligence”, 1950
- Loebner prize, 1990 -…

- IA abordare simbolică
- IA abordare non-simbolică

Definiții IA

- Inteligența artificială este studiul facultăților mentale pe baza modelelor computaționale.
- Un program inteligent este un program care manifestă o comportare similară cu aceea a omului când este confruntat cu o problemă similară. *Nu este necesar ca programul sa rezolve sau să încerce sa rezolve problema in acelasi mod in care ar rezolva-o oamenii.*
- Abilitatea de a executa sarcini și de a rezolva probleme care sunt executate și/sau rezolvate de inteligența naturala, în particular de inteligența umană

Definiții IA

- Definiția Inteligenței Artificiale a Grupului de experți la nivel înalt pe probleme de IA al UE, 2019
- *„Sistemele de inteligență artificială (IA) sunt sisteme software (și, eventual, hardware) proiectate de oameni care au un obiectiv complex, acționează în dimensiunea fizică sau digitală, percepând mediul prin intermediul preluării datelor, prin interpretarea datelor structurate sau nestructurate colectate, prin raționament cu privire la cunoștințe sau prin prelucrarea informațiilor obținute din aceste date și prin decizii asupra celor mai bune acțiuni care trebuie întreprinse pentru a realiza obiectivul dat. Sistemele IA pot să utilizeze reguli simbolice sau să învețe un model numeric și își pot adapta comportamentul analizând modul în care mediul este afectat de acțiunile lor anterioare.”*

2. Caracteristicile problemelor IA

- **Generale**
- **Dinamica modelului**
- **Dificile de rezolvat (complexitatea calculului)**
- **Cunoștințe versus date**
- **Utilizarea cunoștințelor euristice**
- **Utilizarea cunoștințelor incerte**
- **Necesită raționament, inferențe**
- **Comportament autonom**
- **Adaptare/învățare**

Inferențe

- Inferență
- Regulă de inferență
  - Consistentă (sound) vs. inconsistentă
  - Completă vs. incompletă
- Strategie de inferență (control al inferentelor)
  - Consistentă vs. inconsistentă
  - Completă vs. incompletă

Exemple de reguli de inferență

$$\frac{A \qquad A \rightarrow B}{B}$$

Inferențe deductive

$$\frac{\text{frumos(mircea)} \qquad \text{frumos(dan)}}{\forall x\ \text{frumos}(x)}$$

Inferențe nedeductive

3. Structura unui sistem IA

- Sistemele de IA pot folosi reguli simbolice sau pot învăța un model numeric și, de asemenea, își pot adapta comportamentul analizând modul în care mediul este afectat de acțiunile lor anterioare
- **Intrarea** sistemului poate fi reprezentată de videocamere, microfoane, text introdus de la tastatură sau senzori care detectează cantități fizice precum temperatura, viteza etc.
- Datele colectate pot fi structurate, adică date organizate conform unui model predefinit (de exemplu baze de date relaționale) sau date nestructurate, date care nu au o structură organizată, de exemplu, o imagine, o zonă de text sau un semnal sonor.

- Partea de **prelucrare** realizează analiza datelor și luarea deciziilor (raționament).
- Se presupune că un sistem de IA acționează, respectiv decide rațional, în sensul că ia decizia corectă pentru atingerea scopului sau rezolvarea problemei.
- Termenul de decizie nu implică neapărat totala autonomie a sistemului. Majoritatea sistemelor actuale de IA sunt semi-autonome, oferind o recomandare operatorului uman, care va fi factorul de decizie final.
- **Ieșirea** este răspunsul sistemului pe baza deciziilor luate
- Elementele de execuție (actuatoare) nu trebuie neapărat să fie fizice, pot fi de asemenea componente software, rezultate prezentate utilizatorului uman etc.

- Capacitățile sistemelor de IA pot fi grupate în două grupuri principale:
  - **capacitatea de raționament a sistemului** - este realizată prin modele care includ reprezentarea cunoștințelor și utilizarea acestora în luarea deciziilor, planificare, căutare și optimizare.
  - **capacitatea de a învăța** – este realizată prin modele cum ar fi rețele neurale, învățare profundă (*deep learning),* sisteme de suport vectoriale (SVM), metode de grupare (*clustering*), algoritmi genetici și altele.

    Aceste tehnici permit unui sistem de IA să învețe cum să rezolve probleme care nu pot fi specificate cu exactitate sau pentru care metoda de rezolvare nu poate fi descrisă prin reguli de raționament simbolic.

- Un aspect important la ora actuală este acela al cuplării abordărilor de IA cu robotica.
- Robotica poate fi definită ca „***IA în acțiune în lumea fizică***” (numită și IA încorporată).
- Un robot este o mașină fizică care trebuie să facă față dinamicii, incertitudinilor și complexității lumii fizice.
- Percepția, raționamentul, acțiunea, învățarea, precum și capacitățile de interacțiune cu alte sisteme sunt, de obicei, integrate în arhitectura de control a sistemului robotizat.

4. Scurt istoric

- Conferința de la Dartmouth College din 1956 - primii patru mari inițiatori ai domeniului: John McCarthy, Marvin Minsky, Alen Newell si Herbert Simon.
- 1956 - 1957 A. Newell, J. Shaw si H. Simon - primul program de demonstrare automată a teoremelor, "The Logic Theorist."
- 1959 Samuel Checkers program
- Începand din 1960 apar primele programe de inteligență artificială.

Istoric

- 1965 J. A. Robinson – rezoluția
- 1965 – DENDRAL - J. Lederberg si E. Feigenbaum. - sistem expert capabil să sintetizeze structura moleculelor organice pe baza formulelor chimice și a spectogramelor de masă
- 1959 - Limbajul Lisp (LISt Processing) - John McCarthy (Dartmouth College)
- 1972 - Limbajul Prolog (PROgrammation et LOGique) - Alain Colmerauer (universitatea Marseille-Aix)
- 1983 - Smalltalk - Goldberg, Robson

Istoric

- Anii ’70 – importanța cunoștințelor
- **Sisteme bazate pe cunoștințe**
- **Ingineria cunoștințelor**
- Sistemul MYCIN - Buchanan, Shortliffe - sistem expert pentru diagnosticarea infecțiilor bacteriene ale sângelui, Stanford University - '74-'75
- Sisteme expert
- Sisteme cadru pentru dezvoltarea sistemelor expert

Istoric

- Anii ’80-’90 – dezamăgire
- Anii ’90-’00 – relansare IA
- **IA distribuită**
- **Agenți inteligenți**
- **Sisteme multi-agent**
- Anii ’90-’00 - apariția învățării profunde
- **Anii 2010 – 2021-** din ce în ce mai multe sisteme sau componente – inteligente

Istoric

Deep learning

- 1989 – CNN cu backpropagation (Yann LeCun)
- 1997 – LSTM (Sepp Hochreiter and Jürgen Schmidhuber)
- 2006 – Deep Belief Networks (Geoffrey Hinton, Ruslan Salakhutdinov)
- 2008 – use of GPU pentru DNN
- 2009 – ImageNet (Fei-Fei Li)

Deep learning

- 2011 Andrew Ng, Google Brain -Rețea neurală antrenata cu algoritmi deep learning – recunoașterea conceptelor cum ar fi pisica pe baza video din YouTube
- Yann LeCun, Facebook - deep learning pentru identificarea fețelor și a obiectelor în 350 milioane de fotografii incarcate pe Facebook
- 2012 – AlexNet – CNN pt ImageNet (Alex Krizhevsky)
- 2014 – GAN (Ian Goodfellow)

Actual

- Machine learning
- Combinare simbolic cu non-simbolic
- IA omniprezentă
- Strong AI

5. Domeniile IA

- Învățare automată
- Vedere computerizată
- Înțelegerea limbajului natural
- Sinteza automată a vorbirii
- Reprezentarea cunoștințelor
- Raționament automat
- Rezolvarea problemelor prin căutare
- Agenți inteligenți și sisteme multi-agent
- Robotică
- Demonstrarea automată a teoremelor
- Expertiză: inginerie, medicină, analiză financiară, sisteme de suport a deciziei, predicție, etc.

6. Aplicații

Chatbots

Alice agent (bot)

A. L. I. C. E. The Artificial Linguistic Internet Computer Entity

Chatbots

Siri, Alexa

Endurance: A Companion for Dementia Patients

Casper: Helping Insomniacs Get Through the Night

Disney: Solving Crimes with Fictional Characters

UNICEF: Helping Marginalized Communities Be Heard

MedWhat: Making Medical Diagnoses Faster

Jocuri

- In 1997 super-computerul *Deep Blue* l-a învins pe campionul de șah Gary Kasparov.
- Actual, calculatoarele pot juca șah mai bine decât orice campion

- 2016 – AlphaGo (DeepMind) – învinge un campion profesional al jocului de Go (Lee Sedol)

Watson

- **Watson**
- Proiectul IBM DeepQA
- Întrebari în limbaj natural pentru quiz show *Jeopardy*
- În 2011, Watson a concurat cu Brad Rutter and Ken Jennings și a obținut locul I - $1 million.
- 200 milioane de pagini, 4 terabytes de disc

- **$400** : With much "Gravity", this young fellow of Trinity became the Lucasian Professor of Mathematics in 1669

- **$400** : Some hedgehogs enter periods of torpor; the Western European species spends the winter in this dormant condition

- **$400** : This title gal, "Children at your feet, wonder how you manage to make ends meet“

- **$800** : The first flight takes place at Kitty Hawk & baseball's first World Series is played

Competiția Robocup

Can football-playing robots beat the World Cup winners by 2050?
https://www.robocup.org/

Mașini autonome

DARPA Grand Challenge 2004, 2005

DARPA Urban Challenge 2007

2012 – DARPA Robotics Challenge

2018-2021 – DARPA Subterranean Challenge

General Motors, Uber, Waymo, Toyota, Honda, Tesla
Waymo – 20 mil mile
General Motor Cruise – ½ mil mile
Accidente 2018: Uber, Tesla

Google, 2015

Waymo, 2017

Uber, San Francisco, 2017

Nvidia self-driving chip Xavier

Genral Motors Cruise

MIT + Stanford + Toyota

Volkswagen

Bosh + Daimler

General Motors' Cruise, San Francisco 2020

Xavier, 2018

AutoDrive, UPB, AIMAS

https://drive.google.com/drive/folders/1pu0_naqyv61Rx7jLTeKzeuORE3LPuSww

7. Impactul Inteligenței Artificiale

- Majoritatea studiilor a subliniat faptul că Inteligența Artificială va avea un ***impact economic și social*** semnificativ la nivel global

Impactul Inteligenței Artificiale

Un studiu al PricewaterhouseCoopers din 2019 estimează că PIB-ul global ar putea crește cu până la 14% până în 2030 ca rezultat al introducerii tehnologiei IA, în special pe baza cantității imense de date care se produce, de exemplu datele care se vor produce prin răspândirea pe scară largă a IoT

Estimarea câștigului pe baza introducerii IA în economie în diferite regiuni ale globului (Sursă: The macroeconomic impact of artificial intelligence, PricewaterhouseCoopers, 2018)

Conectivitatea, disponibilitatea informației și inteligența artificială – a patra revoluție industrială (conform www.worldbank.org)

Source: Deloitte Research

Evoluția cifrei de afaceri a pieței mondiale de Inteligență Artificială
Conform Deloitte Research (2019)

Cifra de afaceri pentru aplicațiile de IA în funcție de industrie conform Deloitte Research (2019)

Leaderi din industrie creează un parteneriat pentru IA

- Amazon, DeepMind/Google, Facebook, IBM și Microsoft au creat o organizație non-profit pentru a promova înțelegerea tehnologiilor IA de către marele public și pentru a formula bune practici și oportunități în domeniul IA
- Partnership on Artificial Intelligence to Benefit People and Society

  https://www.partnershiponai.org/

Inițiative ale Uniunii Europene

- **EU Declaration on Cooperation in Artificial Intelligence, 2018**

https://ec.europa.eu/jrc/communities/en/community/digitranscope/document/eu-declaration-cooperation-artificial-intelligence

- **White Paper on Artificial Intelligence: A European approach focused on excellence and trust, 2020**

https://ec.europa.eu/info/sites/info/files/commission-white-paper-artificial-intelligence-feb2020_en.pdf

- **Confederation of EU AI laboratories**

https://claire-ai.org/

- **EU-Robotics**

https://www.eu-robotics.net/

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 2

Strategii de căutare

- Reprezentarea soluției problemei
- Strategii de căutare de bază
- Strategii de căutare informate
- Strategii de căutare informate cu memorie limitată
- Determinarea funcției euristice

1 Reprezentarea soluției problemei

- Grafuri reprezentate explicit
- Grafuri reprezentate implicit

- Graful asociat unei probleme de cautare
- Graf neponderat sau graf ponderat

Reprezentarea soluției problemei

- Reprezentare prin spatiul starilor
- Reprezentare prin grafuri SI/SAU (AND-OR)
- Echivalenta reprezentarilor
- Caracteristicile mediului de rezolvare

- Pentru a reprezenta si gasi o solutie:
  - Structura simbolica
  - Instrumente computationale
  - Metoda de planificare

1.1 Rezolvarea problemei reprezentată prin spațiul stărilor

- (S_i, O, S_f)
- Stare
- Spatiu de stari (S_Set)
- Stare initiala
- Stare/stari finala/finale
- Solutia problemei

8-puzzle

S_i

2		3
1	8	4
7	6	5

(a) Stare initiala

$\longrightarrow$

S_f

1	2	3
8		4
7	6	5

(b) Stare finala

SUS - Mutare patrat liber in sus
STINGA - Mutare patrat liber la stinga
JOS - Mutare patrat liber in jos
DREAPTA - Mutare patrat liber la dreapta

(c) Operatori

S_i

2		3
1	8	4
7	6	5

STINGA ↙ JOS ↓ DREAPTA ↘

S_1

	2	3
1	8	4
7	6	5

S_2

2	8	3
1		4
7	6	5

S_3

2	3	
1	8	4
7	6	5

(d) Tranzitii posibile din starea S_i

15-puzzle

(n^2-1)-puzzle

General sliding tile puzzle

Donkey' puzzle

Century puzzle

Dad' puzzle

Block world

(a) "Pick up a big red block."

(b) "Find a block which is taller than the one you are holding and put it into the box."

Probleme cu cost

- Planificarea rutelor
- Problema comisvoiajorului

1.2 Rezolvarea problemei reprezentată prin grafuri SI/SAU

- (P_i, O, P_e)
- Nod problema
- Spatiul problemelor (P_Set)
- Problema initiala
- Problema elementara / probleme elementare
- Semnificatie graf SI/SAU (AND-OR)
- Nod rezolvat
- Nod nerezolvabil
- Solutia problemei

Graf SI/SAU

Turnurile din Hanoi

(a) Stare initiala (b) Stare finala

(c) Arborele SI/SAU de descompunere in subprobleme

PROLOG

```
parinte(ada,gelu)
parinte(gelu, misu).
str1(X, Z) :- parinte(X, Z).
str1(X, Z) :- parinte(X, Y), str1(Y, Z)
?-str1(ada,misu).
```

Echivalența reprezentărilor

S_j, S_k - stari intermediare S_f - stare finala

(a) Spatiul starilor (b) Descompunerea problemei in subprobleme

1.3 Caracteristicile mediului

- Observabil / neobservabil
- Discret / continuu
- Finit / infinit
- Determinist / nedeterminist

2. Strategii de căutare de bază

Criterii de caracterizare

- Completitudine
- Optimalitate
- Complexitate: timp, spatiu
- Capacitatea de revenire
- Informare

Costuri ale cautarii

2.1 Căutari neinformate

- Gasirea unei cai sau a tuturor cailor, cu cost sau fara cost

Grafuri specificate explicit

- **Cautarea pe nivel** si **cautarea in adancime** - parcurgerea se face in ordinea nodurilor succesoare starii curente in cautare:
  - Cautarea pe nivel – cele mai apropiate intai
  - Cautarea in adancime – cele mai departate intai
- **Algoritmul lui Dijkstra** rezolva problema cai de cost minim daca toate costurile arcelor sunt ≥ 0
- **Algoritmul Bellman-Ford** rezolva problema cai de cost minim dar costurile arcelor pot fi si negative (de la un nod sursa la toate)
- **Algoritmul Floyd-Warshall** rezolva problema gasirii tuturor cailor de cost minim, costuri pozitive si negative dar cicluri ne-negative

Căutari neinformate pentru grafuri specificate implicit / probleme de cautare

- Cautarea pe nivel (Breadth First Search - BFS)
- Cautare in adancime (Depth First Search - DFS)
- Cautare in adancime cu nivel iterativ (Iterative Deepening - ID)
- Cautare de tip backtracking
- Cautare bidirectionala

Căutari neinformate în spațiul stărilor

- Intr-o reprezentare a solutiei problemei prin spatiul starilor *adancimea unui nod* se defineste astfel:
  - $Ad(S_i) = 0$, unde S_i este nodul stare initiala,
  - $Ad(S) = Ad(S_p)+1$, unde S_p este nodul predecesor nodului S.
- Cele mai multe implementari bazate pe utilizarea a 2 liste: OPEN si CLOSED
- DFS – Open – stiva (LIFO)
- BFS – Open – coada (FIFO)
- In cele mai multe implementari CLOSED este implementata ca o tabela de dispersie (Hash)

Algoritm BFS / DFS(AdMax): Cautare pe nivel/in adancime in spatiul starilor

1. Initializeaza listele OPEN $\leftarrow$ {S_i}, CLOSED $\leftarrow$ {}
2. **daca** OPEN = {}
   atunci intoarce INSUCCES
3. Elimina primul nod S din OPEN si insereaza-l in CLOSED
4. **daca** S $\in$ OPEN $\cup$ CLOSED (inainte de inserare S)
   atunci repeta de la 2

4'. **daca** Ad(S) = AdMax **atunci repeta de al 2** /* pt DFS */

5. Expandeaza nodul S
   5.1. Genereaza toti succesorii directi S_j ai nodului S
   5.2. **pentru** fiecare succesor S_j al lui S **executa**
      5.2.1. Stabileste legatura $S_j \rightarrow S$
      5.2.2. **daca** S_j este stare finala
      atunci
         i. Solutia este (S_j,.., S_i)
         ii. **intoarce** SUCCES
      5.2.3. Insereaza S_j in OPEN, ***la sfarsit*** / ***la inceput***
6. **repeta de la** 2

sfarsit.

Algoritm DFS(AdMax): Cautare in adancime cu adancime limitata

Intrari: Starea initiala S_i

Iesiri: SUCCES *si solutia sau* INSUCCES sau *AdMax*

1. Initializeaza lista OPEN $\leftarrow \{S_i\}$,
2. **daca** OPEN = {} **atunci intoarce** INSUCCES
3. Elimina primul nod S din OPEN
4. **daca** S este stare finala
   atunci i. Solutia este (S,.., S_i)
   ii. **intoarce** SUCCES
5. **daca** Ad(S) > AdMax **atunci intoarece** AdMax
6. Expandeaza nodul S
   6.1. Genereaza toti succesorii directi S_j ai nodului S
   6.2. **pentru** fiecare succesor S_j al lui S **executa**
      6.2.1 Stabileste legatura $S_j \rightarrow S$
      6.2.2. **daca** $S_j \notin$ OPEN
      atunci introduce S_j in OPEN
7. **repeta de la 2**

sfarsit.

Algoritm Iterative Deepening: Cautare in adancime cu nivel iterativ

Intrari: Starea initiala S_i

Iesiri: SUCCES *si solutia sau* INSUCCES

1. **pentru** AdMax = 0 la inf **repeta**

    1.1 **rezultat** = DFS(AdMax)

    1.2 **daca** rezultat <> AdMax

    atunci intoarece rezultat

sfarsit

Abordare hibrida

Căutari neinformate în grafuri SI/SAU

- Se folosesc notiunile de nod rezolvat, nerezolvabil
- BFS, DFS
- Notiunea de adancime a unui nod este diferita
- Intr-o reprezentare a solutiei problemei prin grafuri SI/SAU *adancimea unui nod* se defineste astfel:
  - $Ad(S_i) = 0$, unde S_i este nodul problema initiala,
  - $Ad(S) = Ad(S_p) + 1$ daca S_p este nod SAU predecesor al nodului S,
  - $Ad(S) = Ad(S_p)$ daca S_p este nod SI predecesor al nodului S.

Algoritm BFS-AND-OR: Cautare pe nivel in grafuri SI/SAU

1. Initializeaza listele OPEN $\leftarrow \{S_i\}$, CLOSED $\leftarrow \{\}$
2. Elimina primul nod S din OPEN si insereaza-l in CLOSED
3. Expandeaza nodul S

   3.1. Genereaza toti succesorii directi S_j ai nodului S

   3.2. **pentru** fiecare succesor S_j al lui S **executa**

   3.2.1. Stabileste legatura $S_j \rightarrow S$

   3.2.2. **daca** S_j reprezinta o multime de cel putin 2 subprobleme

   atunci /* este nod SI */

   i. Genereaza toti succesorii subprobleme S^k_j ai lui S_j

   ii. Stabileste legaturile intre nodurile $S^k_j \rightarrow S_j$

   iii. Insereaza nodurile S^k_j in OPEN, ***la sfarsit***

   3.2.3. **altfel** insereaza S_j in OPEN, ***la sfarsit***

4. **daca** nu s-a generat nici un succesor al lui S in pasul precedent (3)

atunci

4.1. **daca** S este nod terminal etichetat cu o problema neelementara

atunci

4.1.1. Eticheteaza S nerezolvabil

4.1.2. Eticheteaza cu nerezolvabil toate nodurile predecesoare lui S care devin nerezolvabile

datorita lui S

4.1.3. **daca** nodul S_i este nerezolvabil

atunci intoarce INSUCCES /* problema nu are solutie */

4.1.4. Elimina din OPEN toate nodurile care au predecesori nerezolvabili

4.2. **altfel** /* S este nod terminal etichetat cu o problema elementara */

4.2.1. Eticheteaza S rezolvat

4.2.2. Eticheteaza cu rezolvat toate nodurile predecesoare lui S care devin rezolvate datorita lui S

4.2.3. **daca** nodul S_i este rezolvat

atunci

i. Construieste arborele solutie urmarind legaturile

ii. **intoarce** SUCCES /* s-a gasit solutia */

4.2.4. Elimina din OPEN toate nodurile rezolvate si toate nodurile care au predecesori rezolvati

5. **repeta de la** 2

sfarsit.

2.2 Complexitatea strategiilor de căutare

- B - *factorul de ramificare* al unui spatiu de cautare

8-puzzle

- Numar de miscari:
- 2 m pt colt = 8
- 3 m centru lat = 12
- 4m centru $\Rightarrow$ **24 miscari**
- **B = nr. misc. / nr. poz. p. liber = 2.67**
- Numar de miscari:
- 1 m pt colt = 4
- 2 m centru lat = 8
- 3m centru $\Rightarrow$ **15 miscari** $\Rightarrow$ **B = 1.67**

Complexitatea strategiilor de căutare

- **B** - *factorul de ramificare*
- **d**- adancimea celui mai apropiat nod solutie (de cost minim daca exista costuri)
- **m** – lungimea maxima a oricarei cai din spatiul de cautare

Rad – B noduri, B^2 pe niv 2, etc.

- Numarul de stari posibil de generat pe un nivel de cautare *d* este B^d
- T - numarul total de stari generate intr-un proces de cautare, *d* – adancime nod solutie

$$T = B + B^2 + \ldots + B^d = O(B^d)$$

Complexitatea strategiilor de căutare

- **Cautare pe nivel**

  Numar de noduri generate

  $B + B^2 + \ldots + B^d = O(B^d)$

- **Cautare in adancime**

  Numar de noduri generate - **B*d** – daca nodurile expandate se sterg din CLOSED

  Sau B*m cu AdMax=m

Complexitatea strategiilor de căutare

- **Cautare backtracking**

  Numar de noduri generate **m**

- **Cautare in adancime cu nivel iterativ**

  Numar de noduri generate

  $$d*B+(d-1)*B^2+ \ldots + (1)*B^d = O(B^d)$$

Complexitatea strategiilor de căutare

Criteriu	Nivel	Adancime	Adanc. limita	Nivel iterativ	Bidirectionala
Timp	B^d	B^d	B^m	B^d	$B^{d/2}$
Spatiu	B^d	B*d	B*m	B^d	$B^{d/2}$
Optimalitate?	Da	Nu	Nu	Da	Da
Completa?	Da	Nu	Da daca $m \geq d$	Da	Da

B – factor de ramificare, **d** – adancimea solutiei,
m – adancimea maxima de cautare (AdMax)

3. Strategii de căutare informate

Cunostintele euristice pot fi folosite pentru a creste eficienta cautarii in trei moduri:

- Selectarea nodului urmator de expandat in cursul cautarii
- In cursul expandarii unui nod al spatiului de cautare se poate decide pe baza informatiilor euristice care dintre succesorii lui vor fi generati si care nu
- Eliminarea din spatiul de cautare a anumitor noduri generate

3.1 Căutare informata de tip "best-first"

- Evaluarea cantitatii de informatie
- Calitatea unui nod este estimata de *functia de evaluare euristica*, notata **w(n)** pentru nodul n
- O functie euristica w(n) este o evaluare a unui nod care mapeaza acel nod la o valoare reala >=0
- Exemple
  - Strategia de cautare “best-first”
  - Strategia de cautare a alpinistului

Algoritm BestFS: Cautare "best-first" in spatiul starilor

Intrari: Starea initiala S_i *si functia* w(S) *asociata starilor*

Iesiri: SUCCES *si solutia sau* INSUCCES

1. Initializeaza listele OPEN $\leftarrow$ {S_i}, CLOSED $\leftarrow$ {}
2. Calculeaza w(S_i) si asociaza aceasta valoare nodului S_i
3. **daca** OPEN = {}

   atunci intoarce INSUCCES
4. Elimina nodul S cu w(S) minim din OPEN si insereaza-l in CLOSED
5. **daca** S este stare finala

   atunci

   i. Solutia este (S,.., S_i)

   ii. **intoarce** SUCCES
6. Expandeaza nodul S

   6.1. Genereaza toti succesorii directi S_j ai nodului S

6.2. **pentru** fiecare succesor S_j al lui S **executa**

6.2.1 Calculeaza $w(S_j)$ si asociaza-l lui S_j

6.2.2. Stabileste legatura $S_j \rightarrow S$

6.2.3. **daca** $S_j \notin OPEN \cup CLOSED$

atunci introduce S_j in OPEN cu $w(S_j)$ asociat

6.2.4. **altfel**

i. Fie S'_j copia lui S_j din OPEN sau CLOSED

ii. **daca** $w(S_j) < w(S'_j)$

atunci

- Elimina S'_j din OPEN sau CLOSED

(de unde apare copia)

- Insereaza Sj cu w(Sj) asociat in OPEN

iii. **altfel** ignora nodul S_j

7. **repeta de la 3**

sfarsit.

Cazuri particulare

- Strategia de cautare "best-first" este o generalizare a strategiilor de cautare neinformate

  - strategia de cautare pe nivel w(S) = Ad(S)

  - strategia de cautare in adincime w(S) = -Ad(S)

- Strategia de cautare de cost uniform / Dijkstra

$$w(S_j) = \sum_{k=i}^{j-1} cost_arc(S_k, S_{k+1})$$

- Minimizarea efortului de cautare – cautare euristica

  w(S) = functie euristica

3.2 Căutarea soluției optime în spațiul starilor. Algoritmul A*

w(S) devine **f(S)** cu 2 comp:

- g(S), o functie care estimeaza costul real g*(S) al caii de cautare intre starea initiala S_i si starea S,
- h(S), o functie care estimeaza costul real h*(S) al caii de cautare intre starea curenta S si starea finala S_f.
- f(S) = g(S) + h(S)
- f*(S) = g*(S) + h*(S)

Calculul lui f(S)

- Calculul lui g(S)

$$g(S) = \sum_{k=i}^{n} cost_arc(S_k, S_{k+1})$$

- Calculul lui h(S)
- Trebuie sa fie admisibila
- O functie euristica h se numeste *admisibila* daca pentru orice stare S, **$h(S) \leq h^*(S)$ & $h(S_f)=0$**
- Definitia stabileste ***conditia de admisibilitate*** a functiei h si este folosita pentru a defini ***proprietatea de admisibilitate*** a unui algoritm A*.

A* - proprietatea de admisibilitate

Fie un algoritm A* care utilizeaza cele doua componente **g** si **h** ale functiei de evaluare **f**. Daca

(1) functia **h** satisface conditia de admisibilitate

(2) $\text{cost_arc}(S,S') \geq c$

pentru orice doua stari S, S', unde **c** > 0 este o constanta si costul **c** este finit

atunci **algoritmul A* este *admisibil***, adica este garantat sa gaseasca calea de cost minim spre solutie.

- Completitudine – garantat sa gaseasca solutie daca solutia exista si costurile sunt pozitive

Implementare A*

Strategia de cautare "best-first" se modifica:

…

2. Calculeaza **$w(S_i)=g(S_i) + h(S_i)$** si asociaza aceasta valoare nodului S_i
3. **daca** OPEN = {}

   atunci intoarce INSUCCES - *nemodificat*
4. Elimina nodul S cu **w(S)** minim din OPEN si insereaza-l in CLOSED - *nemodificat*

…..

6.2.4. **altfel**

i. Fie S'_j copia lui S_j din OPEN sau CLOSED

ii. **daca** **$g(S_j) < g(S'_j)$**

atunci …

h(S) aproape de h*(S)?

- Fie doi algoritmi A*, A1 si A2, cu functiile de evaluare h_1 si h_2 admisibile, $g_1=g_2$

$$f_1(S) = g_1(S) + h_1(S) \qquad f_2(S) = g_2(S) + h_2(S)$$

- Se spune ca algoritmul **A2 este *mai informat* decat** algoritmul **A1** daca pentru orice stare S cu $S \neq S_f$

$$h_2(S) > h_1(S)$$

h_2 domina h_1

Consistenta si monotonia functiei euristice

- **Euristica consistenta**

O functie euristica h este **consistenta** daca

$$h(S) \leq h(S') + cost_arc(op, S, S') \ \& \ h(S_f)=0$$

pentru orice doua stari S si S' succesor al lui S din spatiul de cautare

- **Euristica monotona**

O functie euristica h este **monotona** daca

$f(S_j) \geq f(S_i)$ pentru orice $j > i$, $0 \leq i,j \leq n$, cu $S_n=S_f$

si $f(S)=g(S)+h(S)$

Estimarea costului total al caii de cautare este nedescrescatoare de la un nod la succesorii lui

Consistenta si monotonia functiei euristice

Teorema 1

O functie euristica h este **consistenta** daca si numai daca este **monotona**

Consistenta si monotonia functiei euristice

Teorema 2

O functie euristica **consistenta** este **admisibila**

Consistenta si monotonia functiei euristice

- Maximul a 2 functii admisibile este o fct admisibila
- Maximum a 2 functii consistente este o fct consistenta
- Fct euristice admisibile nu sunt necesare si consistente
- Daca ***h este monotona*** atunci avem garantia ca un nod introdus in CLOSED nu va mai fi niciodata eliminat de acolo si reintrodus in OPEN iar implementarea se poate simplifica corespunzator

Relaxarea condiției de optimalitate a algoritmului A*

- O functie euristica h se numeste **ε-admisibila** daca

$$h(S) \le h^*(S) + \varepsilon$$

cu $\varepsilon > 0$

- Algoritmul A* care utilizeaza o functie de evaluare f cu o componenta ***h*** ε -admisibila gaseste intotdeauna o solutie al carei cost depaseste costul solutiei optime cu cel mult ε.
- Un astfel de algoritm se numeste *algoritm A* ε -admisibil* iar solutia gasita se numeste *solutie ε -optimala*.

Relaxarea condiției de optimalitate

- De fapt putem utiliza o functie euristica ne-admisibila
- A* ponderat (Weighted A* search)

$$f(S) = g(S) + W * h(S) , \quad W>1$$

(a)

(b)

4. Strategii de căutare informată cu memorie limitată

- A* se termina intotdeauna gasind o solutie optima si poate fi aplicat pe probleme generale
- Cu toate acestea, cantitatea de memorie necesara creste repede pe masura avansului algoritmului.
- Algoritmi pentru cautare solutiei de cost optim cu memorie limitata sunt:
  - Beam search
  - Depth first iterative deepening cu cost (DFID)
  - Iterative deepening A* (IDA*)
  - Memory bound A* (MA*)
  - Simplified memory bound A* (SMA*)

DFID cu cost

- Cautarea realizeaza BFS cu o serie de DFS care opereaza pe o frontiera de cautare care creste succesiv
- Cautarea in adancime este modificata a.i. sa utilizeze o **limita de cost** in loc de o limita a adancimii
- Fiecare iteratie expandeaza nodurile din interiorul unui **contur de cost** pentru a vedea care sunt nodurile de pe urmatorul contur
- Daca cost arce 1 atunci DFID fara cost

$w \leq LimCost$

DFID cu cost

- Algoritmul utilizeaza **doua limite U si U’** pentru urmatoarea iteratie.
- Apeleaza repetitiv functia DFID care cauta o cale de cost minim **p**.
- DFID actualizeaza **variabila globala U’ la valoarea minima a costului cailor** generate pana intr-un moment al cautarii
- Daca spatiul de cautare nu contine solutia si este infinit, algoritmul nu se termina

Algoritm DFID: Depth first iterative deepening cu cost

Foloseste

- functia iterativa **BuclaDFID**
- functia recursiva **DFID**
- Functia **Expand** pentru generarea succesorilor unui nod
- Functia **Goal** care testeaza daca stare finala
- U' variabila globala

BuclaDFID

Intrari: Starea initiala **s** si functia de cost **w(s)** asociata starilor

Iesiri: Calea de la s la starea finala sau {}

U' ← 0, bestPath ← {} /* initializare limita globala si cale */

cat timp (bestPath = {} si U'≠ inf) **repeta** ***/**** nu s-a gasit solutie, exploreaza */

  U ← U', U' ← inf /* reset limita, init limita globala noua */

  bestPath ←**DFID(s, 0, U)**

intoarce bestPath

sfarsit

DFID(s, g, U)

Intrari: starea s, costul caii g, limita superioara U

Iesiri: calea de la s la starea finala sau {}

Efect lateral: Actualizarea lui U'

```
daca (Goal(s)) atunci intoarce Cale(s)
Suc(s) ← Expand(s)
pentru fiecare v in Suc(s) repeta
    daca    g + w(s,v) ≤ U                    /* starea este in limita U */
    atunci  p ← DFID(v, g+w(s,v), U)
            daca p ≠ {} atunci intoarce (p)        /* s-a gasit solutie */
    altfel                                  /* cost mai mare decat limita veche U */
            daca g + w(s,v) < U'          /* cost mai mic decat limita globala */
            atunci U' ← g + w(s,v)        /* seteaza limita globala noua */
intoarce {}
sfarsit
```

Cautare DFID

Pas	Iteratie	Selectie	Apeluri	U	U'	Obs
1	1	{}	{(a,0)}	0	inf	
2	1	a	{}	0	2	g(b), g(c) si g(d) > U
3	2	{}	{(a,0)}	2	inf	Incepe o noua iteratie
4	2	a	{(b,2)}	2	6	g(c) si g(d) > U
5	2	b	{}	2	6	g(e) si g(f) > U
6	3	{}	{(a,0)}	6	inf	Incepe o noua iteratie
7	3	a	{(b,2),(c,6)}	6	10	g(d) > U
8	3	b	{(e,6),(f,6),(c,6)}	6	10	
9	3	e	{(f,6),(c,6)}	6	10	g(f) > U
10	3	f	{(c,6)}	6	10	g(e) > U
11	3	c	{}	6	9	g(d)
12	4	{}	{(a,0)}	9	inf	Incepe o noua iteratie

.....

pentru fiecare v in Suc(s) **repeta**
  daca g + w(s,v) ≤ U
  atunci p ← DFID(v, g+w(s,v), U)
    daca p ≠ {} **atunci intoarce** (p)
  altfel
    daca g + w(s,v) < U' **atunci** U' ← g + w(s,v)

cat timp (bestPath = {} si U'≠ inf) **repeta**
  U ← U', U' ← inf
  bestPath ←DFID(s, 0, U)

IDA*

- Iterative deepening A*
- Bazat pe DFID cu cost
- Garantat sa gaseasca solutia de cost minim
- f(S) = g(S) + h(S)

Algoritm IDA*: Iterative deepening A*

Foloseste

- functia iterativa **BuclaIDA***
- functia recursiva **IDA***
- Functia **Expand** pentru generarea succesorilor unui nod
- Functia **Goal** care testeaza daca stare finala

BuclaIDA*

Intrari: Starea initiala s, functia de cost w(s) si h euristica asociata starilor

Iesiri: Calea de la s la starea finala sau {}

U' ← h(s), bestPath ← {}

cat timp (bestPath = {} si U'≠ inf) **repeta**

  U ← U', U' ← inf

  bestPath ←IDA*(s, 0, U)

intoarce bestPath

sfarsit

IDA*(s, g, U)

Intrari: starea s, costul caii g, limita superioara U

Iesiri: calea de la s la starea finala sau {}

Efect lateral: Actualizarea lui U'

```
daca (Goal(s)) atunci intoarce Cale(s)
Suc(s) ← Expand(s)
pentru fiecare v in Suc(s) repeta
    daca  g + w(s,v) + h(v) ≤ U                /* starea este in limita U */
    atunci
        p ← IDA*(v, g +w(s,v), U)
        daca p ≠ {} atunci intoarce p          /* s-a gasit solutie */
    altfel                                     /* cost mai mare decat limita veche U */
        daca  g + w(s,v) + h(v) < U'           /* cost mai mic decat noua limita  */
        atunci U' ← g + w(s,v) + h(v)          /* actualizez noua limita */
intoarce {}
sfarsit
```

5. Determinarea funcției de evaluare h

- Specifica problemei, determinate manual (hand crafted)

SAU

- Generare automata de euristici
- **Transformare abstracta** prin relaxarea unor restrictii ale problemei
- Pattern databases – precalculeaza si memoreaza distanta pana la solutie intr-un spatiu abstract generat de relaxarea restrictiilor impuse miscarilor/actiunilor
- Ne intereseaza o functie euristica h(s) cat mai apriape de h*(s)
- Am dori si un effort cat mai mic

Exemple de funcții euristice

- **Problema comis-voiajorului**

$$h_1(S) = cost_arc(S_i, S)$$

- h2(S) = costul arborelui de acoperire de cost minim al oraselor neparcurse pana in starea S

Exemple de funcții euristice

- **8-puzzle**

$$h_1(S) = \sum_{i=1}^{8} t_i(S)$$

$$h_2(S) = \sum_{i=1}^{8} \text{Distanta}(t_i)$$

Distanta Manhattan

dx=|nod.x-scop.x|

dy=|nod.y-scop.y|

h(nod) = (dx+dy)

Relaxarea condiției de optimalitate pentru 8-puzzle

- 8-puzzle

$$f_3(S) = g(S) + h_3(S) \qquad h_3(S) = h_2(S) + 3 \cdot T(S)$$

$$T(S) = \sum_{i=1}^{8} Scor[t_i(S)]$$

$$Scor[t_i(S)] = \begin{cases} 2 & \text{daca patratul } t_i \text{ in starea S nu este urmat de} \\ & \text{succesorul corect din starea finala} \\ 0 & \text{pentru orice pozitie a lui } t_i \text{ diferita de centru} \\ 1 & \text{pentru } t_i \text{ aflat la centrul mozaicului} \end{cases}$$

Transformari abstracte pt functii euristice

- O transformare abstracta F:S→S’, spatiu abstract, mapeaza stari ***s*** din S (problema reala) in stari F(s) din S’ si mutari/actiuni ***a*** din problema reala in actiuni in spatial abstract
- Daca distanta intre oricare 2 stari F(u) si F(v) in spatial abstract este mai mica sau egala decat distanta intre u si v, atunci aceasta distanta poate fi utilizata ca o euristica admisibila
- Calculate pe parcursul cautarii sau stocate in pattern databases

Transformari abstracte pt functii euristice

- **Variante “relaxate” ale problemei**

O piesa poate fi mutata de la X la Y daca X si Y sunt adiacente si X este blank

O piesa poate fi mutata de la X la Y daca X si Y sunt adiacente

$$h_2(S) = \sum_{i=1}^{8} \text{Distanta}(t_i)$$

O piesa poate fi mutata de la X la Y

$$h_1(S) = \sum_{i=1}^{8} t_i(S)$$

Cum putem găsi o funcție euristică?

- A* in jocuri pt parcurgerea unui teritoriu

Distanta Manhattan

dx=|nod.x-scop.x|

dy=|nod.y-scop.y|

h(nod) = (dx+dy)

Distanta Euclidiana

dx=|nod.x-scop.x|

dy=|nod.y-scop.y|

h(nod) = rad(dx^2+dy^2)

Cum putem găsi o funcție euristică?

- **Pattern database pt euristici**
- Memoreaza o colectie de solutii ale unor subprobleme care trebuie rezolvate pentru a rezolva problema
- Fiecare solutie de subproblema are o functie euristica precalculata (costul cautarii) si memorata
- **Pattern** – o specificare partiala a unei stari
- **Target pattern** – a specificare partiala a starii scop

Pattern database pt euristici

- **Pattern database** – multimea tuturor pattern-urilor care pot fi obtinute prin relaxari sau permutari ale target pattern
- Pentru fiecare pattern din baza de date calculam distanta (nr minim de mutari) fata de target pattern folosind analiza inversa.
- Distanta este costul pattern-ului

Pattern database pt euristici

- 15 puzzle
- Baza de date va indica numarul minim de mutari necesare pt a duce la locul bun 3, 7, 11, 12, 13, 14 si 15; apoi se rezolva 8-puzzle (albastru)
- 31 mutari pt a rezolva piesele rosii (22 mutari pentru a rezolva piesele albastre)

Cubul lui Rubik

- 9 patrate cu 6 culori diferite
- Cea mai buna solutie IDA*

Cubul lui Rubik

Euristici

- ***Distanta Manhattan 3D*** = Calculeaza distanta liniara intre 2 puncte in R3 prin insumarea distantelor punctului in fiecare dimensiune
- Distanta Manhattan 3D intre punctele p1 si p2

  $MD3d(p1, p2) = |x1-x2|+|y1-y2|+|z1-z2|$
- Poate fi calculata in timp liniar
- Trebuie impartita la 8 pentru a fi admisibila deoarece fiecare miscare muta 4 colturi si 4 muchii

Cubul lui Rubik

- *Dureaza mult*
- Adancime 18 – aprox 100 ani
- **Pattern database**
- Se memoreaza intr-o tabela numarul de miscari necesare pt a rezolva colturile cubului sau subprobleme

Cea mai buna?

- Avem mai multe euristici bune
- Pe care o alegem?
- $h(n) = \max(h1(n), \ldots h_k(n))$

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 3

Strategii de cautare

- Cautari locale
- Cautari on-line
- Problema satisfacerii restrictiilor (CSP)

1. Strategii de cautare locala

- **Cautari locale** – opereaza asupra starii curente generand succesorii
- Calea catre solutie nu are importanta
- Gasire solutie – CSP - nu conteaza calitatea solutiei
- Gasire solutie optima – functie de evaluare sau de cost
- Probleme in gasirea solutiei optime pe baza de cautari locale in functie de forma spatiului de cautare
- Maxim global, maxim local, platou, umar

Cautari locale

Caracteristici

- Folosesc putina memorie
- Gasesc solutii destul de bune in spatii finite f mari si chiar in spatii infinite
- Folosite si pt probleme de optimizare (cu fct obiectiv)
- Algoritmi pentru spatii de cautare discrete
  - Hill climbing
  - Simulated annealing

Algoritm: **Hill climbing** (greedy local search)

/* intoarce o stare cu functia de evaluare un maxim local */

1. $S \leftarrow$ stare initiala
2. Genereaza S_j=Succ(S), toti succesorii starii S
3. $S' \leftarrow S_j$ cu Eval(S_j) maxim dintre toti succesorii S_j
4. **daca** Eval(S') $\leq$ Eval(S) **atunci intoarce** S
5. $S \leftarrow S'$
6. **repeta** de la 2

   sfarsit

- Cautarea se orienteaza permanent in directia cresterii valorii;
- Se termina cand ajunge la un maxim (nici un succesor nu are valoarea mai mare)

Hill Climbing

- Problema 8 regine
- S – tabla completa
- Succesori – toate starile posibil de generat prin mutarea a 1 regina intr-un alt patrat in coloana
- o stare are 8 x 7 = 56 succesori
- Fct de evaluare = nr de perechi de regine care se ataca
- Minim global = 0
- 86% instante nu gaseste solutia (3 pasi)
- 14% gaseste soluta (4 pasi)
- Spatiul starilor $8^8 \approx 17$ milioane de stari

Hill Climbing - imbunatatiri

- Miscari laterale – se spera sa fie “umar” si nu “platou” (continui si pentru Eval(S') = Eval(S))
- Daca “platou” – risc de bucla infinita – necesita limita in cautare
- Cu aprox. 100 miscari laterale – problema 8 regine – 94% instante pt care se gaseste solutia

Hill Climbing - imbunatatiri

Hill climbing stohastic

- Dintre starile succesoare cu $Eval(S_j) \geq Eval(S)$, se alege aleator un S_j

First choice hill climbing

- Genereaza aleator succesori pana gaseste $Eval(S_j) \geq Eval(S)$, continua cautarea cu S_j

Random restart Hill Climbing

- Repeta diferite HC cu stari initiale generate aleator
- Daca fiecare HC are o probabilitate de succes de p $\rightarrow$ 1/p repetitii

Simulated annealing

- Simuleaza un proces fizic (calirea)
- T – temperatura
- Scaderea gradientului – solutii de cost minim
- Alege o miscare la intamplare
- Daca starea este mai buna, cauta in continuare de la aceasta
- Altfel alege starea mai "proasta" cu o probabilitate
- Scade probabilitatea de selectie a starilor mai "proaste" pe masura ce temperatura scade

Algoritm: **Cautare Simulated Annealing**

1. T ← temperatura initiala
2. S ← stare initiala
3. v ← Eval(S)
2. **cat timp** T > temp finala **executa**

    2.1 **pentru** i=1,n **executa**

        S' ← Succ(S)

        v' ← Eval(S')

        daca v' < v **atunci S** ← S'

        altfel S ← S' cu prob. exp(-(v'-v)/T)

        (in rest S nemodificat)

    2.2 T ← 0.95 * T

sfarsit

Local beam search

- K stari generate aleator
- Stochastic local beam search

Cautari locale in spatii continue

- Factor de ramificare infinit
- First choice HC, Simulated annealing
- ***Problema:*** dorim sa plasam 3 supermarket-uri pe o harta a.i. suma patratelor distantelor de la fiecare comuna de pe harta la cel mai apropiat supermarket sa fie minima.
- Spatiul starilor este definit prin coordonatele supermarket-urilor

$$(x_1,y_1)\ (x_2,y_2)\ (x_3,y_3)$$

- (spatiu *n*-dimensional cu *n* variabile)
- O miscare in acest spatiu – miscarea unui sm pe harta
- C_i- multimea de orase care sunt cel mai aproape de sm ***i*** in starea curenta

$$f(x_1,y_1,x_2,y_2,x_3,y_3) = \Sigma_{i=1,3}\ \Sigma_{c\in Ci}\ (x_i-x_c)^2+(y_i-y_c)^2$$

Cautari locale in spatii continue

Cum putem gasi solutia?

VARIANTE

- Discretizare spatiu – discretizarea vecinatatii fiecarei stari

(de ex mutam cate un sm o data fie in directa x fie in directia y cu cate o cantitate fixa M => 12 stari succesoare)

- Utilizarea gradientului

Si aplicam HC prin actualizarea starii curente cu x + constanta * gradient

2. Strategii de cautare on-line

- Cautare “offline”
- **Cautare “on-line” – cautare + actiune**
- Exemplu: robot care investigheaza mediul
- **Cautare online:** pentru fiecare actiune, agentul primeste perceptia care ii spune in ce stare a ajuns. Construieste Rezultat[s,a]
- **Cautari locale online:**
  - HC, nu pot random restart,
  - Adauga memorie la HC – memoreaza cea mai buna estimare curenta

Cautare on-line

- Caracteristici

- Metode
  - On-line DFS
  - Programare dinamica asincrona (ADP)
  - Learning Real-Time A* (LRTA*)
  - Cautare cu tinta mobila

2.1 On-line DFS

- Agentul nu are un model al mediului, il descopera pe masura ce il parcurge
- Nodurile sunt expandate pe baza localittaii lor

On-line DFS

On-line DFS(s′) intoarce o actiune

var globale: s,a, starea si actiunea anterioare, initial nule

Rezultat – o tabela care mapeaza (s,a) la s’, initial vida

Neincercate – o tabela care mapeaza s la o lista de actiuni neincercate

Non-revenite – o tabela ce mapeaza s la o lista de stari la care nu s-a revenit inca

daca s’ este stare scop **atunci intoarce** stop

daca s’ este stare noua **atunci** Neincercate[s’]← actiuni(s′) /* Neincercate null pt s’ */

daca s ≠ null **atunci**

Rezultat[s,a] ← s′

adauga s la inceputul lui Non-revenite[s’]

daca Neincercate[s’] este vida **atunci**

daca Non-revenite[s’] este vida **atunci intoarce** stop

a ← o actiune b a.i. Rezultat[s’,b] = pop(Non-revenire[s’])

s’ ← null

altfel a ← pop(Neincercate[s’])

s ← s’

intoarce a

2.2 Programare dinamica asincrona (ADP)

Programare dinamica

Descompunerea problemei in subprobleme ne-distincte

Principiul optimalitatii – o cale este optima ↔ orice segment (subcale) a acesteia este optima

ADP: Daca se cunoaste **h*** pt fiecare nod, calea de cost minim poate fi obtinuta repetand procedura:

- Pentru fiecare nod succesor **j** a nodului curent **i** **calculeaza** $f^*(j)=c(i,j) + h^*(j)$
- Mergi la starea j pt care **f*(j) este minim**

Programare dinamica asincrona

Presupunem urmatoarea situatie

- Pentru fiecare nod **i** exista un proces care corespunde lui **i**
- Fiecare proces inregistreaza **h(i)** – estimarea lui **h*(i)**
- Valoarea initiala a lui **h(i)** este arbitrara cu exceptia nodurilor stare finala
- Fiecare proces **i** actualizeaza **h(i):**
  - pentru fiecare vecin **j - calculeaza f(j)=c(i,j) + h(j)**
  - $\mathbf{h(i) \leftarrow \min_j f(j)}$

ADP

$$f(j)=c(i,j) + h(j)$$

Programare dinamica asincrona

Ordinea de actualizare este arbitrara

Daca costul arcelor este pozitiv, converge la valorile reale

Spatiu de stari mare – nepractic

Foloseste ca fundament pt A* in timp real

- Modific A* pentru a putea fi utilizat on-line
- De ce nu pot folosi direct A*?

2.3 Learning Real-Time A* (LRTA*)

- Intrepatrunde calculul miscarii urmatoare cu executia miscarilor si alege miscarile intr-un timp constant
- Construieste si actualizeaza o tabela ce contine estimari eursitice ale distantei fiecarei stari la starea scop.
- Actualizeaza estimarea de cost a starii pe care a parasit-o si alege miscarea (aparent) cea mai buna
- Initial, intrarile in aceasta tabela corespund unor evaluari euristice sau 0 si sunt subestimari
- Prin explorarea repetata a spatiului, valorile sunt actualizate pana cand ajung, intr-un final, la valorile corecte

1 9 1 2 1 2 1 4 1
1 9 1 3 1 2 1 4 1
1 9 1 3 1 4 1 4 1
1 9 1 5 1 4 1 4 1
1 9 1 5 1 5 1 4 1

Learning Real-Time A* (LRTA*)

Agentul considera numai nodul curent

Agent in nodul **i**

Agentul inregistreaza distanta estimata pentru un nod

1. Lookahead

- Pentru fiecare vecin **j** a lui **i calculeaza**

$$\mathbf{f(j)=c(i,j) + h(j)}$$

h(j) – estimarea caii $j \rightarrow S_f$

c(i,j) – cost arc $i \rightarrow j$

Learning Real-Time A* (LRTA*)

2. Actualizeaza h

- Actualizeaza h estimat a nodului **i**

$$\mathbf{h(i) \leftarrow min_j\, f(j)}$$

3. Selecteaza actiunea si memoreaza

- Mergi la starea **j** cu valoarea **minima f(j)** – actiunea ***a***

- Memoreaza $i \rightarrow_{a} j$

Valorile initiale ale lui **h** trebuie sa fie admisibile

Learning Real-Time A* (LRTA*)

- Agentul aflat in starea $\boldsymbol{s'}$ face backtrack intr-o stare anterior vizitata $\boldsymbol{s}$ in momentul in care estimarea rezolvarii problemei din starea $\boldsymbol{s'}$ + costul de intoarcere in $\boldsymbol{s}$ este *mai mic* decat costul de a merge inainte din starea $\boldsymbol{s'}$.
- In LRTA* meritul fiecarui nod este calculat relativ la pozitia curenta (nu la starea initiala).
- Este suficient sa memoram *tabela hash H* cu $\boldsymbol{h}$ a nodurilor deja vizitate

Learning Real-Time A* (LRTA*)

Algoritm LRTA*(s′) intoarce o actiune

intrari: s′, o perceptie care identifica starea curenta

variabile globale: Rezultat: o tabela indexata dupa stari si actiuni, initial vida

H – o tabela cu estimarea starilor, initial vida

s, a – starea anterioara si actiunea anterioara, initial nule

daca s′ este stare scop **atunci intoarce** stop

daca s′ este stare noua (s′ $\notin$ H) **atunci** H[s′] $\leftarrow$ h(s′)

daca s $\neq$ null **atunci**

Rezultat[s,a] $\leftarrow$ s′

H[s] $\leftarrow$ min $_{b \in \text{Actiuni}(s)}$ **LRTA*Cost**(s, b, Rezultat[s,b], H)

a $\leftarrow$ o actiune b din Actiuni(s′) care minimizeaza LRTA*Cost(s′, b, Rezultat[s′,b], H)

s $\leftarrow$ s′

intoarce a

Agentul selecteaza o actiune in fct de valorile starilor invecinate, care sunt actualizate pe masura ce agentul exploreaza spatiul

Algoritm LRTA*Cost(s,a,s′, H) intoarce o estimare de cost

daca s′ nu este definita **atunci intoarce** h(s)

altfel intoarce c(s,a,s′) + H[s′]

Learning Real-Time A* (LRTA*)

- Intr-un spatiu de cautare finit cu costuri pozitive, in care exista o cale de la orice stare S la S_f si se utilizeaza estimari admisibile nenegative, LRTA* este complet (va ajunge in final in starea scop)
- In plus « invata » solutia optima in timp

2.4 Cautare cu tinta mobila (MTS)

- Generalizare a LRTA*
- Starea scop se schimba pe parcursul cautarii
- MTS – generalizare a LRTA*
- MTS trebuie sa obtina informatie despre locatia starii scop
- Consideram spatiul ca un graf neorientat
- PS nu are harta mediului dar stie pozitia T
- PS stie starile adiactente

Cautare cu tinta mobila

- Presupunem ca PS si T se misca alternativ
- Presupunem ca viteza T este mai mica decat cea a PS (la cativa pasi T sta pe loc)
- Stare finala (scop) = PS si T au aceeasi pozitie
- x, x' – pozitia curenta si pozitia vecinilor pt PS
- y, y' – pozitia curenta si pozitia urmatoare pt T

Cautare cu tinta mobila (MTS)

- Presupun ca toate arcele au ***cost 1***
- Matrice de valori **h(x,y)** – estimatrea distantei intre PS si T
- Matricea poate fi mare dar este suficient sa actualizam numai valorile care se schimba
- h admisibila
- 2 evenimente, fiecare face o actualizare a valorii euristicilor:
  - Miscare a PS
  - Miscare a T

Cautare cu tinta mobila (MTS)

Miscare PS

- Calculeaza $h(x',y)$ pentru fiecare vecin x' a lui x
- Miscare la x' cu $h(x',y)$ minim

  $x \leftarrow x'$
- Actualizeaza valoarea $h(x,y)$ astfel

  $h(x,y) = \max \{ h(x,y), \min_{x'} (h(x',y)+1) \}$

Deoarece orice cale de la x la y trebuie sa treaca prin vecinul x', costul drumului minim de la x la y va fi cel putin la fel de mare ca cel al drumului prin oricare vecin al lui x.

Cautare cu tinta mobila (MTS)

Miscare T

PS observa miscarea T de la y la y′

- Calculeaza $h(x,y')$ pentru noua pozitie y' a lui T
- Actualizeaza valoarea $h(x,y)$ astfel

  $$h(x,y) = \max \{ h(x,y), h(x,y')-1 \}$$
- Actualizeaza starea scop cu noua pozitie a lui T

  $$y \leftarrow y'$$

Daca $h(x,y') > h(x,y)+1$ atunci $h(x,y) \leftarrow h(x,y')-1$
deoarece, distanta y si y′ fiind 1, distanta la vechea pozitie trebuie sa fie cel putin la fel de mare ca distanta la noua pozitie -1

Cautare cu tinta mobila (MTS)

- Intr-un spatiu de cautare finit cu costuri pozitive in care exista o cale de la fiecare stare S la starea scop S_f, daca se porneste cu valori ale functiei euristice admisibile si se permit miscari ale PS si T in orice directie cu cost unitar, PS care executa MTS va ajunge la T daca T sare periodic peste miscari.

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 3

Strategii de cautare

- Cautari locale
- Cautari on-line
- **Problema satisfacerii restrictiilor** (CSP)

3. Problema satisfacerii restrictiilor

$\{X_1 \dots X_N\}$ $\{(X_1, x_1), \dots., (X_N, x_N)\}$

$D = \{D_1 \dots D_N\}$

$R = \{R_1 \dots R_k\}$

- Restrictii explicite sau implicite
- Domeniul de valori
- Restrictii unare, binare, globale

Exemple

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

$$Alldiff(A1, A2, A3, A4, A5, A6, A7, A8, A9)$$
$$Alldiff(B1, B2, B3, B4, B5, B6, B7, B8, B9)$$
$$\ldots$$
$$Alldiff(A1, B1, C1, D1, E1, F1, G1, H1, I1)$$
$$Alldiff(A2, B2, C2, D2, E2, F2, G2, H2, I2)$$
$$\ldots$$
$$Alldiff(A1, A2, A3, B1, B2, B3, C1, C2, C3)$$
$$Alldiff(A4, A5, A6, B4, B5, B6, C4, C5, C6)$$
$$\ldots$$

Instante ale CSP

- Determinarea unei solutii sau a tuturor solutiilor
- CSP totala
- CSP partiala
- CSP binara – graf de restrictii

CSP – problema de cautare, in NP

- Reducerea timpului (reducerea sp. de cautare)
- Metoda de baza: backtracking

3.1 Imbunatatirea performantelor BKT

Algoritmi care modifica spatiul de cautare prin eliminarea unor portiuni care nu contin solutii

- **Algoritmi de imbunatatire a consistentei reprezentarii (utilizati inainte de inceperea cautarii)**
  - Consistenta locala a arcelor sau a cailor in graful de restrictii
- **Algoritmi de cautare (cauta solutia si elimina portiuni din spatiul de cautare)**
  - Imbunatatesc performantele rezolvarii prin reducerea numarului de teste.

Utilizarea euristicilor in cautare

3.2 Propagarea locala a restrictiilor

- Propagarea restrictiilor

$$D_{X_1} = D_{X_2} = D_{X_3} = \{a, b, c\}$$

(a) (b)

Propagarea locala a restrictiilor

$V=\{X,Y,Z\}$ $D_X=D_Y=D_Z=\{1,2,3,4,5,6\}$

$R=\{ X < Y, Z < X-2 \}$

Deoarece $X < Y$ avem $D_X= \{1,2,3,4,5\}$

$D_Y= \{2,3,4,5,6\}$

$D_Z= \{1,2,3,4,5,6\}$

Deoarece $Z < X-2$ avem $D_X= \{4,5\}$

$D_Y= \{2,3,4,5,6\}$

$D_Z= \{1,2\}$

Apoi consideram din nou restrictia $X < Y$

si avem $D_X= \{4,5\}$

$D_Y= \{5,6\}$

$D_Z= \{1,2\}$

Propagarea locala a restrictiilor

- Combinatia de valori ***x*** si ***y*** pentru variabilele X_i si X_j este permisa de restrictia explicita $R_{ij}(x,y)$.

- Un **arc (X_i, X_j)** intr-un graf de restrictii orientat se numeste ***arc-consistent*** daca si numai daca pentru orice valoare $x \in D_i$, domeniul variabilei X_i, exista o valoare $y \in D_j$, domeniul variabilei X_j, astfel incat $R_{ij}(x,y)$.
- **Graf de restrictii orientat *arc-consistent* – orice arc din graf este arc-consistent**

AC-3: Realizarea arc-consistentei pentru un graf de restrictii

/* *Intoarce fals daca inconsistenta, adevarat in caz contrar* */

Creeaza o coada $Q \leftarrow \{ (X_i, X_j) \mid (X_i, X_j) \in$ Multime arce, $i \neq j\}$

cat timp Q nu este vida **executa**

Elimina primul arc (X_k, X_m) din Q

daca Verifica(X_k, X_m) **atunci**

daca $|D_{Xk}| = 0$ **atunci intoarce** fals

$Q \leftarrow Q \cup \{ (X_i, X_k) \mid (X_i, X_k) \in$ Multime arce, $i \neq k,m\}$

intoarce adevarat

Verifica (X_k, X_m) /* *intoarce adevarat daca se modifica D_{Xk}* */

modif $\leftarrow$ fals

pentru fiecare $x \in D_{Xk}$ **executa**

daca nu exista nici o valoare $y \in D_{Xm}$ astfel incat $R_{km}(x,y)$

atunci elimina x din D_{Xk}

modif $\leftarrow$ adevarat

/* Modifica domeniul D_{Xk} prin efect lateral */

intoarce modif

Complexitate

- **N -** numarul de variabile
- **a -** cardinalitatea maxima a domeniilor de valori ale variabilelor
- **e -** numarul de restrictii.
- Algoritmului de realizare a arc-consistentei - *AC-3*: complexitate timp este $\mathbf{O(e*a^3)}$
- S-a gasit si un algoritm de complexitate timp $\mathbf{O(e*a^2)}$ – *AC-4*

m-Cale-consistență

- O cale de lungime *m* prin nodurile $i_0,\ldots,i_m$ ale unui graf de restrictii orientat se numeste ***m-cale-consistenta*** daca si numai daca pentru orice valoare $x \in D_{i0}$, domeniul variabilei i_0 si o valoare $y \in D_{jm}$, domeniul variabilei i_m, pentru care $R_{i0im}(x,y)$, exista o secventa de valori $z_1 \in D_{i1} \ldots z_{m-1} \in D_{im-1}$ astfel incat $R_{i0i1}(x,z_1), \ldots, R_{im-1im}(z_{m-1},y)$
- **Graf de restrictii orientat *m-cale-consistent* – daca orice cale din graf este cale-consistenta**
- Algoritmul de realizare - *PC-4*: complexitatea timp **$O(N^3*a^3)$**
- **2-cale consistenta**
- A face un graf de restrictii m-cale-consistent are o complexitate exponentiala in m

2-Cale-consistență

$V=\{A,B,C\}$ $D_A=D_B=D_C=\{1,2,3\}$

$R=\{B>1, A<C, A=B, B>C-2\}$

Putem exprima restrictiile sub forma de matrici booleene

B>1	A=B	A<C	B>C-2
000	100	011	110
010	010	001	111
001	001	000	111

$R_{i,j}$ – intre X_i si X_j

$R_{k,k}$ – domeniu D

Consistenta pe calea (i,k,j) poate fi detereminata recursiv utilizand ecuatia

$$R_{i,j} \leftarrow R_{i,j} \wedge (R_{i,k} * R_{k,k} * R_{k,j})$$

De ex $R_{AC} \leftarrow R_{A,C} \wedge (R_{A,B} * R_{B,B} * R_{B,C})$

Cale consistență (ordin 2)

/* *Intari: Multime de variabile V, n=|V|, multime de matrici de restrictii R*
Iesiri: Matrici de restrictii cale-consistente */

$Y^n \leftarrow R$

repeta

| $Y^0 \leftarrow Y^n$

| **pentru** k=1,n **repeta**

| | **pentru** i=1,n **repeta**

| | | **pentru** j=1,n **repeta**

| | | | $Y_{ij}^{k} \leftarrow Y_{i,i}^{k-1} \wedge (Y_{i,k}^{k-1} * Y_{k,k}^{k-1} * Y_{k,j}^{k-1})$

pana $Y^n = Y^0$

intoarce Y^0

sfarsit

Este o extindere a algoritmului Floyd Warshall (all shortest path) in care minimizarea este inlocuita cu conjunctia si aduarea cu multiplicarea matricilor

3.3 Cautare cu BKT

- BKT – algoritmul de baza
- Imbunatatiri
- Realizarea arc-consistentei pe parcursul cautarii
- Backjumping

Backtracking

/* *Intrari: variabile V, restrictii R*
Iesiri: Atribuiri pt X si adev daca R satisfacute, fals in caz contrar */
/* *L- variabile instantiate, U – variabile neinstantiate* */

(b,L) ← BKT(V, {}, R)
daca b **atunci intoarce** L
intoarce fals
sfarsit

BKT(U,L,R)
daca U={} **atunci intoarce** (adevarat, L)
X ← Selectie(U)
pentru fiecare x ∈ D_X **repeta**
  b ← Consistent(L∪{X=x},R)
  daca b **atunci**
    (b,L) ← BKT(U\{X}, L ∪{(X,x)},R)
    daca b **atunci intoarce** (b,L)
intoarce fals
sfarsit

Consistent(L,R)
pentru fiecare r ∈R **repeta**
  daca Variabile(r) ⊆ L)
  atunci
    daca not(Satisfacut(r,L)
    atunci intoarece fals
  intoarce adevarat

Realizarea arc-consistentei pe parcursul cautarii

- Forward checking
- **Maintaining Arc-Consistency** (MAC)
- Dupa ce s-a alocat o valoare unei variabile X, algoritmul apeleaza AC-3 dar in loc sa faca o coada cu toate arcele din problema, se introduce in coada numai arcele (X,Y_i) pentru toate variabilele Y_i neinstantiate care sunt legate de X printr-o restrictie
- AC-3 face propagarea restrictiilor si, daca un domeniul este redus la multimea vida, intoarce fals si se intra in backtracking

Backtracking cu MAC

/* *Intrari: variabile V, restrictii R*
Iesiri: Atribuiri pt X si adev daca R satisfacute, fals in caz contrar */
/* *L- variabile instantiate, U – variabile neinstantiate* */

(b,L) ← BKT_MAC(V, {}, R,D)

daca b **atunci intoarce** L

intoarce fals

sfarsit

BKT_MAC(U,L,R,D)

daca U={} **atunci intoarce** (adevarat, L)

X ← Selectie(U)

pentru fiecare $x \in D_X$ **repeta**

  (b,D') ← AC-3'(L ∪{(X,x),R,D)

  daca b **atunci** (b,L) ← BKT_MAC(U\{X}, L ∪{(X,x)},R,D')

    daca b **atunci intoarce** (b,L)

intoarce fals

sfarsit

Backjumping

- BKT cu salt – determina nivelul in care apare eventual un conflict

Exemplu

$V = \{A,B,C,D\}$ $D_A=D_B=D_C=D_D=\{1,2,3,4\}$

$R=\{A > D\}$

Backjumping

/* *Intrari: variabile V, restrictii R*
Iesiri: Atribuiri pt X si adev daca R satisfacute, fals in caz contrar */

(n,L) ← Backjump(V, {}, R, 0) /* *n – variabila intreaga* */
daca n=|V|+1 **atunci intoarce** L
intoarce fals
sfarsit

Backjump(U,L,R,p)

/**Intrari: U, L, R, nivel anterior p, Iesiri: L sau nivelul de salt pt variabila conflict*/

daca U={} **atunci intoarce** (|L+1|, L)
X ← Selectie(U) m ← 0 /* *init variabila jump* */
pentru fiecare x ∈ D_X **repeta**
(b,j) ← Consistent(L∪{(X,x,p+1), R, p+1) /* *b var booleana, j var intraga* */
daca b **atunci**
m ← p
(n,L) ← Backjump(U\{X}, L ∪{(X,x,p+1)}, R, p+1) /* *n intreg* */
daca (n ≠ p+1) **atunci intoarce** (n,L)
altfel
m ← max{m,j}
intoarce (m,.)
sfarsit

Consistent(L,R,nivel)

/* *intoarce nivelul variabilei conflictuale* */

j ← nivel

b ← fals

pentru fiecare r ∈R **repeta**

  daca Variabile(r) ⊆ L)

  atunci

    daca not(Satisfacut(r,L)

    atunci

      b ← adevarat

      j ← min{j, $\max_k$ (X∈Variabile(r) **si** (X,x,l)∈L **si** k < l)}

daca b

atunci intoarce (fals, j)

intoarce (adevrat,.)

sfarsit

3.4 Euristici

Euristici generale

Ordonarea variabilelor – vezi functie Selectie(U)

- aleator
- Minimum remaining value (MRV) – se incepe cu variabila cea mai restrictionata intai (fail-first) – variabila cu cele mai putine valori legale
- Degree heuristic – selectie variabila care este implicata in cel mai mare numar de restrictii cu variabile neinstantiate

Ordonarea valorilor

- Least-constrained value – selectie valoarea care elimina cele mai putine valori din domeniul variabilelor neinstantiate cu care este legata prin restrictii (fail-last)

O solutie: variabila fail-first, valoare fail last

Toate solutiile: variabila fail-last, valoare fail first

3.5 CSP partiala

- Memoreaza cea mai buna solutie gasita pana la un anumit moment (gen IDA*) – *distanta d* fata de solutia perfecta
- Abandoneaza calea de cautare curenta in momentul in care se constata ca acea cale de cautare nu poate duce la o solutie mai buna
- NI - numarul de inconsistente gasite in "cea mai buna solutie" depistata pana la un moment dat – *limita necesara*

CSP partiala

- S - *limita suficienta* - specifica faptul ca o solutie care violeaza un numar de S restrictii (sau mai putine), este acceptabila.
- PBKT(Cale, Distanta,Variabile,Valori)
  - Semnificatie argumente
  - Rezultat: GATA sau CONTINUA
- variabile globale: CeaMaiBuna, NI, S

Algoritm: **CSP Partiala**

PBKT(Cale, Distanta, Variabile, Valori)

/ intoarce GATA sau CONTINUA */*

1. **daca** Variabile = {}
   atunci
   - 1.1 CeaMaiBuna ← Cale
   - 1.2 NI ← Distanta
   - 1.3 **daca** NI ≤ S **atunci intoarce** GATA
     altfel intoarce CONTINUA
2. **altfel**
   - 2.1 **daca** Valori = {} **atunci** CONTINUA

/ s-au incercat toate valorile si se revine la var ant. */*

   - 2.2 **altfel**
     - 2.2.1 **daca** Distanta ≥ NI
       atunci intoarce CONTINUA

/ revine la var ant pentru gasirea unei solutii mai bune*/*

2.2.2 **altfel**

i. Var ← first(Variabile)

ii. Val ← first(Valori)

iii. DistNoua ← Distanta

iv. Cale1 ← Cale

v. **cat timp** Cale1 <> {} **si** DistNoua < NI **executa**

- (VarC, ValC) ← first(Cale1)
- **daca** Rel(Var,Val,VarC,ValC) = fals
- **atunci** DistNoua ← DistNoua + 1
- Cale1 ← Rest(Cale1)

vi. **daca** DistNoua < NI **si**
PBKT(Cale+(Val,Var),DistNoua,Rest(Variabile),ValoriNoi)
= GATA

/* ValoriNoi - domeniul de valori asociat primei variabile din Rest(Variabile) */

atunci intoarce GATA

altfel intoarce
PBKT(Cale,Distanta,Variabile,Rest(Valori)

sfarsit

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 4

Strategii de căutare

- **Căutare adversarială în jocuri**

1. Strategii de căutare în jocuri

- Teoria jocurilor – teoria deciziei pt agenți care interacționează
- Căutări specifice datorita acestei particularități
- S: set de stări cu S_0
- N – număr de jucători
- A – mulțime de acțiuni
- f: S x A $\rightarrow$ S (f sau next)
- Q: S $\rightarrow R^N$ – funcția de utilitate / recompensa
- J: S $\rightarrow$ (1, 2…., N) – jucătorul care joaca

1.1 Jocuri cu 2 adversari

- Jocuri ce implică doi adversari (N=2)
  - Jucător
  - Adversar
- Algoritmul Minimax
- Algoritmul Alfa-Beta
- Algoritmul Monte Carlo Tree Search

Minimax

- 1944 - von Neumann si Morgenstern descriu cum un proces, numit Minimax, este capabil sa identifice rezultatul final al unui joc si sa aleaga mutarea cea mai buna pentru orice stare de joc, în cazul jocurilor cu informatie perfecta.
- Bazele teoretice ale algoritmului – 1928 - von Neumann
- **Non-cooperative Game Theory** / 2 players zero-sum games
- Valoarea **maxmin** a J1 este cea mai mare valoare pe care J1 poate spera sa o obtina fara sa stie actiunea lui J2

$$\mathbf{max}_{J1}\, \mathbf{min}_{J2} \mathbf{u}_{J2}\, (\mathbf{a}_{J1}, \mathbf{a}_{J2})$$

- **Combinatorial game theory /** sequential games with perfect information
- **Algoritmul Minimax**

Minimax

- Jucător – MAX
- Adversar – MIN
- Etichetez fiecare nivel din AJ cu **MAX** (jucator) și **MIN** (adversar)
- Etichetez frunzele cu scorul jucatorului
- Parcurg AJ
  - dacă nodul parinte este **MAX** atunci i se atribuie valoarea maxima a succesorilor sai;
  - dacă nodul parinte este **MIN** atunci i se atribuie valoarea minima a succesorilor sai.

Minimax pentru spații de căutare investigate până la o adancime *n*

- Spatiul de cautare este f mare sau infinit, nu poate fi investigat exhaustiv
- Algoritmul Minimax pana la o adancime *n*
- *nivel(S)*
- O functie euristica de evaluare a unui nod *eval(S)*

Exemplu de funcție de evaluare

Jocul de Tic-Tac-Toe (X si O)

- Functie de estimare euristica **eval(S) -** conflictul existent in starea **S**.
- eval(**S**) = numarul total posibil de linii castigatoare ale lui **MAX** in starea **S** - numarul total posibil de linii castigatoare ale lui **MIN** in starea **S**.
- Daca **S** este o stare din care **MAX** poate face o miscare cu care castiga, atunci eval(**S**) = ∞ (o valoare foarte mare)
- Daca **S** este o stare din care **MIN** poate castiga cu o singura mutare, atunci eval(**S**) = $-\infty$ (o valoare foarte mica).

eval(S) în Tic-Tac-Toe

X are 6 linii castigatoare posibile

O are 5 linii castigatoare posibile

eval(**S**) = 6 - 5 = 1

Algoritmul tăierii alfa-beta

- Este posibil sa se obtină decizia corecta a algoritmului **Minimax** fara a mai inspecta toate nodurile din spatiului de cautare pana la un anumit nivel.
- Procesul de eliminare a unei ramuri din arborele de cautare se numeste *taierea arborelui de cautare* (*pruning*).
- Alpha-beta pruning (Knuth and Moore, 1975)

Algoritmul tăierii alfa-beta

- Fie α cea mai buna valoare (cea mai mare) gasita pentru **MAX** si β cea mai buna valoare (cea mai mica) gasita pentru **MIN**.
- Algoritmul **alfa-beta** actualizeaza α si β pe parcursul parcurgerii arborelui si elimina investigarile subarborilor pentru care α sau β sunt mai proaste.
- Terminarea cautarii (taierea unei ramuri) se face dupa doua reguli:
  - **α -taieri -** În cazul în care exista, pentru un nod **Min**, o actiune ce are asociata o valoare v <= α, atunci putem renunta la expandarea subarborelui sau, deoarece **Max** poate atinge deja un câstig mai mare, dintr-un subarbore precedent.
  - **β -taieri -** În cazul în care exista, pentru un nod de tip **Max**, o actiune ce are asociata o valoare v >= β , atunci putem renunta la expandarea subarborelui sau, deoarece **Min** a limitat deja castigul lui **Max** la β

$\alpha = 3$
3
Max
$\beta = 3$
3
$\beta = 0$
0
2
$\beta = 2$
Min
α-tăiere
α-tăiere
3
0
2
Max
β-tăiere
2
3
2
5
0
2
1
Min

Algoritm: **Alfa-beta**

MAX(S, α, β) { intoarce valoarea maxima a unei stari. }

0. daca S este nod final **atunci intoarce** scor(S)

1. daca nivel(S) = n **atunci** intoarce eval(S)

2. altfel

2.1 **pentru** fiecare succesor S_j al lui S **executa**

2.1.1 $\alpha \leftarrow \max(\alpha, MIN(S_j, \alpha, \beta))$

2.1.2 **daca** $\alpha \geq \beta$ **atunci** intoarce β

2.2 **intoarce** α

sfarsit

MIN(S, α, β) { intoarce valoarea minima a unei stari. }

0. daca S este nod final **atunci intoarce** scor(S)

1. daca nivel(S) = n **atunci intoarce** eval(S)

2. altfel

2.1 **pentru** fiecare succesor S_j al lui S **executa**

2.1.1 $\beta \leftarrow \min(\beta, MAX(S_j, \alpha, \beta))$

2.1.2 **daca** $\beta \leq \alpha$ **atunci intoarce** α

2.2 **intoarce** β

sfarsit

Algoritmul tăierii alfa-beta

- Eficienta algoritmului depinde semnificativ de ordinea de examinare a starilor
- Se recomanda o ordonare euristica a succesorilor, eventual de generat numai primii cei mai buni succesori
- Poate reduce semnificativ timpul de cautare
- *De exemplu – incepe cu cea mai "simpla" miscare sau favorizeaza nodurile cu taieri B*

Imbunatatiri

- Timp limitat pentru executarea unei miscari – anytime algorithm
- Foloseste **Iterative Deepening**
- incepe cu ply 1 si obtine cea mai buna miscare
- apoi ply 2 folosind cele mai bune stari cf evaluarii anterioare
- continua cresterea ply pana la expirarea timpului

Imbunatatiri

- Diferite secvente de mutari pot duce la aceleasi pozitii
- Mai multe pozitii de joc pot fi functional echivalente (de ex pozitiile simetrice)
- **Memoize** – tabela hash cu pozitiile de joc pentru a obtine:
  - Estimari ale nodurilor
  - Cea mai buna miscare dintr-un nod

Imbunatatiri

- **Efectul de orizont** – cauta mai mult decat limita de cautare pentru anumite pozitii

Jocuri cu elemente de șansă

- Jucatorul nu cunoaste mișcările legale ale oponentului
- 3 tipuri de noduri:
  - MAX
  - MIN
  - Șansă (chance nodes)

Noduri șansă – ramurile care pleaca dintr-un nod șansa indica posibile rezultate ale sansei (de exemplu zar)

Noduri de decizie

Noduri sansa

- 36 rez pt 2 zaruri, toate la fel de probabile
- 21 noduri distincte (3-4 la fel cu 4-3)
- Zaruri egale (6 dist) - > 1/36 pt 1-1
- Zaruri diferite (15 dist) -> 1/18 pt zaruri diferite

Functia de evaluare

- scor – nod terminal
- max din Minimax succesori - MAX
- min din Minimax succesori - MIN
- Σ [$P(S_j)$*Minimax(S_j)] succesori - SANSA

Valoarea estimata pt noduri sansa

- $\Sigma_{S_j \text{ suc } S}[P(S_j)*\text{Minimax}(S_j)]$

1.2 Jocuri cu mai mulți jucători

Exemplu: Chinese checkers (table chinezesti)

- Chinese checkers nu este un joc de origine chineza, ci este o variatie moderna si simplificata, aparuta în Germania prin 1892, a celebrului joc american Halma, inventat de George H. Monks în 1880.
- Abia în momentul în care a devenit cunoscut în Statele Unite ale Americii, a primit denumirea Chinese checkers

George Howard Monks

Jocuri cu mai mulți jucători

Chinese checkers

- Este un joc cu informatie perfecta pentru 2-6 jucatori
- Scopul este sa deplasam 10 piese dintr-o pozitie de start intr-o pozitie finala cat mai repede.

- **Chinese checkers**
- Piesele se muta prin deplasare intr-o pozitie alaturata sau sarind peste piese alaturate daca exista un loc liber. Se poate sari peste orice numar de piese si se pot inlantui mai multe sarituri. Piesele nu sunt eliminate dupa sarituri.

Jocuri cu mai mulți jucători

Nu exista algoritmi buni consacrati

In general, 2 strategii

- **Max^n** - generalizare a Minimax pt **n** jucatori
- **Paranoic** – reduce la joc cu 2 jucatori in care se presupune ca toti ceilalti colaboreaza impotriva jucatorului simulat

Presupunem jocuri cu informatie perfecta

Jocuri cu mai mulți jucători

- Functia de evaluare – dependenta de joc
- Regula de decizie in parcurgerea arborelui de cautare – generica

Jocuri cu mai mulți jucători

Strategie Max^n

- Generalizarea Minimax pentru n jucatori
- Frunzele arborelui de joc sunt n-tuple, in care elementul pe pozitia ***i*** este scorul jucatorului ***i***.
- Pentru nodurile din interior, valoarea Max^n a unui nod in care jucatorul ***i*** muta este valoarea Max^n a succesorului pentru care a ***i***-a componenta din vector este maxima.

Strategie Max^n

Maxn(Nod, Juc)

1. daca Nod este nod final **atunci intoarce** scor(Nod[Juc])

2. daca nivel(Nod) = n **atunci** intoarce eval(Nod[Juc])

3. altfel

3.1 P ← Prim_succesor(Nod)

3.2 Best [Juc] ← Maxn (P, Juc_Urm)

3.3 **pentru** fiecare succesor $S_j \neq P$ al lui Nod **executa**

- Curent ← Maxn(S_j, Juc_Urm))
- **if** Curent[Juc] > Best [Juc]

then Best ← Curent

4. Intoarce Best [Juc]

Strategie Max^n

- Pot exista multe valori egale Max^n intr-un arbore
- Rezultatul poate depinde drastic de felul in care se face alegerea
- E.g., (2,3,3) vs. (2,1,7)

- Alpha-beta in adancime nu poate fi aplicat
- Max^n – shallow pruning, Korf, 1991 (analog to alpha-beta dar cu performante proaste)

Strategie Paranoic

- **Paranoic** – reduce jocul la 2 jucatori si se poate aplica Minimax

Strategie Paranoic

- La fel ca la Minimax:
  - Exista o unica valoare
  - Se poate utiliza Alpha-Beta

Strategie Paranoic

- Pe masura ce numarul jucatorilor creste beneficiul adus de taiere scade
- Pt jocuri cu 3-6 jucatori, adancimea este cu 20-50% mai mare decat la Max^n
- Presupunerea Paranoic este foarte pesimista
- Exista cazuri in care greseste; in aceasta situatie, cu cat se cauta mai adanc cu atat este mai proasta estimarea

1.3 Monte Carlo Tree Search

- MCTS - algoritm probabilistic care utilizeaza o serie de simulari aleatoare pentru a expanda selectiv arborele de joc
- Reprezinta o metoda buna pentru luarea deciziilor în probleme cu un spatiu de cautare mare
- Este un fel de best-first search ghidat de rezultatele unei simulari Monte-Carlo
- Metoda se bazeaza pe 2 ipoteze:
  - Adevarata **valoare a unei actiuni** (mutare in joc) poate fi aproximata utilizand simulari aleatoare
  - Valorile astfel obtinute pot fi utilizate pentru a **ajusta politica de selectie** spre o cea mai buna strategie
- MCTS (Coulom, 2006)

Monte Carlo Tree Search

- Baza metodei este o **unda de joc** (“**playout**”)

- **Playout** = un joc rapid jucat cu mutari dintr-o anumita stare pana la sfarsitul jocului, obtinandu-se castig/pierdere sau un scor

- Fiecarui nod parcurs i se asociaza un merit
- In varainta cea mai simpla acest merit este un procent de castig = de cate ori s-a castigat daca s-a pornit unda din acel nod

Monte Carlo Tree Search

- Algoritmul construieste progresiv un **arbore de joc partial**, ghidat de rezultatele explorarilor anterioare ale acestui arbore
- Arborele este utilizat pentru a **estima valoarea miscarilor**, estimarile devenind din ce in ce mai bune pe masura ce arborele este construit
- Algoritmul implica construirea iterativa a arborelui de cautare pana cand o anumita cantitate de efort s-a atins, si intoarce cea mai buna actiune gasita

Monte Carlo Tree Search

Pentru fiecare iteratie se aplica 4 pasi:

- **Selectie** – Pornind de la radacina o politica de selectie a copiilor este aplicata recursiv pana cand se gaseste cel mai interesant nod neexpandat (un nod E care are un copil ce nu este inca parte a arborelui)
- **Expandare** – Un nod copil (sau mai multe noduri copii) a lui E este adaugate in arbore cf. actiunilor disponibile

Monte Carlo Tree Search

- **Simulare** – Se executa o simulare de la nodul/nodurile noi cf. politicii implicite pentru a obtine un rezultat R
- **Backpropagation** – rezultatul simulatii este propagat inapoi catre nodurile care au fost parcurse si se actualizeaza valorile acestora

Monte Carlo Tree Search

Algoritm MCTS(Radacina) **intoarce** cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_l \leftarrow$ TreePolicy(v_0)

$\Delta \leftarrow$ DefaultPolicy(v_l)

BackUp(v_l, Δ)

intoarce a(BestChild(v_0))

- **TreePolicy** – construieste un nod frunza din nodurile aflate deja in arborele de cautare; nodul la care se ajunge cu TreePolicy este v_l
- **DefaultPolicy** – joaca jocul dintr-o stare neterminala v_l pentru a produce o estimare a valorii / recompensa (pana stare terminala)
- **a(BestChild(v_0))** – actiunea care selecteaza cel mai bun copil a lui v_0

Algoritm MCTS(Radacina) **intoarce** cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_l \leftarrow$ TreePolicy(v_0)

$\Delta \leftarrow$ DefaultPolicy(v_l)

BackUp(v_l, Δ)

intoarce a(BestChild(v_0))

TreePolicy(v) **intoarce** un nod

cat timp v este nod neterminal **executa**

daca v nu este complet expandat **atunci** v $\leftarrow$ Expand(v); **break**

altfel v $\leftarrow$ BestChild(v)

intoarce v

Expand(v) intoarce un nod

alege a$\in$actiunile neincercate inca din A(v) /* A(v) act legale in v */

adauga un copil nou v' la v cu v' = next(v,a) /* aleator sau in fct de merit */

intoarce v'

Algoritm MCTS(Radacina) **intoarce** cea mai buna miscare

Creaza nodul radacina v_0 cu starea s_0

cat timp nu s-au epuizat resursele **repeta**

$v_l \leftarrow$ TreePolicy(v_0)

$\Delta \leftarrow$ DefaultPolicy(v_l)

BackUp(v_l, Δ)

intoarce a(BestChild(v_0))

DefaultPolicy(s) **intoarce** recompensa

cat timp s este nod neterminal **executa**

alege aleator $a \in A(s)$ /* sau cf unei politici de selectie */

$s \leftarrow$ next(s,a)

intoarce recompensa pentru starea s

BackUp(v, Δ)

cat timp v nu este null **executa**

$N(v) \leftarrow N(v) + 1$ /* numar vizitari nod v */

$Q(v) \leftarrow Q(v) + \Delta(v)$ /* recompensa nod */

$v \leftarrow p$ /* p parintele lui v */

Monte Carlo Tree Search

Ce strategii se folosesc pt fiecare pas?

- **Selectia**
- Problema exploatarii vs explorare
  - Selectam stari din care s-a castigat des si au fost parcurse de multe ori
  - Ori selectam stari cu putine simulari anterioare

- Diferite strategii propuse in literatura

Monte Carlo Tree Search

Exemplu de strategie pentru selectie

Fie I multimea de noduri succesoare nodului curent **p**. Se selecteaza copilul **K** a nodului **p** care satisface formula

Exploatare

Explorare

$$K \in \arg\max_{i \in I} \left(\frac{Q(i)}{N(i)} + C * \sqrt{\frac{2 * \ln N(p)}{N(i)}} \right)$$

Q(i) - recompensa nodului i

N(i) – numarul de vizitari a nodului i

N(p) – numarul de vizitari a nodului p (parinte)

C – coeficient experimental

BestChild(v) intoarce K

Monte Carlo Tree Search

Expandarea

Se genereaza un nou de unde se va incepe simularea

Simularea

- Total aleator, sau pseudoaleator, sau in functie de o politica

Backpropagation

- Diferite metode

$$V_p = \frac{V_{med} * W_{med} + V_r * N_r}{W_{med} * N_r}$$

$$\Delta(v) = V_p$$

V_{med} – media valorilor nodurilor copii

W_{med} – ponderea acestei medii

V_r – mutarea cu cel mai mare numar de simulari

N_r – numarul de ori de care s-a jucat V_r

Monte Carlo Tree Search

- Converge spre valorile Minimax, dar lent
- Cu toate acestea mai eficient decat AlfaBeta
- Este un algoritm de tip anytime
- Algoritmul nu gaseste întotdeauna cea mai buna mutare, dar are, în general, un succes rezonabil în cazul alegerii mutarilor care duc la sanse mari de câstig
- Poate sa nu detecteze o ramura care conduce la pierdere (din cauza componentei aleatoare)
- Light playouts – aleator
- Heavy playouts – euristici/politici pentru selectarea miscarii urmatoare (nu strict aleator)

Monte Carlo Tree Search

MoGo

A participat in 30 de turnee intre 2006 si 2010 (9 x 9 GO)

A castigat contra profesionistilor

- MoGo invinge pe campionul Myungwan Kim, august 2008, utilizand MCTS
- **FUEGO -** 2009 – a invins mai multi campioni la 9 x 9 GO

Monte Carlo Tree Search

2009 – **MoHex** devine campion la jocul **Hex**

11 x 11

13 x 13

14 x 14

19 x 19

Joc inventat in 1940
de Piet Hein and John Nash

1.4 Invatarea functiei de evaluare pentru selectie

- Functia de evaluare – depinde de parametrii starii si o serie de ponderi ale acestora

$$Eval_w(u)$$

- Daca avem evaluari corecte ale unor stari $Eval^*(u)$ putem aplica **metoda scaderii gradientului** pentru a reduce **eroarea**

$$L = (Eval_w(u) - Eval^*(u))^2$$

$$\Delta w = \alpha\left(Eval^*(u) - Eval_w(u)\right) * \nabla Eval_w(u)$$

Q-Learning

Temporal credit assignment problem

Q-Learning

- **Functie actiune-valoare** = atribuie o utilitate estimata executarii unei actiuni intr-o stare
- Q: A x S $\rightarrow$ U
- Foloseste o matrice (stare,actiune) - valoare

Q-Learning

Foloseste **ecuatia lui Bellman**

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

calculata dupa fiecare tranzitie din s in s'.

α - viteza de invatare

Q-Learning

1. Initializeaza δ si matricea **R**
2. Initializeaza **Q** la 0 sau aleator
3. Pentru fiecare episod repeta
   - Selecteaza starea initiala **s** aleator
   - **cat timp** **s** nu este stare scop **repeta**
     - obtine recompensa **R(s)**
     - selecteaza o actiune **a** din starea **s**
     - executa trecerea in **s'** cu **a**
     - $Q(a,s) \leftarrow$

$$Q(a,s) + \alpha(R(s) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

     - $s \leftarrow s'$

Q-Learning

Utilizare Q

1. s ← starea initiala
2. gaseste actiunea **a** care maximizeaza Q(a,s)
3. Executa **a**
4. **Repeta** de la 2 **pana** stare scop

Deep Q-Learning

- Combina Q-Learning cu DNN
- Reteaua neurala invata estimarea perechilor stare-actiune numai pe baza descrierii starii si a recompensei, fara a sti regulile de joc

Atari Deep Q-Learning

- Deep learning – extrage caracteristici de nivel inalt din imagini

Atari

- O retea convolutionala este antrenata sa invete evaluarea stare-actiune pt a aplica RL
- Intrare: pixelii din imagine
- Iesire: valori Q

Two classic Atari 2600 video games, “Freeway” and “Seaquest”, from the Arcade Learning Environment

Atari Deep Q-Learning

- Selecteaza actiunea
- Actiunea trimisa emulatorului de joc care modifica starea si scorul jocului
- Agentul primeste o imagine (vectori de pixeli) si recompensa = schimbarea scorului
- Considera o secventa de stari si observatii

$$S_t = s_1, a_1, s_2, a_2, .., a_{t-1}, s_t$$

si invata strategia in functie de astfel de secvente

Atari Deep Q-Learning

- Foloseste un Q-network cu parametrii **w** care minimizeaza functia de eroare (**Loss**)

L(w) = Estimare [y - Q(s,a,w)]^2

$$y = R(s) + \delta \max_{a'} Q(s',a')$$

AlphaGo

- Octombrie 2015 - AlphaGo joaca impotriva campionului euroepan Fan Hui si catiga cu un scor de 5-0
- In 2016 castiga in fata lui Lee Sedol, detinator a 18 titluri mondiale
- A obtinut cel mai mare scor
- A inventat mutari
- Foloseste Deep Q-Learning

AlphaGo

- Foloseste:

- MCTS

- 2 retele neurale adanci: una pentru a invata politica de selectie si una pentru a invata evaluarea nodurilor

- Algoritmul de Q-Learning – initializat cu politica invatata si imbunatatita in functie de recompensele viitoare

Un exemplu simplu

- **Jocul Snake**
- Se dau sistemului parametrii legati de stare si recompensa
- Nu necesita reguli de joc pt invatarea politicii
- Trebuie sa maximizeze recompensa

Un exemplu simplu

- **Tabela Q**

Stare	**Right**	**Left**	**Up**	**Down**
1	0	0.31	0.12	0.87
2	0.98	-0,21	0.01	0.14

Se actualizeaza cu Q-Learning si ecuatia lui Bellman

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

1. Initializeaza **Q** aleator
2. Initializeaza **δ** si matricea **R**
3. Pentru fiecare episod repeta

    Obtine starea curenta **s**

    Executa o actiune **a** (aleator sau selectata de RNA)

    Obtine recompensa **R(s)**

    Executa trecerea in **s'** cu **a**

$$Q(a,s) \leftarrow Q(a,s) + \alpha(R(s) + \delta \max_{a'} Q(a', s') - Q(a,s))$$

$$s \leftarrow s'$$

Reprezentare stare pentru retea

11 variabile boolene

- Daca este pericol in jurul sarpelui (Right, Left, Straight)
- Daca sarpele se misca Up, Down, Left, Right
- Daca mancarea este Up, Down, Left, Right

- RNA estimeaza actiunea cea mai buna pentru o anumita stare incercand sa maximizeze recompensa pe baza functiei de eroare (**Loss**)

$$\mathbf{L_i(\theta_i) = [R(s) + \delta \max_{a'} Q(s',a') - Q(s,a,\theta_i)]^2}$$

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 5 si 6

Reprezentarea cunostintelor incerte

- Teoria probabilitatilor
- Retele Bayesiene
- Inferente exacte si aproximative in retele Bayesiene

1. Teoria probabilitatilor

1.1 Cunostinte incerte

$\forall$p simpt(p, Dur_d) $\rightarrow$ factor(p,carie)

$\forall$p simpt(p, Dur_d) $\rightarrow$ factor(p,carie) $\vee$ factor(p,infl_ging) $\vee$…

- LP

- dificultate (« lene »)

- ignoranta teoretica

- ignoranta practica

- Teoria probabilitatilor ➔ un grad numeric de **incredere** sau **plauzibilitate** a afirmatiilor in [0,1]
- Gradul de adevar (fuzzy logic) $\neq$ gradul de incredere

1.2 Definitii TP

- ***Probabilitatea unui eveniment incert A*** este masura gradului de incredere sau plauzibilitatea produceri unui eveniment
- Camp de probabilitate, S
- ***Probabilitate neconditionata*** (apriori) - inaintea obtinerii de probe pt o ipoteza / eveniment
- ***Probabilitate conditionata*** (aposteriori) - dupa obtinerea de probe

Exemple

P(Carie) = 0.1

P(Vreme = Soare) = 0.7

P(Vreme = Ploaie) = 0.2 P(Vreme = Nor) = 0.1

Vreme - **variabila aleatoare**

- ***Distributie de probabilitate***

Definitii TP - cont

- ***Probabilitate conditionata*** (aposteriori) - P(A|B)

  P(Carie | Dur_d) = 0.8

  Masura probabilitatii producerii unui eveniment A este o functie $P:S \rightarrow R$ care satisface **axiomele**:

- $0 \leq P(A) \leq 1$
- $P(S) = 1$ (sau P(adev) = 1 si P(fals) = 0)
- $P(A \vee B) = P(A) + P(B) - P(A \wedge B)$

$P(A \vee \sim A) = P(A)+P(\sim A) - P(fals) = P(adev)$

$\Rightarrow P(\sim A) = 1 - P(A)$

Definitii TP - cont

Evenimente mutual exclusive

Moneda – cap/pajura – mutual exclusive

Zar – 1, 2, 3 – mutual exclusive

Evenimente exhaustive

Moneda – cap/pajura – mutual exhaustive

Zar – 1, 2, 3, 4, 5, 6 – mutual exhaustive

Definitii TP - cont

A si B mutual exclusive ➔

$P(A \vee B) = P(A) + P(B)$

$P(e_1 \vee e_2 \vee e_3 \vee \ldots e_n) =$

$$P(e_1) + P(e_2) + P(e_3) + \ldots + P(e_n)$$

e(a) – multimea de evenimente atomice mutual exclusive si exhaustive in care apare **a**

$$\boxed{P(a) = \sum_{e_i \in e(a)} P(e_i)}$$

1.3 Regula produsului

Probabilitatea conditionata de producere a evenimentului A ***in conditiile producerii evenimentului B***

- **$P(A|B) = P(A \wedge B) / P(B)$**

$$P(A \wedge B) = P(A|B) * P(B)$$

1.4 Teorema lui Bayes

$P(A|B) = P(A \wedge B) / P(B)$ – regula produsului

$$P(A|B) = P(A \wedge B) / P(B)$$

$$P(B|A) = P(A \wedge B) / P(A)$$

$$\boxed{P(B|A) = P(A|B) * P(B) / P(A)}$$

APosteriori = **Plauzibilitate** x **APriori** / **Evidenta**

Teorema lui Bayes

$P(B|A) = P(A|B) * P(B) / P(A)$

- Daca **B si ~B sunt mutual exclusive si exhaustive**, probabilitatea de producere a lui A in conditiile producerii lui B se poate scrie

$P(A)$ $= P(A \wedge B) + P(A \wedge \sim B) =$ **$P(A|B)*P(B) + P(A|\sim B)*P(\sim B)$**

$P(B|A) =$
$P(A \mid B) * P(B) / [P(A|B)*P(B) + P(A|\sim B)*P(\sim B)]$

Teorema lui Bayes

Generalizarea la mai multe ipoteze

B - h, A - e

P(h|e) = P(e | h) * P(h) / [P(e|h)*P(h) +P(e| ~h)*P(~h)]

Daca h_i mutual exclusive si exhaustive, i=1,k

$$P(h_i|e) = \frac{P(e|h_i)\cdot P(h_i)}{\sum_{j=1}^{k} P(e|h_j)\cdot P(h_j)}, \quad i = 1,k$$

Teorema lui Bayes

Generalizarea la mai multe ipoteze si probe

h_i – evenimente / ipoteze probabile (i=1,k);

$e_1,\ldots,e_n$ – probe (evenimente)

$P(h_i)$

$P(h_i \mid e_1,\ldots,e_n)$

$P(e_1,\ldots,e_n \mid h_i)$

$$P(h_i|e_1,e_2,\ldots,e_n) = \frac{P(e_1,e_2,\ldots,e_n|h_i)\cdot P(h_i)}{\sum_{j=1}^{k} P(e_1,e_2,\ldots,e_n|h_j)\cdot P(h_j)},\ i=1,k$$

Teorema lui Bayes - cont

$$P(h_i|e_1,e_2,...,e_n) = \frac{P(e_1,e_2,...,e_n|h_i)\cdot P(h_i)}{\sum_{j=1}^{k} P(e_1,e_2,...,e_n|h_j)\cdot P(h_j)},\ i=1,k$$

Daca $e_1,\ldots,e_n$ sunt ipoteze independente atunci

$$\boxed{P(e|h_j) = P(e_1,e_2,...,e_n|h_j) = P(e_1|h_j)\cdot P(e_2|h_j)\cdot\ldots\cdot P(e_n|h_j),\ j=1,k}$$

PROSPECTOR – sistem expert pentru consultari privind exploatari miniere si evaluarea resurselor

1.5 Inferente din DP si TB

Distributie de probabilitate **P(Carie, Dur_d)**

	Dur_d	~Dur_d
Carie	0.04	0.06
~Carie	0.01	0.89

$P(Carie) = 0.04 + 0.06 = 0.1$

$P(Carie \vee Dur_d) = 0.04 + 0.01 + 0.06 = 0.11$

$P(Carie|Dur_d) = P(Carie \wedge Dur_d) / P(Dur_d) = 0.04 / 0.05$

Inferente din DP si TB

	Dur_d		~Dur_d	
	Evid	~Evid	Evid	~Evid
Carie	0.108	0.012	0.072	0.008
~Carie	0.016	0.064	0.144	0.576

Distributie de probabilitate **P(Carie, Dur_d, Evid)**

P(Carie) = 0.108 + 0.012 + 0.72 + 0.008 = 0.2
P(Carie ∨ Dur_d) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28

P(Carie, Dur_d, Vreme) – o tabela cu 2 x 2 x 3 = 12 intrari
Distributie de probablitate completa

Inferente din DP si TB

	Dur_d		~Dur_d	
	Evid	~Evid	Evid	~Evid
Carie	0.108	0.012	0.072	0.008
~Carie	0.016	0.064	0.144	0.576

P(Carie | Dur_d) = P(Carie $\wedge$ Dur_d) / P(Dur_d)
P(~Carie | Dur_d) = P(~Carie $\wedge$ Dur_d) / P(Dur_d)
α = 1 / P(Dur_d) – constanta de normalizare a distributiei

P(Carie | Dur_d) = α **P(Carie $\wedge$ Dur_d)** =
α [P(Carie $\wedge$ Dur_d $\wedge$ Evid) + P(Carie $\wedge$ Dur_d $\wedge$ ~Evid)] =
α [<0.108, 0.016> + <0.012, 0.064>] = α <0.12, 0.08> = <0.6, 0.4>

<u>Chiar daca nu cunoastem α</u>, adica P(Dur_d), putem calcula <u>α</u>
α = 1/(0.12+0.08) = 1/0.2

Inferente din DP si TB

Generalizare – procedura generala de inferenta bazata pe DPC
Interogare asupra lui X

X – variabila de interogat (Carie)
E – lista de variabile probe (Dur_d)
e – lista valorilor observate pt aceste variabile E
Y – lista variabilelor neobservate (restul) (Evid)

P(Carie | Dur_d) = **α** [P(Carie ∧ Dur_d ∧ Evid) + P(Carie ∧ Dur_d ∧ ~Evid)]

Insumarea se face peste toate combinatiile de valori ale variab. neobservate Y

$$\mathbf{P}(X \mid e) = \alpha\, \mathbf{P}(X, e) = \alpha\, \Sigma_{Y=y}\, \mathbf{P}(X, e, Y)$$

Inferente din DP si TB

$$\mathbf{P}(X \mid e) = \alpha\, \mathbf{P}(X, e) = \alpha \Sigma_{Y=y}\, \mathbf{P}(X, e, Y)$$

Avand DPC ecuatia poate da raspuns la interogari cu variabile discrete

Complex computational

n var Bool – tabela $O(2^n)$

- timp $O(2^n)$

1.6 Independenta conditionala

- Evenimentele **X1 si X2 sunt independente conditional fiind dat un eveniment Y** daca

  Stiind ca Y apare, aparitia lui X1 nu influenteaza aparitia lui X2 si aparitia lui X2 nu influenteaza aparitia lui X1

altfel spus

X1 si X2 sunt independente conditional fiind dat un eveniment Y daca, pentru orice valoare a lui Y:

- distributia de probabilitate a lui X1 este aceeasi pentru orice valoare ar lua X2
- distributia de probabilitate a lui X2 este aceeasi pentru orice valoare ar lua X1

$$\mathbf{P}(X1,X2|Y) = \mathbf{P}(X1|Y)*\mathbf{P}(X2|Y)$$

Independenta conditionala

$\mathbf{P}(\text{cauza} \mid \text{efect}) = \mathbf{P}(\text{efect} \mid \text{cauza}) * \mathbf{P}(\text{cauza}) / \mathbf{P}(\text{efect})$

$\mathbf{P}(y \mid x_1,..,x_n) = P(x_1,..,x_n \mid y) * \mathbf{P}(y) / \mathbf{P}(x_1,..x_n)$

$\mathbf{P}(y \mid x_1,..,x_n) = \alpha * P(x_1,..,x_n \mid y) * \mathbf{P}(y)$

Daca $x_1,..,x_n$ sunt independente conditional fiind dat y atunci $P(x_1,..x_n \mid y) = \Pi_i P(x_i \mid y)$

$\mathbf{P}(\text{Cauza} \mid \text{Efect}_1, \text{Efect}_2, \ldots) =$

$\alpha\, \mathbf{P}(\text{Cauza}) * \Pi_i\, \mathbf{P}(\text{Efect}_i \mid \text{Cauza})$

1.7 Model Bayesian naiv

$$\mathbf{P}(\text{Cauza}_y \mid \text{Efect}_1, \text{Efect}_2, \ldots) = \alpha\, \mathbf{P}(\text{Cauza}_y) * \Pi_i\, \mathbf{P}(\text{Efect}_i \mid \text{Cauza}_y)$$

Ipoteza de independenta conditionala / naiva

$$\text{Cauza_MAP} = \text{argmax}_y\, \mathbf{P}(\text{Cauza}_y) * \Pi_i\, \mathbf{P}(\text{Efect}_i \mid \text{Cauza}_y)$$

Model Bayesian naiv

Model Bayesian naiv

Vector de caracteristici $x_1,..,x_n$ si o serie de clase C_k

$$\mathbf{P}(C_k \mid x_1, x_2, \ldots) = \alpha\, \mathbf{P}(C_k) * \Pi_i\, \mathbf{P}(x_i|C_k)$$

$$\boxed{C_MAP = \mathrm{argmax}_k\, \mathbf{P}(C_k) * \Pi_i\, \mathbf{P}(x_i|C_k)}$$

Model Bayesian naiv

MAP – Maximum a Posteriori

1.8 Modele grafice probabiliste

- Fiecare nod reprezinta o variabila aleatoare si fiecare legatura reprezinta o relatie probabilistica
  - Retele Bayesiene – modele grafice orientate - permit reprezentare compacta a DP si punerea in evidenta a independentei conditionale
  - Modele Markov – lanturi Markov – stari si tranzitii probabilistice

Figure 2

2 Retele Bayesiene

- Reprezinta dependente intre variabile aleatoare
- Specifica distributiei de probabilitate
- Simplifica calculele
- Au asociata o reprezentare grafica convenabila
- DAG care reprezinta relatiile cauzale intre variabile
- Pe baza structurii retelei se pot realiza diverse tipuri de inferente
- Calcule complexe in general dar se pot simplifica pentru structuri particulare

2.1 Structura retelelor Bayesiene

O RB este un DAG in care:

- Nodurile reprezinta variabilele aleatoare
- Legaturile orientate **X→Y**: X are o influenta directa asupra lui Y, **X=Parinte(Y)**
- Fiecare nod are asociata o tabela de probabilitati conditionate care cuantifica efectul parintilor asupra nodului
  - $\mathbf{P(X_i \mid Parinti(X_i))}$

Structura retelelor Bayesiene - cont

P(H)
0.001

P(C)
0.002

H	C	P(A|H,C)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(M|A)
T	0.9
F	0.05

A	P(D|A)
T	0.7
F	0.01

Tabela de probabilitati conditionate

H	C	P(A | H, C) T	P(A | H, C) F
T	T	0.95	0.05
T	F	0.94	0.06
F	T	0.29	0.71
F	F	0.001	0.999

Structura retelelor Bayesiene

In general X (Cauza) $\rightarrow$ Y (Efect)

- Stabilesc topologia
- Specifica distributia de probabilitati conditionate
- Combinarea topologiei si distributia de probabilitati conditionate este suficienta pentru a specifica (implicit) intreaga DPC
- DPC poate raspunde la interogari
- Si RB la fel, mai eficient

2.2 Semantica retelelor Bayesiene

- Reprezentare a distributiei de probabilitate
- Specificare a independentei conditionale – constructia retelei

- Fiecare valoare din distributia de probabilitate poate fi calculata ca:

$$\mathbf{P(X_1=x_1 \wedge \ldots X_n=x_n) = P(x_1,\ldots, x_n) = \Pi_{i=1,n} P(x_i \mid parinti(x_i))}$$

unde *parinti*(x_i) reprezinat valorile specifice ale variabilelor *Parinti*(X_i)

2.3 Construirea retelei

Cum sa construim o retea a.i. RB/DPC sa fie o buna reprezentare?

Ecuatia $\mathbf{P(x_1,\ldots, x_n) = \Pi_{i=1,n} P(x_i \mid parinti(x_i))}$ implica anumite relatii de independenta conditionala care pot ghida construirea retelei

$\mathbf{P(X_1=x_1 \wedge \ldots X_n=x_n) = P(x_1,\ldots, x_n) =}$

$\mathbf{P(x_n \mid x_{n-1},\ldots, x_1) * P(x_{n-1},\ldots, x_1) = \ldots =}$

$\mathbf{P(x_n \mid x_{n-1},\ldots, x_1) * P(x_{n-1} \mid x_{n-2},\ldots, x_1)* \ldots P(x_2|x_1) * P(x_1) =}$

$\mathbf{\Pi_{i=1,n} P(x_i \mid x_{i-1},\ldots, x_1)}$ – valabila in general

- DPC daca, pt fiecare variabila X_i din RB

$\mathbf{P(X_i \mid X_{i-1},\ldots, X_1) = P(x_i \mid Parinti(X_i))}$ cu conditia ca

$\mathbf{Parinti(X_i) \subseteq \{ X_{i-1},\ldots, X_1\}}$

Construirea retelei

Pt fiecare variabila X_i din RB

$\mathbf{P(X_i \mid X_{i-1},\ldots, X_1) = P(x_i \mid Parinti(X_i))}$ cu conditia ca

$$\mathbf{Parinti(X_i) \subseteq \{ X_{i-1},\ldots, X_1\}}$$

- O RB este o reprezentare corecta a domeniului cu conditia ca fiecare nod sa fie independent conditional de nondescendenti, fiind dati parintii lui

Construirea retelei

- Conditia poate fi satisfactuta prin etichetarea nodurilor intr-o ordine consitenta cu DAG

- Intuitiv, parintii unui nod X_i trebuie sa fie toate acele noduri $X_{i-1},\ldots, X_1$ care influenteaza direct X_i.

Construirea retelei - cont

- Alege o multime de variabile aleatoare relevante care descriu problema
- Alege o ordonare a acestor variabile
- **cat timp** mai sunt variabile **repeta**

  (a) alege o variabila X_i si adauga un nod corespunzator lui X_i

  (b) atribuie Parinti(X_i) $\leftarrow$ un set minim de noduri deja existente in retea a.i. proprietatea de independenta conditionala este satisfacuta

  (c) defineste tabela de probabilitati conditionate pentru X_i

Deoarece fiecare nod este legat numai la noduri anterioare $\rightarrow$ DAG

2.4 Inferente probabilistice

$$P(A \wedge V \wedge B) = P(A) * P(V|A) * P(B|V)$$

$$P(A \wedge V \wedge B) = P(V) * P(A|V) * P(B|V)$$

$$P(A \wedge V \wedge B) = P(A) * P(B) * P(V|A,B)$$

Inferente probabilistice

P(H)
0.001

P(C)
0.002

H	C	P(A|H,C)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(M|A)
T	0.9
F	0.05

A	P(D|A)
T	0.7
F	0.01

$P(M \wedge D \wedge A \wedge \sim H \wedge \sim C) =$

$P(M|A) * P(D|A) * P(A|\sim H \wedge \sim C) * P(\sim H) \wedge P(\sim C) =$

$0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00062$

Inferente probabilistice

P(H)
0.001

P(C)
0.002

H	C	P(A|H,C)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(M|A)
T	0.9
F	0.05

A	P(D|A)
T	0.7
F	0.01

P(A|H) = P(A|H,C) *P(C|H) + P(A| H,~C)*P(~C|H)

= P(A|H,C) *P(C) + P(A| H,~C)*P(~C)

= 0.95 * 0.002 + 0.94 * 0.998 = 0.94002

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 5 si 6

Reprezentarea cunostintelor incerte

- Teoria probabilitatilor
- Retele Bayesiene
- Inferente exacte si aproximative in retele Bayesiene

Inferente probabilistice

P(H)
0.001

P(C)
0.002

H	C	P(A|H,C)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(M|A)
T	0.9
F	0.05

A	P(D|A)
T	0.7
F	0.01

$P(M \wedge D \wedge A \wedge \sim H \wedge \sim C) =$

$P(M|A) * P(D|A) * P(A|\sim H \wedge \sim C) * P(\sim H) \wedge P(\sim C) =$

$0.9 * 0.7 * 0.001 * 0.999 * 0.998 = 0.00062$

Inferente probabilistice

P(H)
0.001

P(C)
0.002

H	C	P(A|H,C)
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

A	P(M|A)
T	0.9
F	0.05

A	P(D|A)
T	0.7
F	0.01

P(A|H) = P(A|H,C) *P(C|H) + P(A| H,~C)*P(~C|H)

= P(A|H,C) *P(C) + P(A| H,~C)*P(~C)

= 0.95 * 0.002 + 0.94 * 0.998 = 0.94002

2.5 Inferență prin enumerare

X – variabila de interogare (Carie)
E – lista de variabile probe (Dur_d)
e – lista valorilor observate pt aceste variabile E
Y – lista variabilelor neobservate (restul) (Evid)

$$\mathbf{P}(X \mid e) = \alpha\, \mathbf{P}(X, e) = \alpha \Sigma_Y\, \mathbf{P}(X, e, Y)$$

$$\mathbf{P}(H|M,D) = \alpha\, P(H,M,D) =$$

$$\alpha\, \Sigma_C\, \Sigma_A\, P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$

n var bool -> $O(2^n)$

Inferență prin enumerare

$$\mathbf{P(H|M,D)} = \alpha \, \Sigma_C \, \Sigma_A \, P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$

$$P(H|M,D) = \alpha \, P(H) \, \Sigma_C \, P(C) \, \Sigma_A \, P(A|H,C)P(M|A)P(D|A)$$

$$= \alpha \, \langle 0.00059224, \; 0.0014919 \rangle =_{aprox} \langle 0.284, \; 0.716 \rangle$$

Complexitate spatiu – O(n)

Complexitate timp $O(2^n)$

Inferență prin enumerare - algoritm

algoritm **Enumerare(X,e,rb)** **intoarce** distributie X

X – var de interogare

e – valori observate pt E

rb – Retea Bayesiana cu var $\{X\} \cup E \cup Y$

1. $Q(X) \leftarrow$ o distributie X, initial vida
2. **pentru** fiecare valoare x_i a lui X **repeta**

   $Q(x_i) \leftarrow$ **EnumToate**(**rb.Vars**, $\mathbf{e_{xi}}$)

   unde $\mathbf{e_{xi}}$ este **e** extins cu $X=x_i$
3. **intoarce** Normalizare(Q(x))

sfarsit

Inferență prin enumerare - algoritm

algoritm **EnumToate(Vars,e)** **intoarce** un numar real

1. daca Vars = [] atunci intoarce 1.0
2. Y ← first(Vars)
3. **daca** Y are valoare ***y*** in ***e***

   atunci intoarce P(y|parinti(Y)) * **EnumToate**(Rest(Vars),***e***)

   altfel intoarce

   Σ_y P(y|parinti(Y)) * **EnumToate**(Rest(Vars),$\boldsymbol{e_y}$)

   unde $\boldsymbol{e_y}$ este ***e*** extins cu Y=y

sfarsit

P(H|M,D)= $\alpha\ \Sigma_C\ \Sigma_A$ P(H)P(C)P(A|H,C)P(M|A)P(D|A)

$$\mathbf{P}(X \mid e) = \alpha\, \mathbf{P}(X, e) = \alpha \Sigma_Y\, \mathbf{P}(X, e, Y)$$

$$\mathbf{P}(\mathbf{H}|\mathbf{M},\mathbf{D}) = \alpha\, \Sigma_C\, \Sigma_A\, P(H)P(C)P(A|H,C)P(M|A)P(D|A)$$

Eliminarea variabilelor (Variable elimination)

Pt a elimina calculele duplicate – se calculeaza o data si se salveaza rezultatul

Idee – se calculeaza ecuatia de la dreapta la stanga

$P(H|M,D) = \alpha \Sigma_C \Sigma_A P(H) \; P(C) \; P(A|H,C) \; P(M|A) \; P(D|A)$

f1(H) f2(C) f3(A,H,C) f4(A) f5(A)

f5(A) = [P(D|A) P(D|~A)] = [.07 .001]

f4(A) = [P(M|A) P(M|~A)] = [.09 .05]

f3(A,H,C) va fi o matrice de 2 x 2 x 2

[P(A|H,C) P(~A|H,C) P(A|H, ~C) P(~A|H, ~C) … P(~A|~H,~C)]

Eliminarea variabilelor

$P(H|M,D) = \alpha \, \Sigma_C \, \Sigma_A \, P(H) \; P(C) \; \underline{P(A|H,C) \, P(M|A) \, P(D|A)}$

f1(H) f2(C) f3(A,H,C) f4(A) f5(A)

Insumam peste A in f3, f4, f5 – rezulta un factor f6(H,C)

f6(H,C) = Σ_A **f3**(A,H,C) x **f4**(A) x **f5**(A) =

f3(a,H,C) x **f4**(a) x **f5**(a) + **f3**(~a,H,C) x **f4**(~a) x **f5**(~a)

Insumam peste C – rezulta un factor f7(H)

P(H|M,D)= α **f1**(H) x $\underline{\Sigma_C \, \textbf{f2}(C) \text{ x } \textbf{f6}(H,C)}$

f7(H) = Σ_C **f2**(C) x **f6**(H,C) = **f2**(c) x **f6**(H,c) + **f2**(~c) x **f6**(H,~c)

P(H|M,D)= α **f1**(H) x **f7**(H)

Avem nevoie de 2 operatii

- **Pointwise product** x
- **Insumarea unei variabile dintr-un produs de factori**

Implementarea operațiilor

Pointwise product

$(\mathbf{f_1} \text{ x } \mathbf{f_2})(X_1,\ldots,X_i,Y_1,\ldots,Y_j,Z_1,\ldots,Z_k) =$

$$\mathbf{f_1}(X_1,\ldots,X_i,Y_1,\ldots,Y_j) \text{ x } \mathbf{f_2}(Y_1,\ldots,Y_j,Z_1,\ldots,Z_k)$$

Insumarea unei variabile dintr-un produs de factori

- Insumarea peste **X1**, rezultatul este un factor peste $X_2,\ldots,X_i$

$$(\Sigma_{X1}\mathbf{f})(X_2,\ldots,X_i) = \mathbf{f}(X_1=v_1,\ldots,X_i)+\ldots+ \mathbf{f}(X_1=v_k,\ldots,X_i)$$

Implementarea operațiilor - exemplu

Pointwise product

f1(A,B) x f2(B,C) = f3(A,B,C) are 2^{1+1+1} = 8 intrari

A	B	f1(A,B)	B	C	f2(B,C)	A	B	C	f3(A,B,C)
a	a	.3	a	a	.2	a	a	a	.3 x .2 = .06
a	f	.7	a	f	.8	a	a	f	.3 x .8 = .24
f	a	.9	f	a	.6	a	f	a	.7 x .6 = .42
f	f	.1	f	f	.4	a	f	f	.7 x .4 = .28
						f	a	a	.9 x .2 = .18
						f	a	f	.9 x .8 = .72
						f	f	a	.1 x .6 = .06
						f	f	f	.1 x .4 = .04

Implementarea operațiilor - exemplu

Insumarea unei variabile dintr-un produs de factori

- Insumarea peste **A** din **f3(A,B,C)**

f(B,C) = Σ_A f3(A,B,C) = f3(a,B,C) + f3(~a,B,C) =

$$\begin{vmatrix} .06 & .24 \\ .42 & .28 \end{vmatrix} + \begin{vmatrix} .18 & .72 \\ .06 & .04 \end{vmatrix} = \begin{vmatrix} .24 & .96 \\ .48 & .32 \end{vmatrix}$$

2.6 Tipuri de inferență

Inferente de diagnosticare (efect → cauza)

P(Hot | TelMihai)

Inferente cauzale (cauza → efect)

P(TelMihai |Hot), P(TelDana | Hot)

Tipuri de inferență

Inferente intercauzale (intre cauza si efecte comune)

$P(Hot \mid Alarma \wedge TelDana)$

Inferente mixte

$P(Alarma \mid TelMihai \wedge \sim Cutremur) \rightarrow$ diag + cauzal

$P(Hot \mid TelMihai \wedge \sim Cutremur) \rightarrow$ diag + intercauzal

2.7 Independenta in RB

- Evenimentele **X si Y sunt independente conditional fiind dat un eveniment Z** daca

  Stiind ca Z apare, aparitia lui X nu influenteaza aparitia lui Y si aparitia lui Y nu influenteaza aparitia lui X

- **Structura de graf RB codifica anumite relatii de independenta: fiecare nod este independent conditional de nondescendenti, fiind dati parintii lui**

Independenta in RB

- Mai sunt si alte relatii de independenta?
- Doua sau mai multe relatii de independenta conditionala pot conduce la o noua relatie folosind un mecanism bazat pe axiome ale grafurilor
- **D-separabilitatea** este un concept care surprinde astfel de relatii de independenta conditionala derivate
- Z **d-separa** pe X si Y intr-un DAG daca X si Y sunt independente conditional fiind dat Z dpv al axiomelor grafului.
- **(X $\perp$ Y | Z) – X este independent conditional de Y fiind dat Z**

D-separabilitate

D-separabilitate

- **Cand nu este indeplinita ($X \perp Y \mid Z$) ?**

Conexiuni directe $X \rightarrow Y$

- Se pot influenta indiferent de Z

Conexiuni indirecte: X si Y nu sunt direct conectate dar exista un lant intre ele (trail)

D-separabilitate

- Fie cazul a 3 noduri

Caz a: Efect cauzal

Lantul cauzal **X → Z → Y**

- X nu influenteaza Y via Z daca <u>Z observat</u>

Caz b: Efect evidenta

- **Y → Z → X**
- Identic cu cazul **a:**
- X nu influenteaza Y via Z daca <u>Z observat</u>
- **daca (X ⊥ Y | Z) nu este indeplinit atunci nici (Y ⊥ X | Z) nu este indeplinit**

D-separabilitate

Caz c: Cauza comuna

- **X ← Z → Y**
- X nu influenteaza pe Y via Z daca <u>Z observat</u>

Case d: Efect comun

- **X → Z ← Y**
- Influenta nu poate merge de-a lungul lantului

X → Z ← Y daca <u>Z nu este observat</u>

Deci X nu influenteaza pe Y via Z daca <u>Z NU este observat</u>

D-separabilitate

Daca influenta poate curge de la X la Y via Z spunem ca lantul $X \leftrightarrow Z \leftrightarrow Y$ este <u>activ</u>

- Efect cauzal $X \rightarrow Z \rightarrow Y$ – activ $\leftrightarrow$ Z nu este observat
- Efect evidenta $Y \rightarrow Z \rightarrow X$ – activ $\leftrightarrow$ Z nu este observat
- Cauza comuna $X \leftarrow Z \rightarrow Y$ – activ $\leftrightarrow$ Z nu este observat
- Efect comun $X \rightarrow Z \leftarrow Y$ - activ $\leftrightarrow$ fie Z fie unul din descendentii lui Z este observat
- **Influenta probabilistica** = **flux in graf** – cand influenta din X poate curge prin Z catre Y a.i. sa afecteze probabilitatea lui Y,
- Deci X nu este conditional independent de Y fiind dat Z

d^0	d^1
0.6	0.4

i^0	i^1
0.7	0.3

	g^1	g^2	g^3
i^0, d^0	0.3	0.4	0.3
i^0, d^1	0.05	0.25	0.7
i^0, d^0	0.9	0.08	0.02
i^0, d^1	0.5	0.3	0.2

	s^0	s^1
i^0	0.95	0.05
i^1	0.2	0.8

	l^0	l^1
g^1	0.1	0.9
g^2	0.4	0.6
g^2	0.99	0.01

2.8 Inferențe de aproximare în RB

- Inferente exacte in RB:
  - Inferenta prin enumerare
  - Eliminarea variabilelor (eficientizare)

Asa cum s-a aratat anterior, exprimam probabilitatea conditionata ca probabilitate neconditionata, apoi insumam peste variabilele neobservate

$$\mathbf{P}(X \mid e) = \alpha\, \mathbf{P}(X, e) = \alpha \Sigma_Y\, \mathbf{P}(X, e, Y)$$

Inferente de aproximare in RB

Inferente exacte in RB foarte costisitoare

- Complexitate spatiu – O(n)
- Complexitate timp $O(2^n)$ – pt *n* variabile booleene

Pentru retele complexe aceste metode devin de multe ori nepractice

Asigurarea auto

Inferențe de aproximare

Inferente de aproximare

- Algoritmi de esantionare aleatoare, cunoscuti si ca algoritmi Monte Carlo
- Ofera raspunsuri aproximative care depind de numarul de esantioane care se genereaza
- Ne intereseaza esantionare pentru calculul probabilitatilor conditionate

Esantionare directa

Esantionare cu lanturi Markov

a) Esantionare directa

- Se genereaza esantioane dintr-o distributie de probabilitate cunoscuta
- Cel mai simplu proces de esantionare intr-o RB genereaza esantioane/evenimente din retea fara nici o proba
- Ideea este de a esantiona fiecare variabila pe rand in ordine topologica
- Distributia de probabilitate din care se obtin valorile unei variabile din esantion este conditionata de valorile care au fost deja atribuite parintilor

RB exemplu

P(N)

a	f
0.5	0.5

N	P(A|N)	
a	0.1	0.9
f	0.5	0.5

N	P(P|N)	
a	0.8	0.2
f	0.2	0.8

A	P	P(I | A, P) a	P(I | A, P) f
a	a	0.99	0.01
a	f	0.9	0.1
f	a	0.9	0. 1
f	f	0.1	0.9

Generarea esantionului

Presupun ordinea [*Nori*, *Aspersor*, *Ploaie*, *IarbaUda*]

1. Esantion **P(Nori)** = <0.5 0.5> **a**
2. Esantion din **P(Aspersor|Nori=a)** = <0.1 0.9> **f**
3. Esantion **P(Ploaie|Nori=a)** = <0.8 0.2> **a**
4. Esantion din **P(IarbaUda|Aspersor=f, Ploaie=a)** = <0.9 0.1> **a**

Esantionarea este **[a, f, a, a]**

RB exemplu

P(N)	
a	f
0.5	0.5

N	P(A|N)	
a	0.1	0.9
f	0.5	0.5

N	P(P|N)	
a	0.8	0.2
f	0.2	0.8

A	P	P(I | A, P) a	P(I | A, P) f
a	a	0.99	0.01
a	f	0.9	0.1
f	a	0.9	0. 1
f	f	0.1	0.9

Algoritm de esantionare

algoritm **Esantionare(rb)** **intoarce** esantion

intrari: rb specificand d.p. $P(X_1,\ldots,X_N)$

x ← esantion cu n elemente

pentru fiecare variabila X_i din $X_1,\ldots,X_N$ **repeta**

    x[i] ← un esantion aleator din $P(X_i|\text{parinti}(X_i))$

intoarce x

sfarsit

Algoritm de esantionare

- Intr-un algoritm de esantionare, raspunsurile sunt calculate prin numararea numarului de esantioane generate
- Sa presupunem ca avem N esantionari

  $N_{PS}(x_1,\ldots,x_n)$ – de cate ori a aparut $x_1,\ldots,x_n$ in multimea de esantioane
- Se presupune ca acest numar va converge chiar la probabilitate

  $$\lim_{N\to\infty} (N_{PS}(x_1,\ldots,x_n)/N) = P(x_1,\ldots,x_n)$$
- Fie S_{PS} probabilitatea ca un eveniment(esantion) specific sa fie generat de algoritmul de esantionare

  $$S_{PS}(x_1,\ldots,x_n) = \Pi_{i=1,n} P(x_i|\text{parinti}(X_i)) = P(x_1,\ldots,x_n)$$

deoarece fiecare pas de esantionare depinde de valorile parintilor

Algoritm de esantionare

$S_{PS}(x_1,\ldots,x_n) = \Pi_{i=1,n}P(x_i|\text{parinti}(X_i)) = P(x_1,\ldots,x_n)$

$\mathbf{\lim_{N\to\infty} (N_{PS}(x_1,\ldots,x_n)/N) = S_{PS}(x_1,\ldots,x_n) = P(x_1,\ldots,x_n)}$

De exemplu sa consideram esantionul

[a, f, a, a] generat anterior

Probabilitatea de esantionare a acestui eveniment

$$\mathbf{S_{PS}(a,f,a,a) = 0.5 \times 0.9 \times 0.8 \times 0.9 = 0.324}$$

Pentru N foarte mare, putem presupune ca 32,4% din esantioane vor fi egale cu acesta

b) Eliminarea esantioanelor nepotrivite

Eliminarea esantioanelor (***Rejection sampling***)

- Se refera la situatia in care avem de estimat o probabilitate conditionata P(X|**e**)
- Se genereaza esantioanele din distributia de probabilitate specificata de retea
- Apoi se elimina acele esantioane care nu se potrivesc cu probele **e**
- In final estimarea lui P(X=x|**e**) este obtinuta prin numararea numarului de aparitii X=x in esantioanele ramase

Eliminarea esantioanelor nepotrivite

- Dorim sa estimam

  P(Ploaie|Aspersor = a)

- Generam, de ex, 100 esantioane
- Presupunem ca din cele 100 esantioane 73 au Aspersor = f, le eliminam
- Raman 27 cu Aspersor = a
- Din acestea 8 au Ploaie=a si 19 au Ploaie=f
- **P(Ploaie|Aspersor = a) = Normalizare(<8, 19>) =**

  < 0.296, 0.704>

Rapunsul corect este <0.3,0.7>

- Pe masura ce colectam mai multe esantioane, raspunsul va converge catre valoarea reala

Algoritm de esantionare cu eliminare

algoritm **EsantionareEliminare(X,e,rb,N)**

intoarce o estimare P(X|**e**)

intrari: rb specificand d.p. $P(X_1,\ldots,X_N)$

X – variabila de interogare

e – valorile observate pentru variabilele E

N – numarul total de esantioane generate

variabile locale VN – vector care numara aparitiile valorilor lui X

pentru j ← 1,N **repeta**

y ← Esantionare (rb)

daca *y* este consistent cu **e atunci**

VN[x] ← VN[x]+1 unde x este valoarea lui X in *y*

intoarce Normalizare(VN)

sfarsit

Eliminarea esantioanelor nepotrivite

- Principala problema a acestei abordari este faptul ca elimina prea multe esantioane

- Esantioane pt P(Cutremur|Alarma=a)

  ~~Cutremur =f, Alarma = f~~

  ~~Cutremur =f, Alarma = f~~

  ~~Cutremur =f, Alarma = f~~

  Cutremur =a, Alarma = a

- Procentul de esantioane consistent cu probele scade exponential pe masura ce creste numarul de probe, deci problematic pentru probleme complexe

c) Esantionarea ponderata

Esantionare ponderata ***(Likelihood weighting)***

- Elimina ineficienta potentiala a eliminarii esantioanelor prin generarea numai a acelor evenimente (esantioane) care sunt consistente cu probele **e**
- Se fixeaza valorile variabilelor probe **E** si se esantioneaza numai pentru celelalte variabile
- Acest lucru garanteaza faptul ca fiecare esantion generat este consistent cu probele; dar nu toate evenimentele sunt la fel de importante

Esantionarea ponderata

- Inainte de a le numara, fiecare eveniment este ponderat cu o **plauzibilitate** (likelihood) pe care esantionul o acorda probei

- **Plauzibilitatea (likelihood)** este masurata ca **produsul probabilitatilor conditionate pentru fiecare variabila proba, fiind dati parintii ei**

Esantionarea ponderata - Exemplu

P(Ploaie|Nori=a, IarbaUda=a)

si ordonarea *[Nori, Aspersor, Ploaie, IarbaUda]*

Initial w=1.0

Se genereaza un esantion

1. *Nori* este o variabila proba cu valoarea **a**

   In consecinta **w ← w x P(Nori=a) = 0.5**

2. *Aspersor* nu este o variabila proba deci esantionam

   P(Aspersor|Nori=a) = (0.1, 0.9) - presupun **f**

3. Similar P(Ploaie|Nori=a) = (0.8, 0.2) - presupun **a**

A	P	P(I | A, P) a	f
a	a	0.99	0.01
a	f	0.9	0.1
f	a	0.9	0. 1
f	f	0.1	0.9

Esantionarea ponderata - Exemplu

P(Ploaie|Nori=a, IarbaUda=a)

si ordonarea *[Nori, Aspersor, Ploaie, IarbaUda]*

4. *IarbaUda* este o variabila proba cu valoarea **a**

w ← w x P(IarbaUda=a | Aspersor=f, Ploaie=a) =

0.9 x 0.5 = 0.45

Esantionarea ponderata intoarce un esantion **[a,f,a,a] cu ponderea 0.45** in conditiile Ploaie=a

P(N)	
a	f
0.5	0.5

N	P(A|N)	
a	0.1	0.9
f	0.5	0.5

N	P(P|N)	
a	0.8	0.2
f	0.2	0.8

A	P	P(I | A, P) a	P(I | A, P) f
a	a	0.99	0.01
a	f	0.9	0.1
f	a	0.9	0. 1
f	f	0.1	0.9

Algoritm de esantionare ponderata

algoritm **EsantionarePonderata(X,e,rb,N)**

intoarce o estimare P(X|e)

intrari: rb specificand d.p. $P(X_1,\ldots,X_N)$

X – variabila de interogare

e – valorile observate pentru variabilele E

N – numarul total de esantioane generte

variabile locale W – vector care numara aparitiile valorilor lui X, ponderate

pentru j ← 1,N **repeta**

x,*w* ← Esantion-Ponderat(rb,e)

W[x] ← W[x]+*w* unde x este valoarea lui X in ***x***

intoarce Normalizare(W)

sfarsit

Algoritm de esantionare ponderata

algoritm **Esantion-Ponderat(rb,e)**

intoarce un esantion si o pondere

$w \leftarrow 1$

x $\leftarrow$ un esantion cu n elemente initializate din e

pentru fiecare variabila X_i din $X_1,..,X_n$ **repeta**

daca X_i este o variabila proba cu valoarea x_i in e

atunci $w \leftarrow w * P(X_i=x_i \mid \text{parinti}(X_i))$

altfel **x**[i] $\leftarrow$ un esantion aleator din $P(X_i \mid \text{parinti}(X_i))$

intoarce **x**, w

sfarsit

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs 7

- Elemente de învățare automată
- Arbori de decizie

1. Definiții

- Învățarea este procesul prin care un sistem isi imbunatateste performantele (Herbert Simon)
- Învățarea este achizitia cunostintelor explicite
- Învățarea este achizitia deprinderilor de rezolvare a problemelor
- Învățarea este formare teoriilor, formarea ipotezelor si inferența inductivă
- ML – observa cantitati mari de date și caută șabloane petru predicție

Sistem de invatare
Element de invatare
Feed-back
Date
Rezultate invatare
Mediu
Rezolvare probleme
Profesor
Rezultate
Evaluare performante

Invatare automata

De ce sa invete?

- Taskuri definite prin exemple
- Relatii / corelatii in cantitati mari de date
- Mediu in schimbare
- Date diferite, cu zgomot
- Cantitate de cunostinte prea mare pentru a fi reprezentate explicit

Denumiri utilizate

- Instanță
- Concept
- Concept vizat (target concept)
- Clasă de ipoteze
- Mulțimea de învățare (Training set)
- Mulțimea de test (Test set)

Denumiri utilizate

- T – vector de intrare, vector sablon, vector de caracteristici, esantioane, exemple, instanțe
- x_i- caracteristici, atribute, variabile de intrare, componente
- x_i – pot fi valori reale, valori numerice intregi, valori simbolice, valori booleene
- $f(\mathbf{X_i})$
  - valori reale: h – functie de esantionare
  - valori simbolice: h – clasificator
  - boolene: 1 – instanta pozitiva, 0 – instanta negativa

2. Tipuri de invatare

- **Învățare supervizată** – determinarea ipotezei de invatare pe baza unor date etichetate
  - Simbolica sau non-simbolica
- **Învățare nesupervizată -** determinarea ipotezei de invatare / a unei structuri pe baza unor date neetichetate
  - Simbolica sau non-simbolica
- ***Învățarea prin recompensa***

Model simplu

$T = \{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}\}$ – **multime de invatare**

$$\mathbf{X_i} = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix} \longrightarrow \boxed{h} \longrightarrow h(\mathbf{X_i})$$

$h \in \mathbf{H}$

$f(\mathbf{X}) = ?$

- **Invatare supervizata** – se cunosc $f(\mathbf{X_1}), \ldots, f(\mathbf{X_m})$

Gasim **h** a.i. $h(\mathbf{X_i}) = f(\mathbf{X_i})$, $i=1,m \rightarrow h(\mathbf{X_i}) = f(\mathbf{X_i})$, $\forall i$

Clasificare – f - valori discrete – grupeaza exemple

Regresie – f – valori continue, estimeaza sau prezice valori

Metode de invatare supervizata

Clasificare

- Arbori de decizie
- Clasificare probabilistica (Naïve Bayes)
- K-Nearest neighbours (k-NN pt clasificare)
- SVM
- Retele neurale
- Invățare mulțime (ensemble learning): păduri aleatoare, ADA boost

Metode de invatare supervizata

Regresie

- Regresie liniara
- SVR
- Arbori de regresie
- K-Nearest neighbours (k-NN pt regresie)

Clasificare vs regresie

Clasificare

- Caine / Pisica

- Dan 10.000 lei credit de 30.000 lei
- George 3.000 lei credit de 10.000 lei
- Vlad 15.000 lei credit de 50.000 lei
- Maria 5.000 lei cat credit?

Regresie

Clasificare vs regresie

- Reprezentam ipotezele de invatare sub o forma care sa permita ca iesirea algoritmului sa fie o eticheta (clasa): arbori, retele neurale, etc.

Clasificare

- Reprezentam ipoteza de invatare ca o functie liniara

$h(x)=a_0+a_1x$

- Cat este valoarea y

pt punctul rosu?

Regresie

Model simplu

$T = \{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}\}$ – **multime de invatare**

$$\mathbf{X_i} = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix} \longrightarrow \boxed{h} \longrightarrow h(\mathbf{X_i})$$

$h \in \mathbf{H}$

$f(\mathbf{X}) = ?$

- **Invatare ne-supervizata** – **NU** se cunosc $f(\mathbf{X_1}), \ldots, f(\mathbf{X_m})$

Imparte T in submultimi – clase

Se poate vedea tot ca invatarea unei functii – val f = numele submultimii careia ii apartine $\mathbf{X_i}$

- **Invatare prin recompensa** – Se cunosc recompensele pentru valorile $h(\mathbf{X_1}), \ldots, h(\mathbf{X_m})$, Nu se cunosc valorile lui f

Metode de invatare nesupervizata

- Clustering
  - k-means clustering (grupare), k-means ierarhic (grupare ierarhica)
- Retele neurale
  - Retele neurale cu auto-organizare
  - Autoencoders
  - Deep belief networks

Metode de invatare nesupervizata

- Expectation maximization (EM)
  - HMM – algoritmul Baum-Welch
  - Retele Bayesiene – estimarea parametrilor in cazul datelor incomplete
  - Soft clustering – clasele se pot suprapune
- Principal Component Analysis (PCA)

Caracteristici importante

- **Regimuri de invatare:**
  - Batch
  - Incremental
- **Zgomot:**
  - zgomot intrari (de ex valorile atributelor)
  - zgomot iesiri (alterare iesiri)
- **Ipoteza favorita a inductiei** (Inductive bias)
  - O multime de presupuneri (explicite sau implicite) pe care algoritmul de invatare se bazeaza pentru a obtine un model (a generaliza) din setul de invatare

Inductive bias

- Spatiu de ipoteze restrictionat (Restricted Bias)
  - Functii liniare sau polinomiale de ordin mic
- Ipoteza preferata (Preference Bias)
  - Occam’s razor
  - Nerest neighbours
  - Independenta conditionala maxima
  - Margine maxima
  - CNN, Deep RL agents

Occam's Razor

Principiul lamei lui Occam (*lex parsimoniae*)

- prefera explicatiile simple celor complexe
- selecteaza ipoteza care implica cele mai putine presupuneri, intre ipoteze similare/egal probabile
- Wiliam of Occam, 1285 – 1347
- filozof englez

"non sunt multiplicanda entia praeter necessitatem"

Conditii pentru o invatare "corecta"

Problema: identifica personaje de film "bune" sau "rele" dupa modul in care arata

Conditii pentru o invatare "corecta"

Atribute / Instante	Sex	Masca	Pelerina	Cravata	Urechi	Lupta	**Clasa**
	Set de invatare						
Batman	Masc	Da	Da	Nu	Da	Nu	**Bun**
Robin	Masc	Da	Da	Nu	Nu	Nu	**Bun**
Alfred	Masc	Nu	Nu	Da	Nu	Nu	**Bun**
Pinguin	Masc	Nu	Nu	Da	Nu	Da	**Rau**
Catwoman	Fem	Da	Nu	Nu	Da	Nu	**Rau**
Joker	Masc	Nu	Nu	Nu	Nu	Nu	**Rau**
	Date de test						
Batgirl	Fem	Da	Da	Nu	Da	Nu	**??**
Fred	Masc	Da	Nu	Nu	Nu	Nu	**??**

Conditii pentru o invatare "corecta"

Clasifica corect datele?

Conditii pentru o invatare "corecta"

Clasifica corect datele?

Conditii pentru o invatare "corecta"

Clasifica corect datele?

Aleg prima varianta (cf. lamei lui Occam)

Conditii pentru o invatare "corecta"

- Clasificatoarele trebuie sa fie suficient de "expresive" pentru a fi in concordanta cu setul de invatare
- Dar clasificatoarele care au o complexitate prea mare pot duce la fenomenul de "overfit" (overfitting)

Caracteristici importante

- **Overfitting** = modelul include zgomot sau sabloane de date nerelevante
- Nu mai poate generaliza
- Validare încrucișată
- Regularizare

Caracteristici importante

- **Underfitting** = modelul nu se potriveste nici datelor de învățare și nici nu poate generaliza la date noi

Caracteristici importante

- **Regularizare**
- Evită overfitting
- Se poate aplica aproape oricărui model ML
- Simplifică modelele cu complexitate prea mare prin adăugarea de termeni de penalizare in funcția obiectiv

Caracteristici importante

Cross-validation

- Antrenarea se face pe subseturi de date de învățare și evaluare pe restul
- Metode:
  - Hold out
  - K-fold cross validation
  - Leave-P-out

3. Metrici de performanță

Regresie

- **Mean Absolute Error** (MAE) = media diferenței absolute între valori corecte și valori prezise
- **Root Mean Square Error** (RMSE) = rădăcina pătrată a mediei diferențelor pătratelor între valori corecte și valori prezise

Metrici de performanță pt clasificare

Clasificare

- **Confusion matrix** – corectitudinea și acuratețea modelului

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Metrici de performanță pt clasificare

- **Accuracy**– utilă dacă clasele sunt distribuite uniform

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

Metrici de performanță pt clasificare

- **Precision (hit)**

$$\text{Precision} = \frac{TP}{TP + FP}$$

Precision

Metrici de performanță pt clasificare

- **Recall sau Sensitivity (misses)**

$$\text{Recall} = \frac{TP}{TP + FN}$$

Recall or Sensitivity

4 Arbori de decizie

- ID3 – in jur de 1960s
- C4.5 (Quinlan):
- Permite atribute numerice
- Trateaza cazurile valorilor lipsa
- Trateaza cazul valorilor cu zgomot
- C4.5 – unul din cei mai cunoscuti algoritmi de invatare
- Ultima versiune de cercetare: C4.8, implementata in Weka
- Versiunea comerciala: C5.0

Invatarea inductiva prin AD

- Vede invatarea ca achizitia cunostintelor structurate
- Reprezentarea cunostintelor = **arbori de decizie** (AD)
- Problema de invatare = **clasificare**
- **Invatare supervizata**
- Strategie = invatare batch (ne-incrementala)
- AD se construieste pornind de la radacina spre frunze = ***Top Down Induction of Decision Tree***

ID3 (Quinlan)

- Univers de obiecte **U** descrise in termenii unei colectii de atribute **{A}**
- Fiecare **atribut** masoara o caracteristica importanta a unui obiect **o∈U**
- Domeniul de valori atribute D_A= discret, simbolic (ulterior extins)
- Fiecare obiect apartine unui **clase** dintr-o multime de clase mutual exclusive **{Cl}**
- Se da **setul de invatare** (SI)
- **Problema** = obtinerea unor **reguli de clasificare** / construirea unui **AD** care clasifica corect nu numai **∀o∈SI** dar si **∀o∈U**

ID3 (Quinlan)

- Structura iterativa – fereastra din SI
- S-au gasit AD corecti in cateva iteratii pt 30 000 obiecte cu 50 atribute
- Empiric s-a aratat ca iterativ se obtin arbori mai buni decat daca s-ar construi din tot SI

- Utilizare AD
- Reguli de decizie

ID3 (Quinlan)

Metoda de constructie

- C = multmea de obiecte / ex inv. din SI
- A^k – atribut test cu valori a^k_1, .. a^k_v
- "divide-and-conquer"
- Impartirea/expandarea AD se opreste cand toate obiectele din C apartin unei aceleiasi clase
- Se termina intotdeauna (in cazul cel mai nefavorabil, cate un obiect in fiecare clasa)

ID3 – Exemplu

No.	Atribute				Clasa
	Vreme	Temperatura	Umiditate	Vant	
1	soare	cald	mare	fals	N
2	soare	cald	mare	adev	N
3	nori	cald	mare	fals	P
4	ploaie	placut	mare	fals	P
5	ploaie	racoare	normal	fals	P
6	ploaie	racoare	normal	adev	N
7	nori	racoare	normal	adev	P
8	soare	placut	mare	fals	N
9	soare	racoare	normal	fals	P
10	ploaie	placut	normal	fals	P
11	soare	placut	normal	adev	P
12	nori	placut	mare	adev	P
13	nori	cald	normal	fals	P
14	ploaie	placut	mare	adev	N

ID3 – Exemplu

Vreme
C_{ploaie} = {4P,5P,6N,10P,14N}
soare
nori
ploaie
C_{soare} = {1N,2N,8N,9P,11P}
Umiditate
P
Vant
mare
normal
adev
fals
N
P
N
P
C_{nori} = {3P,7P,12P,13P}

ID3 – Arbore minim

- Din acelasi SI se pot contrui diferiti AD
- Cum se poate obtine cel mai mic arbore (lama lui Occam) ?
- = Cum selectez atributul din radacina unui arbore?

ID3 – Cum selectez A?

- Multimea de exemple C cu un numar de ex pozitive $p \in$ P si exemple negative $n \in$ N

Se presupune ca:

- (1) Orice AD corect va clasifica obiectele proportional cu reprezentarea lor in C

  Un obiect (exemplu) o$\in$C va fi clasificat:

  - $\in$P cu probabilitatea $p/(p+n)$
  - $\in$N cu probabilitatea $n/(p+n)$

- (2) Cand un AD este utilizat pentru a clasifica obiecte, acesta intoarce o clasa $\Rightarrow$

  AD poate fi vazut ca o sursa a unui mesaj 'P' sau 'N' avand informatia necesara pentru a genera acest mesaj

Teoria informatiei ofera criteriul

- Pentru un univers de mesaje

  $M = \{m_1, m_2, ..., m_n\}$

  si o probabilitate $p(m_i)$ de aparitie a fiecarui mesaj, **continutul informational** ***I(M)*** al mesajelor din M se defineste astfel:

$$I(M) = \sum_{i=1}^{n} -p(m_i) log_2(p(m_i))$$

Selectia testului (atributului)

- Fie multimea de obiecte C cu $p \in$ P si $n \in$ N
- **Continutul de informatie** $I(AD_{p,n})$ este

$$I(AD_{p,n}) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

- Selecteaza A in radacina; valori A $\in \{a_1,\ldots,a_v\}$
- Fie C_i cu $p_i \in$ P si $n_i \in$ N, i=1,v
- Continutul de informatie pentru fiecare C_i este

$$I(AD_{pi,ni}),\ i=1,v$$

Selectia testului (atributului)

- Dupa selectarea lui A in radacina, **cantitatea de informatie necesara pentru a termina constructia arborelui** este suma ponderata a continutului de informatie din toti subarborii

$$E(A) = \sum_{i=1}^{v} \frac{p_i + n_i}{p + n} I(AD_{pi,ni})$$

- unde ponderea ramurii ***i*** este fractiunea de obiecte din C care apartin lui C_i;
- ***v*** este numarul de valori ale lui A

Selectia testului (atributului)

- **Castigul informational** al unui atribut A obtinut prin selectia acestuia ca radacina a arborelui de decizie este:

$$\mathbf{G(A) = I(AD_{p,n}) - E(A)}$$

- Se selecteaza A cu castig informational maxim
- Recursiv pentru a forma AD corespunzatori multimilor $C_1 \ldots C_v$

Calcul G(A) pt Ex

- 14 exemple, $9 \in P$, $5 \in N$
- **$I(AD_{p,n})$ =** 0.940 bits

$$-\frac{9}{14}\log_2\frac{9}{14}-\frac{5}{14}\log_2\frac{5}{14}$$

- vreme
  - soare - $2 \in P$, $3 \in N \Rightarrow$ **$I(AD_{p1,n1})$ = 0.971**
  - nori - $4 \in P$, $0 \in N \Rightarrow$ **$I(AD_{p2,n2})$ = ?**
  - ploaie - $3 \in P$, $2 \in N \Rightarrow$ **$I(AD_{p3,n3})$ = ?**

$$\frac{5}{14}I(AD_{p1,n1})+\frac{4}{14}I(AD_{p2,n2})+\frac{5}{14}I(AD_{p3,n3})$$

- **E(vreme) = 0.694 bits**
- **G(vreme) = 0.940-0.694= 0.246 bits**
- G(temperatura) = 0.029 bits
- G(umiditate) = 0.151
- G(vant) = 0.048 bits

C_{soare} = {1N,2N,8N,9P,11P}

C_{nori} = {3P,7P,12P,13P}

C_{ploaie} = {4P,5P,6N,10P,14N}

Generalizare la mai multe clase

- **Continutul de informatie**

$$I(Arb) = \sum_{i=1}^{v} - p(Cl = C_i) * \log_2 p(Cl = Ci)$$

- **Cantitatea de informatie necesara pentru a termina constructia arborelui**

$$E(A) = \sum_{i=1}^{v} \frac{|C_i|}{|C|} I(C_i)$$

- **Castigul informational**

$$\mathrm{G}(A) = \mathrm{I}(Arb) - \mathrm{E}(A)$$

Algoritm ID3

functie ind-arbore (set-invatare, atribute, default)

1. **daca** set-invatare = vid **atunci intoarce** frunza etichetata cu *default* sau "Failure"

Caz 1 – ex inv lipsa

2. **daca** toate exemplele din set-invatare sunt in aceeasi clasa
   atunci intoarce o frunza etichetata cu acea clasa

Bine = recunoaste

3. **daca** atribute este vida
   atunci intoarce o frunza etichetata cu disjunctia tuturor claselor din set-invatare

Caz 2 – atr inadecvate

4. selecteaza un atribut A, creaza nod pt A si eticheteaza nodul cu A
5. sterge A din atribute –> atribute1

6. ***m*** = cea mai frecventa clasa (set-invatare)
7. **pentru** fiecare valoare V a lui A **repeta**
   - fie $partitie_V$ multimea exemplelor din set-invatare, cu valorea V pentru A
   - creaza nod_V = **ind-arbore** ($partitie_V$, atribute1, ***m***)
   - creaza legatura nod A – nod_V etichetata cu V

sfarsit

Complexitate

- In fiecare nod cu exceptia frunzelor trebuie aflat G (castig informational) pt fiecare atribut A
- G depinde de valorile p_i si n_i pentru fiecare valoare a_i a lui A $\Rightarrow$ fiecare obiect din C trebuie investigat (clasa, valoare A) $\Rightarrow$
- **O(|C| * |A|)** , |A| - nr atribute
- Pentru fiecare iteratie, **complexitatea ID3**
- **O(|C| * |A| * |AD|)** , unde |AD| - numar de noduri interne AD

Cazuri speciale

Caz 1. Nu exista obiecte o$\in$ C pentru care A=A_j

(exemple de invatare lipsa)

- ID3 eticheteaza frunzele cu "*null*" sau "*Failure*" – deci nu clasifica in aceste noduri
- **Solutie**
  - Generalizeaza si se atribuie frunzei clasa cu cea mai mare frecventa de aparitie in C (cea mai frecventa)

Cazuri speciale: Zgomot

Caz 2. Informatia din SI este afectata de zgomot (attribute inadecvate)

- Zgomot
  - valori de atribute ale obiectelor din C afectate de zgomot
  - clasificare incorecta a obiectelor din C
- Erorile din C (zgomotele) pot duce la 2 probleme:
  - AD cu complexitate mai mare decat este necesar (a)
  - atribute inadecvate (b)

Cazuri speciale: Zgomot

Modificari necesare ID3 pt a trata zgomotul

- **(a)** Trebuie **sa decida daca testarea unor atribute** suplimentare va creste sau nu acuratetea predictiva a AD
- **(b)** Trebuie sa poata **lucra cu atribute inadecvate**

Cum se realizeaza (a)

Solutie

- G(A) > prag α absolut sau relativ
- α suficient de mare pt a elimina atribute nerelevante - dar elimina si atribute relevante pt cazul fara zgomot

Cazuri speciale: Zgomot atribute

Cum se realizeaza (b – atribute inadecvate)

- Trebuie produsa o eticheta pt C_i dar obiectele nu sunt in aceeasi clasa

Solutia 1

- Se utilizeaza notiunea de apartenenta la o clasa cu o anumita probabilitate, de ex. $p_i/(p_i+n_i)$

Solutia 2

- Eticheteaza cu clasa cea mai numeroasa: P daca $p_i>n_i$, N daca $p_i<n_i$, oricare (P sau N) daca $p_i=n_i$

Cazuri speciale: Extinderi C4.5

Caz 3. Valori necunoscute de atribute

3.1 Valori de atribute lipsa in setul de invatare

Solutia 1

- Atribuie valoarea cu cea mai mare frecventa

Solutia 2

- Foloseste probabilitati pt a determia distributia de probabilitate a valorilor lui A in C in functie de apartenenta la o clasa

$$prob(A = A_i \mid clasa = P) = \frac{prob(A = A_i \wedge clasa = P)}{prob(clasa = P)}$$

si alege valoarea cu cea mai mare probabilitate

Cazuri speciale: atribute lipsa SI

Solutia 3

- Construieste un AD pt a invata valorile atributelor lipsa
- $C' \subset C$, C' cu valori pt A
- In C' clasa este privita ca un atribut cu valori P sau N
- Valorile lui A – clasele de invatat
- Obtine AD' utilizat apoi pentru a clasifica obiectele din C-C' $\Rightarrow$ determin valoarea atributului A
- Solutia 3 > Solutia 2 > Solutia 1

Cazuri speciale: Extinderi C4.5

Caz 4. Atribute cu multe valori

- A_1 .. A_n - f multe valori simbolice sau valori numerice / continue, sau chiar valori aleatoare
- Castig mare $\rightarrow$ arbore cu 1 nivel

Solutia 1 (valori numerice)

- Partitionez in intervale $(A_i+A_{i+1})/2$, fiecare interval o valoare

Cazuri speciale: A cu multe valori

Exemplificare Solutia 1 (valori numerice)

- Partitionez in intervale $(A_i+A_{i+1})/2$, fiecare interval o valoare

- Ordonez valorile atributului din setul de invatare

64	65	68	69	70	71	72	72	75	75	80	81	83	85
da	nu	da	da	da	nu	nu	da	da	da	nu	da	da	nu

64.5 66.5 70.5 72 73.5 77.5 80.5 84

Cazuri speciale: Extinderi C4.5

Solutia 2 (valori numerice)

Pt fiecare A_i, i=1,m imparte obiectele in $(-\infty, A_i]$ si $(A_i, +\infty) \Rightarrow$ partitie P_i

- Pentru fiecare P_i calculeaza castigul informational si selecteaza partitia cu castig informational maxim

Cazuri speciale: A cu multe valori

Solutia 3 (valori simbolice)

- Utilizeaza **Informatia de separare** = cantitatea de informatie necesara pentru a determina valoarea unui atribut A intr-un set de invatare C
- Fie $P_{A,C}$ *distributia de probabilitate* a valorilor lui A

$$P_{AC} = \left(\frac{|A_1|}{|C|}, \ldots, \frac{|A_v|}{|C|}\right)$$

- **Informatia de separare**

$$ISep(A) = -\sum_{i=1}^{v} \frac{p_i + n_i}{p + n} \log_2 \frac{p_i + n_i}{p + n}$$

Cazuri speciale: A cu multe valori

$$GR(A) = \frac{G(A)}{ISep(A)}$$ sa fie cat mai mare

- Dar *GR* poate sa nu fie intotdeauna definita (*ISep*=0) sau sa tinda sa favorizeze *ISep* mici
- Selecteaza dintre atributele cu *G(A)* mare (peste medie) acelea cu *GR(A)* cel mai bun
- G(vreme) = 0.246 bits ←
- G(temperatura) = 0.029 bits
- G(umiditate) = 0.151 bits ←
- G(vant) = 0.048 bits
- ISep(vreme) = 1.578, ISep(umiditate) = 1 bit
- GR(vreme) = 0.246/1.578=0.156 ←
- GR(umiditate) = 0.151/1.0 = 0.151

Cazuri speciale: A cu multe valori

- Utilizeaza diverse metode de invatare pentru a forma clase din aceste valori sau a grupa aceste valori in grupuri (clustere), clase care sa devina valori discrete (sau sa produca mai putine valori) pentru atribut

5. Random forests

- Ipoteza metodei: combina mai multe modele de invatare a conceptului pentru a creste acurateta invatarii
- Bagging – obtine o medie a unor modele de invatare (fiecare poate fi afectat de zgomot sau de impartire favorizanta - biased)
- Idee - Selectam aleator din multimea de invatare o submultime de exemple si tot aleator selectam o submultime de atribute, apoi construim, pentru fiecare selectie cate un arbore (random selection of attributes and random seletion of examples)
- Obtinem o **padure de arbori**

Random forests

- **N** – numarul de exemple de invatare, **M** numarul de atribute
- Se vor selecta in fiecare arbore ***m*** atribute (***m***<M) si ***n*** (***n***<N) exemple, aleator, din setul de invatare (bootstrap sample)
- Restul exemplelor se folosesc pentru evaluare
- Fiecare arbore este complet construit

Random forests

M features
N examples
Location Similarity
Gene Expression
Domain-motif
Neighbor Function Similarity
Interact
Degree
Gene Expression
Interact
Not interact
Interact
Not interact
Interact
Not interact
Neighbor process similarity
Tissue Expression
Centraility
Interact
Not interact
Neighbor Function Similarity
PTM
Interact
Not interact
Interact
Not interact
Neighbor process similarity
Tissue Expression
Centraility
Interact
Not interact
Neighbor Function Similarity
PTM
Interact
Not interact
Interact
Not interact
Take the majority vote

Random forests

- Pentru o padure de arbori, si o instanta necunoscuta se clasifica instanta cu fiecare arbore
- Fiecare arbore aduce un vot, se considera clasificarea data de votul majoritar

Avantajele metodei

- Acuratete mare de invatare
- Reduce influenta unui model favorizant (biased)
- Mai putin sensibil la zgomote
- Totusi are tendinta la overfitting pentru anumite seturi de date

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 8

- **Regresie liniara**
- **Regresie liniara cu atribute multiple**
- **Arbori de regresie**

Model simplu

$T = \{\mathbf{X_1}, \mathbf{X_2}, \ldots, \mathbf{X_m}\}$ – **multime de invatare**

$$\mathbf{X_i} = \begin{pmatrix} x_1 \\ x_2 \\ . \\ . \\ x_n \end{pmatrix} \longrightarrow \boxed{h} \longrightarrow h(\mathbf{X_i})$$

$h \in \mathbf{H}$

$f(\mathbf{X}) = ?$

- **Invatare supervizata** – se cunosc $f(\mathbf{X_1}), \ldots, f(\mathbf{X_m})$

Gasim **h** a.i. $h(\mathbf{X_i}) = f(\mathbf{X_i})$, $i=1,m \rightarrow h(\mathbf{X_i}) = f(\mathbf{X_i})$, $\forall i$

Clasificare – f - valori discrete – grupeaza exemple

Regresie – f – valori continue, estimeaza sau prezice valori

1 Regresie liniara

- Algoritm de invatare supervizata
- *Invata valori continue pe baza exemplelor de invatare*

Regresie liniara

- Cum se reprezinta h?
- In acest exemplu - regresie liniara cu 1 variabila (1 atribut x)

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Functie de cost

- n – numar exemple de invatare
- x – variabila de intrare / atribut
- y – variabila de iesire (iesirea cunoscuta din exemple)
- Cum alegem parametrii θ_0 si θ_1 pentru a gasi modelul?
- Trebuie sa minimizam **<u>o functie de cost</u>**

Functie de cost

$h_\theta(x) = \theta_0 + \theta_1 x$

Functie de cost – eroare patratica

$$J(\theta_0,\theta_1) = \frac{1}{2n}\sum_{i=1}^{n}(h(x^i) - y^i)^2$$

Trebuie sa minimizam

$$\min_{\theta_0 \text{ si } \theta_1} J(\theta_0,\theta_1)$$

Cum minimizam o astfel de functie de cost?

Functie de cost

Trebuie sa alegem parametrii astfel incat $h_\theta(x)$ sa fie cat mai aproape de pct (x,y)

$$h_\theta(x) = \theta_0 + \theta_1 x$$

$\theta_0 = 1.5$
$\theta_1 = 0$

$\theta_0 = 0$
$\theta_1 = 0.5$

$\theta_0 = 1$
$\theta_1 = 0.5$

Presupunem numai drepte cu $\theta_0=0$

$h_\theta(x) = \theta_0 + \theta_1 x$

$\boldsymbol{\theta_1=1}$, $\theta_0=0$

$h_\theta(x) = \theta_1 x$

$$J(\theta_1) = \frac{1}{2n}\sum_{i=1}^{n}(h(x^i) - y^i)^2$$

θ_1**=0.5**, θ_0=0

$h_\theta(x) = \theta_1 x$

$$J(\theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (h(x^i) - y^i)^2$$

$\boldsymbol{\theta_1 = 0}$, $\theta_0 = 0$

$h_\theta(x) = \theta_1 x$

$$J(\theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (h(x^i) - y^i)^2$$

θ_1**=1,0.5, 0**, … θ_0=0

$h_\theta(x) = \theta_1 x$

$$J(\theta_1) = \frac{1}{2n} \sum_{i=1}^{n} (h(x^i) - y^i)^2$$

Pr fiecare valoare θ_1 avem o valoare diferita a $J(\theta_1)$

100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

Scaderea gradientului (Gradient descent)

- ***Metoda iterativa pentru gasirea minimului unei functii***
- Pentru a afla ***minimul*** unei functii, se fac pasi proportionali cu negativul derivatei/gradientului functiei in punctul curent
- Sau proportionali cu pozitivul derivatei/gradientului pentru ***maxim***
- Minime, respectiv maxime locale – nu garanteaza gasirea valorii optime globale

Gradient descent

Minimizarea functiei de cost $J(\theta_0,\theta_1)$

Initializeaza $\boldsymbol{\theta_0}$ $\boldsymbol{\theta_1}$

repeta pana la convergenta

pentru j=0 si j=1 (simultan) **executa**

$$\boldsymbol{\theta_j} \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

- α – viteza de invatare

$h(x) = \theta_0 + \theta_1 x$

$$\min_{\theta_0 \text{ si } \theta_1} \frac{1}{2n} \sum_{i=1}^{n} (h(x^i) - y^i)^2$$

Gradient descent

Minimizarea functiei de cost $J(\theta_0,\theta_1)$

Initializeaza $\theta_0\ \theta_1$

repeta pana la convergenta

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n}\sum_{i=1}^{n}(h(x^i) - y^i)$$

$$\theta_1 \leftarrow \theta_1 - \alpha \frac{1}{n}\sum_{i=1}^{n}(h(x^i) - y^i)\, x^i$$

n – numar exemple de invatare

Gradient descent

- Pe masura ce ne apropiem de miniml local, se vor face pasi din ce in ce mai mici
- Regim de invatare de tip batch – utilizeaza toate exemplele de invatare

- α – viteza de invatare
- Daca α prea mica – converge incet
- Daca α prea mare poate sa nu gaseasca minimul
- Daca initializam θ_0 θ_1 chiar cu valorile minimului – J nu se mai modifica (derivata partiala 0)

Regresie liniara multi-variabile

Consideram un numar de atribute x_1 .. x_m

Minimizarea functiei de cost $J(\theta_0,\theta_1,.. \theta_m)$

Initializeaza θ_j

repeta pana la convergenta

pentru j=0 la **m** (simultan) **executa**

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, .. \theta_m)$$

Regresie liniara multi-variabile

$$h(x) = \theta_0 + \theta_1 x_1 .. \theta_m x_m$$

Consideram un atribut suplimentar $\mathbf{x_0}$ care are valoarea **1** pentru orice exemplu

$$\mathbf{x} = \begin{bmatrix} x_0 \\ .. \\ x_m \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ .. \\ \theta_m \end{bmatrix}$$

$$h(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

Regresie liniara multi-variabile

$$h(x) = \theta_0 + \theta_1 x_1 \,..\, \theta_m x_m$$

$$h(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

$$J(\theta_0, \theta_1, .. \theta_m) = \frac{1}{2n} \sum_{i=1}^{n} (h(x^i) - y^i)^2$$

$$\min_{\theta_0 \ldots \theta_m} J(\theta_0, \theta_1, .. \theta_m)$$

Regresie liniara multi-variabile

Minimizarea functiei de cost $J(\theta_0, \theta_1, .. \theta_m)$

Initializeaza $\theta_0, \theta_1, .. \theta_m$

repeta pana la convergenta

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (h(x^i) - y^i)\, x^i$$

- Actalizeaza simultan θ_j cu j=1,m
- Functia de cost pentru regresie liniara este o functie convexa

Regresie liniara multi-variabile

- Functia de cost are **un singur minim global?**
- Daca **functia de cost este strict convexa** (daca domeiul parametrilor este strict convex, i.e., nu exista restrictii asupra coeficientilor, nu exista regularizare, etc.) atunci scaderea gradientului va avea **o soltie unica care va fi optima global**
- In caz contrar pot exista mai multe solutii,
- Exemplu

$y = (1-x)^2+(-1-x)^2 \quad |x|\geq 1/2$ solutii $x=\pm 1/2$

$x\in(-\infty,-1/2]\cup[1/2,\infty)$ - domeniu nonconvex

Scalarea valorilor de atribute

- Pentru ca GD sa convearga repede este bine ca valorile de atribute sa fie cam in acelasi interval

Ex: Suprafata casa $\in$ [0-1000] m^2

Numar camere 1-7

- GD converge foarte incet daca intervale mult diferite
- Scalam, de ex

$$x_1 = \text{sup}/2000 \quad x_2 = \text{nr cam}/7$$

- La fel pentru valori foarte mici, de ex

$x_1 \in [1..1000]$ si $x_2 \in [-0.000.1..+0.0001]$

2 Arbori de regresie

- **Arborii de regresie** sunt arbori de decizie dar pentru valori continue ale atributelor / variabile
- Realizeaza predictie
- Modelele predictive de regresie liniare sunt **modele globale** in care o singura formula predictiva trebuie (se presupune) sa fie valabila in tot spatiul
- Daca datele de intrare au multe atribute care interactioneaza in moduri complicate, detectarea unui singur model global poate fi ***problematica***

Arbori de regresie

- O metoda pentru a realiza regresie neliniara este partitionarea spatiului in regiuni mici in care interactiunile sunt mai simple
- ***Partitionare recursiva*** pana se ajunge la portiuni ale spatiului care sunt suficient de mici pentru a le putea modela prin modele simple
- Modelul are astfel **2 parti**: **partitionarea recursiva** si un **model simplu pentru fiecare partitie**

Arbori de regresie

- **Arborii de regresie** utilizeaza arbori de decizie pentru a reprezenta partitionarea
- **Arbori de regresie** sunt arbori de decizie cu valori continue
- **Frunzele** corespund unor regiuni din spatiul de intrare (vecinatati) care sunt **similare ca iesiri si cu intrari alaturate**
- Arborii de predictie / regresie sunt un fel de *nearest-neighbourhood adaptiv*

Functie f(X,Y)

Reprezentare f(X,Y)

Arbori de regresie

- O frunza $(x_1,y_1) \ldots (x_c,y_c)$ Exemplele din regiunea corespunzatoare frunzei
- $y_{est} = \frac{1}{n_c} \sum_{i=1}^{n_c} y_i$ Modelul pentru regiunea corespunzatoare frunzei
- Flexibilitatea si eficienta unui arbore sunt date de cate frunze are = in cate celule partitionam

Estimare cost
automobiile 1990
in functie de cai-putere
si apatament

Credit Cosma Shalizi

Arbore de regresie - Estimare cost automobiile 1990 in functie de cai-putere si apatament

All features have been standardized to have zero mean and unit variance
Credit Cosma Shalizi

Estimare cost case
in functie de pozitionare
pe harta

Arbore de regresie - Estimare cost case in functie de pozitionare pe harta

All features have been standardized to have zero mean and unit variance

Estimare cost case
in functie de pozitionare
pe harta

Criteriul de divizare

- Cat de mult cresc arborele?
- Fiecare nod are asociat un anumit numar de puncte
- Adaugarea unui nod trebuie sa reduca eroarea
- RMS- Root Mean Square

De ex RMS 0.41 primul arbore, RMS 0.32 al doilea

- $\text{RMSError} = \sqrt{\frac{\Sigma_{i=1,n}(y'_i - yi)2}{n}}$

Criteriul de oprire

- Care este criteriul de oprire?
  - fiecare nod contine mai putin de q puncte

    sau
  - divizarea creste informatia cu mai putin de o valoare de prag = scade eroarea cu mai putin de valoarea de prag

- Frecvent se utilizeaza patratul erorilor pentru un arbore T in loc de RMSError

Criteriul de oprire

- Suma erorilor patratice pentru un arbore T este

$$S = \sum_{c \in \text{leaves}(T)} \sum_{i \in c} (y_i - m_c)^2$$

$$m_c = \frac{1}{n_c} \sum_{i \in c} y_i$$

- Trebuie sa minimizam S

Construirea arborelui de regresie

1. Incepe cu 1 nod care contine toate punctele
2. Calculeaza $\mathbf{m_c}$ si **S** pentru acel nod
3. **daca** toate punctele din nod au aceeasi valoare pentru variabilele de intrare atunci **stop**

   altfel cauta dintre toate diviziunile binare ale variabilelor pe aceea care reduce S cat mai mult
4. **daca** scaderea in **S < prag** sau unul din nodurile obtinute prin diviziune are **mai putin de q** puncte atunci **stop**

   altfel creeaza 2 noduri si repeta de la 3

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 9

- **Regresie logistica**
- **Invatare “Ensemble”**

1. Regresie logistica

Invatare supervizata

Utilizata in clasificare

- Regresie logistica binomiala (binara)
- Regresie logistica multinomiala (multi-clasa)

- Am putea utiliza regresia liniara cu

  $h_\theta(x) \geq 0.5 \quad y=1$

  $h_\theta(x) < 0.5 \quad y=0$

- Nu este o idee buna

Ipoteza de invatare

$0 \leq h_\theta(x) \leq 1$

$h(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x})$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$h_\theta(x)$ estimeaza probabilitatea ca y=1 pentru o intrare x

$h_\theta(x) = P(\, y=1 \mid x; \theta)$

σ(z)

$\sigma(z) = 1 / (1 + e^{-z})$

Sigmoid

tanh(z)

$\tanh(z) = (e^{z} - e^{-z}) / (e^{z} + e^{-z})$

Tangenta hiperbolica

Ipoteza de invatare

$0 \leq h_\theta(x) \leq 1$ presupunem o relatie liniara intre 2 atribute x1 si x2 si consideram probabilitatea clasei y

$p = P(y=1 \mid x; \theta)$

$$l = ln\frac{p}{1-p} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

"log odds" (***l***)

$$\frac{p}{1-p} = e^{\theta_0+\theta_1 x_1+\theta_2 x_2}$$

$$p = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2)}}$$

Functia de cost

In regresie liniara avem functia de cost

$$J(\theta_0, \theta_1, .. \theta_m) = \frac{1}{2n}\sum_{i=1}^{n}(h(x^i) - y^i)^2$$

$$h_\theta(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$$

In regresie logistica

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Daca am utiliza aceeasi functie de cost aceasta nu ar mai fi convexa (deci nu un unic minim global)

Avem nevoie sa definim o alta functie de cost

Functia de cost

$Cost(h_\theta(\mathbf{x}),y) = -\ \log(h_\theta(\mathbf{x}))$ daca $y=1$

$-\log(1- h_\theta(\mathbf{x}))$ daca $y=0$

Inseamna ca daca $h_\theta(\mathbf{x}) = 0$ costul tinde la infinit deci predictia este $P(y=1 \mid x; \theta) = 0$

Scriem compact

$$Cost(h_\theta(\mathbf{x}),y) = -\ y \log(h_\theta(\mathbf{x})) - (1-y) \log(1- h_\theta(\mathbf{x}))$$

$$J(\theta) = -\frac{1}{2n} \sum_{i=1}^{n} y^i \log(h_\theta(\mathbf{x}^i)) + (1-y^i) \log(1- h_\theta(\mathbf{x}^i))$$

MLE – Maximum Likelihood Estimation

Clasificare

Pentru a face o predictie (a clasifica) trebuie sa determinam parametrii θ

min $_{\theta}J(\theta)$ cu $J(\theta) = -\frac{1}{2n}\sum_{i=1}^{n} y^i \log(h_\theta(\mathbf{x}^i)) + (1-y^i)\log(1-h_\theta(\mathbf{x}^i))$

Aplicam scaderea gradientului

Initializeaza θ_j

repeta pana la convergenta

pentru j=0 la **m** (simultan) **executa**

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, .. \theta_m)$$

Scaderea gradientului

Initializeaza θ_0 , θ_1 , .. θ_m

repeta pana la convergenta

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (h(x^i) - y^i)\, x^i$$

- Actualizeaza simultan θ_j cu j=0,m

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

($h_\theta(x) = \theta^T x$ *pt regresie liniara*)

Diferente

Regresie liniara

– Distributia este Gausiana

- Valorile prezise sunt continue

Regresie logistica

– Distributia este Bernoulli/Binomiala

- Valorile prezise sunt probabilitati pentru apartenenta (sau non-apartenenta) la o clasa

(Distributie Bernoulli – 1 eveniment cu iesire 1/0

Distributie categoariala – 1 eveniment cu k iesiri

Distributie binomiala – n evenimente cu iesire 1/0

Distributie multinomiala – n evenimente cu k iesiri posibile)

Algoritmi de optimizare

- Scaderea gradientului
- Scaderea conjugata a gradientului
- Broyden–Fletcher–Goldfarb–Shanno (BFGS) - metoda quasi-Newtoniana (tip hill climbing)

  (quasi-newtoniana – daca Hessiana este prea complicat de calculat, aceasta este actualizata prin analiza iterativa a vectorilor de gradient)
- Limited memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS)

BFGS nu neecsita stabilirea manuala a α

Clasificare multi-clasa

Clasificare "**one-versus-all**", K clase

Clasa **1** $h^1_\theta(x) = P(Y=1 \mid X; \theta_1)$ – determina setul de parametrii θ_1

Clasa **2** $h^2_\theta(x) = P(Y=2 \mid X; \theta_2)$ – determina setul de parametrii θ_2

etc.

Pentru o instanta necunoscuta x, determina $h^i_\theta(x)$ pt fiecare clasa si alege clasa pentru care $h^i_\theta(x)$ are valoarea maxima

Clasificare multi-clasa

Pentru K clase, se executa regresie logistica pentru K-1 modele de regresie logistica binara, din care o clasa este aleasa ca pivot si celelalte K-1 sunt clasificate in functie de pivot. Daca clasa K este aleasa ca pivot atunci:

$$\ln \frac{\Pr(Y_i = 1)}{\Pr(Y_i = K)} = \theta_1 \cdot \mathbf{X}_i$$

$$\ln \frac{\Pr(Y_i = 2)}{\Pr(Y_i = K)} = \theta_2 \ \mathbf{X}_i$$

$$\cdots\cdots$$

$$\ln \frac{\Pr(Y_i = K-1)}{\Pr(Y_i = K)} = \theta_{K-1} \cdot \mathbf{X}_i$$

Overfitting

Solutii pt minimizare

1) **Reducerea numarului de atribute**

- Manual
- Algoritmi de selectie a modelului

Poate reduce overfitting dar elimina informatie

2) **Regularizare**

Pastrez toate atributele dar reduc dimensiunea parametrilor θ_1 , .. θ_m

Potrivita daca avem multe atribute, fiecare contribuind cu putin/la fel la predictie

2. Regularizare

Tehnica de a minimiza overfitting

Idee = sa pastram θ mici, duce la modele mai simple

$$J(\theta_0, \theta_1, .. \theta_m) = \frac{1}{2n}\left[\sum_{i=1}^{n}(h(x^i) - y^i)^2 + \varphi \sum_{j=1}^{m}\theta_j^2\right]$$

$$\theta_j \leftarrow \theta_j - \alpha\left[\frac{1}{n}\sum_{i=1}^{n}(h(x^i) - y^i)\, x^i + \frac{\varphi}{m}\theta_j\right]$$

Termen de regularizare

pt j=1,m si

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}(h(x^i) - y^i)\, x^i$$

φ - Parametru de regularizare

Regularizare

Se poate rescrie

$$\theta_j \leftarrow \theta_j (1 - \alpha \frac{\varphi}{m}) - \alpha \frac{1}{n} \sum_{i=1}^{n} (h(x^i) - y^i)\ x^i$$

pt j=1,m

Primul termen de obicei mai mic ca 1, are efectul reducerii parametrilor

Al doilea termen actualizeaza parametrii conform scaderii gradientului

Observatii

- Regularizarea se aplica atat la regresie logistica cat si la regresie liniara.

  Formula de actualizare a θ_j , j=1,m este aceeasi (evident difera h(x))

- Regresia logistica poate beneficia, la fel ca cea liniara, de scalarea valorilor de atribute (vezi Curs 8)

3. Invatarea “Ensemble”

- Genereaza mai multe ipoteze din acelasi set de date
- Combina mai multe ipoteze de invatare

Metode

- Naïve Bayes
- Bagging – Random forests
- Boosting – construieste incremental o multime de ipoteze prin antrenarea fiecarei noi instante de model a.i. sa ia in considerare exemplele gresit clasificate de modelul anterior - Adaboost

Adaboost

- N exemple de invatare $\{(\mathbf{x_1}, y_1, \ldots (\mathbf{x_N}, y_N)\}$, M etape

Invatare

Initializare $D_i^1 = \frac{1}{N}$ pt i=1,N

pentru j=1,M **repeta**

1. Construieste un model h_j din setul de invatare utilizand distributia D^j
2. Defineste eroarea ponderata a modelului j

$$\varepsilon_j = \sum_{i:h_j(x_i) \neq y_i} D_i^j$$

Eroarea ponderata a modelului h_j

Adaboost

3. Obtine o ipoteza h_j care minimizeaza eroarea ε_j si satisface conditia $\varepsilon_j<1/2$

$$\alpha_j = \frac{1}{2}\log\left(\frac{1-\varepsilon_j}{\varepsilon_j}\right)$$

$$D_i^{j+1} = e^{-y_i h_j(x_i)\alpha_j} D_i^j$$

$$D_i^{j+1} = \frac{D_i^{j+1}}{\sum_{i=1}^{N} D_i^{j+1}}$$

Predictie

$$H(x) = sign\left[\sum_{j=1}^{M} \alpha_j h_j(x)\right]$$

Boosting

https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe

https://medium.com/mlreview/gradier boosting-from-scratch-1e317ae4587d

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 10

Prelucrarea limbajului natural

- **Prelucrare LN pentru comunicare**
- **Prelucrare LN pt achizitia cunostintelor**

Prelucrare LN pentru comunicare

1 Comunicare

- *Definitie*: schimbul intentional de informatie generat de producerea si perceperea semnelor dintr-un sistem partajat de semne conventionale

- Componentele comunicarii
  - intentie
  - generare
  - sinteza

  Emitator

  - perceptie
  - analiza
  - desambiguare
  - incorporare

  Receptor

Acte de comunicare

J. Austin - How to do things with words, 1962, J. Searle - Speech acts, 1969

Un act de comunicare:

- **locutie** = fraza spusa de locutor
- **illocutie** = intelesul dorit spre a fi comunicat de locutor (performativa)
- **prelocutie** = actiunea care rezulta din locutie

Maria i-a spus lui Ion: "**Te rog** inchide usa"

locutie illocutie continut

prelocutie: **usa inchisa**

Categorii ilocutionale

- **Asertive**
- **Directive**
- **Permisive**
- **Prohibitive**
- **Declarative**
- **Expresive**

2 Modele limbaj

- Modelele n-gram - bazate pe secvente de caractere, cuvinte
- Modele de limbaj bazate pe structura gramaticala
- Categorii lexicale (parts of speech): substantiv, adjectiv, etc
- Categorii sintactice: grup substantival, grup verbal, etc.
- Structuri de fraze

$A \to aB_{\alpha\gamma\beta}$

Modele limbaj

Gramatica / Clasa limbaj	Automat	Reguli de productie
Recursiv numarabile	Masina Turing	$\gamma \to \alpha$
Dependente de context	Masina Turing linear marginita	$\alpha A \beta \to \alpha \gamma \beta$
Independente de context	Automate Push down	$A \to \alpha$
Regulate	Automate cu stari finite	$A \to a$ $A \to aB$

Fernando Pereira: "*The older I get, the further donw the Chomsky hierarchy I go*"

Definire limbaj

- Lexicon, categorii deschise si inchise
- Analiza lexicala
- Analiza sintactica (pars oratoris)
- Gramatici
- Terminale, neterminale
- Reguli de rescriere/productii (LHS $\rightarrow$ RHS)
- Analiza semantica + analiza pragmatica

3 Analiza lexicala

Gramatici regulate

$G=(N,\Sigma,R,S)$

N este multimea de neterminale

Σ este multimea de terminale

R este multimea de reguli de rescriere

$$R:\{\ X \rightarrow a \textit{ sau } X \rightarrow aY \mid X,Y \in N \text{ si } a \in \Sigma)$$

S – simbolul de start

- Genereaza Lexiconul

4 Analiza sintactica

Gramatici independente de context

$G=(N,\Sigma,R,S)$

N este multimea de neterminale

Σ este multimea de terminale

R este multimea de reguli de rescriere

$$R:N \rightarrow (N \cup \Sigma)^*$$

S – simbolul de start

Gramatica pt sintaxa

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Proper\text{-}Noun$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$

Lexicon

$Det \rightarrow that \mid this \mid a$
$Noun \rightarrow book \mid flight \mid meal \mid money$
$Verb \rightarrow book \mid include \mid prefer$
$Pronoun \rightarrow I \mid she \mid me$
$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$Aux \rightarrow does$
$Preposition \rightarrow from \mid to \mid on \mid near \mid through$

Care este arborele de derivare/sintactic al frazei **Book that flight**?

Jurafsky and Martin, 2007

Analiza Top-down

Book that flight

- Analiza top-down cauta arborele de derivare incepand de la S catre frunze
- Pe fiecare nivel se expandeaza neterminalul cel mai din stanga

Jurafsky and Martin, 2007

Analiza Bottom-Up

Book that flight

- Porneste de la cuvintele din fraza si cauta sa obtina S
- Folosita si la analizoarele “shift-reduce” pt limbaje de programare

Book that flight

$\boxed{Det \rightarrow that} \mid this \mid a$

$\boxed{Noun \rightarrow book \mid flight} \mid meal \mid money$

$\boxed{Verb \rightarrow book} \mid include \mid prefer$

$S \rightarrow NP\ VP$

$S \rightarrow Aux\ NP\ VP$

$S \rightarrow VP$

$NP \rightarrow Pronoun$

$NP \rightarrow Proper\text{-}Noun$

$NP \rightarrow Det\ Nominal$

$\boxed{Nominal \rightarrow Noun}$

$Nominal \rightarrow Nominal\ Noun$

$Nominal \rightarrow Nominal\ PP$

$VP \rightarrow Verb$

$VP \rightarrow Verb\ NP$

$VP \rightarrow Verb\ NP\ PP$

$VP \rightarrow Verb\ PP$

$VP \rightarrow VP\ PP$

$PP \rightarrow Preposition\ NP$

Jurafsky and Martin, 2007

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Pronoun$
$NP \rightarrow Proper\text{-}Noun$
$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$
$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$
$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$

Book that flight

Jurafsky and Martin, 2007

Problema ambiguitatii sintactice

- Ambiguitate sintactica sau **ambiguitate structurala**
- Gramatica genereaza mai multi arbori de analiza

Problema ambiguitatii sintactice

Ambiguitate structurala

- **Ambiguitate de asociere** (attachment ambiguity) – un constituent poate fi atasat in arborele de derivare in mai mult de 1 loc
  - I hit a man with an umbrella
  - We saw the Eiffel Tower flying over Paris
- **Ambiguitate de coordonare** (coordination ambiguity) - parti de fraza care pot fi cuplate diferit de conjunctii cum ar fi “and”
  - Old men and women
  - He speaks good French and English

Analiza sintactica pt LN

- Ambiguitatea structurala poate genera multi arbori sintactici

(Old men and women speak good French and English)

- Pentru anumite gramatici, numarul posibil de arbori poate fi exponential in raport cu numarul de cuvinte din fraza
- O solutie – Gramaticile independente de context probabiliste

GICP (PCFG)

- Gramatici independente de context probabiliste (GICP)

  VP → Verb [0.70]

  | Verb NP [0.30]

- Determinare probabilitati
  - de proiectant
  - pe baza de treebanks (fraze deja anlizate corect), de ex. Penn Treebank (https://en.wikipedia.org/wiki/Treebank)

Gramatici regulate probabiliste

- Lexicon

Noun → **breeze** [0.10] | **wumpus** [0.15] | **ball** [0.15] …
Verb → **is** [0.10] | **see** [0.10] | **smells** [0.10] | **hit** [0.10]…
Adjective → **right** [0.10] | **left** [0.10] | **smelly** [0.15] …
Adverb → **here** [0.05] | **there** [0.05] | **ahead** [0.02] …
Pronoun → **me** [0.10] | **you** [0.03] | **I** [0.10] | **it** [0.10] …
RelPronoun → **that** [0.40] | **who** [0.20] ...
Name → **John** [0.1] | **Mary** [0.01] …
Article → **the** [0.40] | **a** [0.30] | **an** [0.10] …
Preposition → **to** [0.20] | **in** [0.10] | **on** [0.05] …
Conjunction → **and** [0.50] | **or** [0.10] | **but** [0.20]…

GIC probabiliste - sintaxa

- Sintaxa

S → NP VP	[0.90]	I feel a breeze	
| S Conjunction S	[0.10]	I feel a breeze and it stinks	
NP → Pronoun	[0.30]		I
| Name	[0.10]		John
| Noun [0.10]	[0.10]		pit
| Article Noun	[0.25]		the wumpus
| NP PP	[0.10]		the wumpus in 1,3
| NP RelClause	[0.05]		the wumpus that is smelly …..
VP → Verb	[0.40]		stinks
| VP NP	[0.35]		feel a breeze
| VP Adjective	[0.05]		smells dead
| VP PP	[0.10]		is in 1,3
| VP Adverb	[0.10]		go ahead
PP → Preposition NP	[1.00]		to the east
RelClause → RelPronoun VP	[1.00]		that is smelly

Analiza sintactica cu GICP

- Fiecare arbore de derivare intr-o GICP are asociata o probabilitate P(t)

  Ex: 0.9 x 0.25 x 0.2 x 0.1 x 0.4 x 0.1 = 0.00018

S → NP VP	[0.90]
NP → Art Noun	[0.25]
Art → every	[0.2]
Noun → wumpus	[0.1]
VP → Verb	[0.4]
Verb → smells	[0.1]

Analiza sintactica cu GICP

- Fiind data o GICP si o fraza ***s***, fie ***T(s)*** multimea de arbori de derivare asociati lui ***s***
- Ne intereseaza sa gasim

$$\arg\max_{t \in T(s)} P(t)$$

Avem 2 posibilitati:

1) **Generez toti arborii posibili**; calculez probabilitatile asociate si aleg arborele cu cea mai mare probabilitate

Ineficienta datorita numarului mare de arbori posibili

2) **Bazata pe programare dinamica / chart parsers**

Analizez subsiruri si memorez analizele partiale

Analiza CKY (Cocke–Younger–Kasami)

- Analiza CKY este o analiza “bottom-up” in care dupa ce analizam un subsir, se memoreaza rezultatul intr-o tabela (**chart)**, astfel incat sa il putem reutiliza - **chart parser**
- GIC – orice subsir/fraza analizata pe o ramura a arborelui sintactic poate fi utilizata pe alta ramura
- Chart parser-ul **CKY** (J. Cocke, D. Young, T. Kasami)
- Gramatica trebuie sa fie in **Forma Normala Chomsky (CNF)**; orice regula trebuie sa aiba una din doua forme:
  - $X \rightarrow a$ cu $a \in \Sigma$ (reguli lexicale)
  - $X \rightarrow Y\ Z$ cu $Y, Z \in N$ (reguli sintactice)

Transformarea in Chomsky Normal Form

Trebuie sa transformam urmatoare tipuri de reguli:

- Reguli care combina terminale cu neterminale

$X \rightarrow a\ Y\ [p]$ transform $X \rightarrow A\ Y\ [p]$ si $A \rightarrow a\ [1.0]$

- Reguli care au 1 singur neterminal in partea dreapta (unitare)

Se rescrie partea dreapta cu partea dreapta a tuturor regulilor neunitare

$X \rightarrow Y\ [p]$ si $Y \rightarrow a\ [p1]$ transform

$X \rightarrow a\ [p \times p1]$

Transformarea in CNF

- Reguli care au mai mult de 2 neterminale in dreapta

  Introduc noi neterminale

$X \rightarrow Y\ Z\ T\ [p]$ transform in

$X1 \rightarrow Y\ Z\ [p]$

$X \rightarrow X1\ T\ [1.0]$

- Orice GIC (nu neaparat GICP) poate fi astfel transformata in CNF

Analiza sintactica

- CKY pt GICP foloseste un spatiu de $O(N^2M)$, N – nr cuvinte (lungimea sirului de intrare), M – nr neterminale din gramatica, pentru a construi o **tabela de probabilitati P**
- Timp $O(N^3M)$
- Nu examineaza toti a.d. posibili ci calculeaza probabilitatea celui mai probabil a.d.
- Toti subarborii sunt implicit reprezentati in tabela P de unde se pot obtine daca dorim

Exemplul 1

The	flight	includes	a	meal
Det → the 0.5				
	N → flight 0.02			
		V → includes 0.05		
			Det → a 0.4	
				N → meal 0.01

Chart parser pas 1

$S \rightarrow NP\ VP\ \ 0.8$

$NP \rightarrow Det\ N\ \ 0.3$

$VP \rightarrow V\ \ NP\ \ 0.2$

$V \rightarrow includes\ \ 0.05$

$Det \rightarrow the\ \ 0.5$

$Det \rightarrow a\ 0.4$

$N \rightarrow meal\ \ 0.01$

$N \rightarrow flight\ 0.02$

Exemplul 1

The	flight	includes	a	meal
Det → the 0.5	NP → Det N 0.3 x 0.5 x 0.02 = 0.003			
	N → flight 0.02			
		V → includes 0.05		
			Det → a 0.4	NP → Det N 0.3 x 0.4 x 0.01= 0.0012
				N → meal 0.01

Chart parser pas 2

S → NP VP 0.8
NP → Det N 0.3
VP → V NP 0.2

V → includes 0.05
Det → the 0.5
Det → a 0.4
N → meal 0.01
N → flight 0.02

Exemplul 1

The	flight	includes	a	meal
Det → the 0.5	NP → Det N 0.3 x 0.5 x 0.02= 0.003			S → NP VP 0.8 x 0.003 x 0.000012= 0.0000000288
	N → flight 0.02			
		V → includes 0.05		VP → V NP 0.2 x 0.05 x 0.0012= 0.000012
			Det → a 0.4	NP → Det N 0.3 x 0.4 x 0.01= 0.0012
				N → meal 0.01

Chart complet

S → NP VP 0.8
NP → Det N 0.3
VP → V NP 0.2

V → includes 0.05
Det → the 0.5
Det → a 0.4
N → meal 0.01
N → flight 0.02

Exemplul 2

Fish people fish tanks

G initiala

$S \rightarrow NP\ VP$	0.9
$S \rightarrow VP$	0.1
$VP \rightarrow V\ NP$	0.5
$VP \rightarrow V$	0.1
$NP \rightarrow NP\ NP$	0.2
$NP \rightarrow N$	0.7
$N \rightarrow people$	0.5
$N \rightarrow fish$	0.2
$N \rightarrow tanks$	0.2
$V \rightarrow people$	0.1
$V \rightarrow fish$	0.6

G transformata in CNF

$S \rightarrow NP\ VP$	0.9
$S \rightarrow V\ NP$	0.05
$S \rightarrow people$	0.001
$S \rightarrow fish$	0.006
$VP \rightarrow V\ NP$	0.5
$VP \rightarrow people$	0.01
$VP \rightarrow fish$	0.06
$NP \rightarrow NP\ NP$	0.2
$NP \rightarrow people$	0.35
$NP \rightarrow fish$	0.14
$NP \rightarrow tanks$	0.14
$N \rightarrow people$	0.5
$N \rightarrow fish$	0.2
$N \rightarrow tanks$	0.2
$V \rightarrow people$	0.1
$V \rightarrow fish$	0.6

Fish	people	fish	tanks
S → fish 0.006 N → fish 0.2 V → fish 0.6 NP → fish 0.14 VP → fish 0.06	**S → NP VP** 0.9*0.14*0.01=0.00126 **S → V NP** 0.05*0.6*0.35 = 0.0105 **VP → V NP** 0.5*0.6*0.35 **NP → NP NP** 0.2*0.14*0.35		
	S → people 0.001 N → people 0. 5 V → people 0.1 NP → people 0.35 VP → people 0.01	**S → NP VP** 0.9*0.35*0.06=0.0189 **S → V NP** 0.05*0.1*0.14 = 0.0007 **VP → V NP** 0.5*0.1*0.14 **NP → NP NP** 0.2*0.35*0.14	
		S → fish 0.006 N → fish 0.2 V → fish 0.6 NP → fish 0.14 VP → fish 0.06	**S → V NP** 0.05*0.6*0.14 = 0.0042 **VP → V NP** 0.5*0.6*0.14 **NP → NP NP** 0.2*0.14*0.14
			N → tanks 0.2 NP → tanks 0.14

S → NP VP	0.9
S → V NP	0.05
S → people	0.001
S → fish	0.006
VP → V NP	0.5
VP → people	0.01
VP → fish	0.06
NP → NP NP	0.2
NP → people	0.35
NP → fish	0.14
NP → tanks	0.14
N → people	0.5
N → fish	0.2
N → tanks	0.2
V → people	0.1
V → fish	0.6

Exemplul 2

adaptat din Jurafsky and Martin, 2007

Fish	people	fish	tanks
S → fish 0.006 N → fish 0.2 V → fish 0.6 NP → fish 0.14 VP → fish 0.06	S → NP VP 0.9*0.14*0.01=0.00126 S → V NP 0.05*0.6*0.35 = 0.0105 VP → V NP 0.5*0.6*0.35 NP → NP NP 0.2*0.14*0.35		
	S → people 0.001 N → people 0. 5 V → people 0.1 NP → people 0.35 VP → people 0.01	S → NP VP 0.9*0.35*0.06=0.0189 S → V NP 0.05*0.1*0.14 = 0.0007 VP → V NP 0.5*0.1*0.14 NP → NP NP 0.2*0.35*0.14	
		S → fish 0.006 N → fish 0.2 V → fish 0.6 NP → fish 0.14 VP → fish 0.06	S → V NP 0.05*0.6*0.14 = 0.0042 VP → V NP 0.5*0.6*0.14 NP → NP NP 0.2*0.14*0.14
			N → tanks 0.2 NP → tanks 0.14

S → NP VP	0.9
S → V NP	0.05
S → people	0.001
S → fish	0.006
VP → V NP	0.5
VP → people	0.01
VP → fish	0.06
NP → NP NP	0.2
NP → people	0.35
NP → fish	0.14
NP → tanks	0.14
N → people	0.5
N → fish	0.2
N → tanks	0.2
V → people	0.1
V → fish	0.6

Exemplul 2

Chart incomplet
De completat de catre studenti
pt examen
De gasit cel mai probabil arbore de analiza

Analiza sintactica

- CYK pt GICP foloseste o **tabela de probabilitati P** de dimensiuni M x N x N (M-nr neterm, N- nr cuv)
- **P(i,j,X)** - contine probabilitatea maxima ca subsirul de la ***i*** la ***j*** sa fie generat de neterminalul **X**
- Intoarce matricea P cu aceste probabilitati calculate
- Foloseste si o matrice **Back(i,j,X)** care indica pentru subsirul de la ***i*** la ***j*** generat de **X** $\rightarrow$Y Z unde s-a impartit subsirul a.i. sa fie generat de **X**

algoritm CYK(sir, gramatica) **intoarce** P, o tabela de probabilitati, si T, cel mai probabil arbore

N ← Lungime (sir) /* sir = $a_1\ a_2 \ldots a_N$ */

M ← nr de neterminale in gramatica

P ← matrice de N x N x M, initial 0

Back ← matrice N+1 x N+1 x M

/* *insereaza reguli lexicale pt fiecare cuvant* */

pentru i=0 **..** N-1 **repeta**

  pentru fiecare regula de forma (X → sir_i, [p]) **repeta**

    P[i, i+1,X] ← p

    (eventual si pointer catre regula)

algoritm CYK(sir, gramatica) **intoarce** P, o tabela de probabilitati, si T,
cel mai probabil arbore

```
/* combina primul si al doilea neterminal
 din RHS a regulilor sintactice , incepand cu cele mai scurte */
pentru span = 2 .. N repeta    /* lungimea analizata */
    pentru begin = 0 .. N-span repeta   /* inceput analiza  */
        end ← begin + span
        pentru split = begin+1 .. end-1 repeta  /* partitie analizata  */
            pentru fiecare regula de forma (X →Y Z [p]) repeta
                daca p > P[begin, end, X] atunci
                        P[begin, end, X] ←
                                P[begin, split, Y] x P[split, end, Z] x p)
                                (eventual si pointer catre regula)
                        Back[begin,end, X] ←  (split, Y,Z)
intoarce P, construiesteArb(Back, P)
sfarsit
```

Observatii

- Algoritmul se poate modifica pentru a trata si productii unare ($A \rightarrow B$)
- Algoritmul se poate modifica si pentru a trata productii vide
- Existenta productiilor cu 2 terminale este esentiala
- Exista si alti algoritmi de tip “chart parser”:
  - Earley – top down
  - Chart parsing – un fel de agenda-based

Analiza sintactica partiala

- Multe taskuri NLP nu necesita arbori de analiza completi (si deci de complexitate mare)
- Pentru aceste taskuri este suficient sa se realizeze o analiza partiala (***partial parse*** or ***shallow parse***)
- *Sistemele de extragere a informatiei* nu extrag toata informatia posibila din text, ci identifica si analizeaza acele segmente de text care contin informatii de interes
- *Sistemele de regasire a informatiei* pot indexa documentele pe baza unei subsmultimi de constituenti din text

5 Definite Clause Grammar (DCG)

- Utilizarea Logicii Predicatelor pentru reprezentarea gramaticilor
- Gramatici cu clauze definite (DCG)
- Fac parte din categoria gramaticilor semantice
- Fiecare regula din gramatica poate fi vazuta ca o regula din GIC

Definite Clause Grammar (DCG)

- Fiecare **categorie sintactica** se reprezinta printr-un **predicat cu un argument sir**
- NP(s) este adevarat daca **s** este NP

$$S \rightarrow NP\ VP \textit{ devine}$$

$$NP(s1) \wedge VP(s2) \Rightarrow S(append(s1,s2))$$

- Parsing = inferenta logica
- Bottom-up parsing – forward chaining
- Top-down parsing – backward chaining
- Aceeasi gramatica poate fi utilizata atat pentru analiza cat si pentru generare

In BNF

$S \rightarrow NP\ VP$

In LP / DGC

$NP(s_1) \wedge VP(s_2) \Rightarrow S(Append(s_1, s_2))$

BNF

$Noun \rightarrow ball \mid book$

In LP / DGC

$(s = \text{“ball”} \vee s = \text{“book”}) \Rightarrow Noun(s)$

BNF, DCG, Prolog

BNF	FOPL/DCG	PROLOG
S → NP VP NP → Noun Noun → stench Noun → wumpus VP → Verb Verb → smells Verb → kills	NP(s1) ∧ VP(s2) ⇒ S(append(s1,s2)) Noun(s) ⇒ NP(s) Verb(s) ⇒ VP(s) (s = “stench” ∨ s = “wumpus”) ⇒ Noun(s) (v = “smells” ∨ v = “kills”) ⇒ Verb(v)	sentence([S1, S2]) :- np(S1), vp(S2). np(S):- noun(S). vp(S):- verb(S). noun(stench). noun(wumpus). verb(smells). verb(kills). ?- sentence([wumpus, smells]). ?-sentence([S1, S2]).

Imbogatire DCG

- Imbogatesc neterminale cu argumente suplimentare
- Argumentele suplimentare permit:
  - Verifica corectitudinea gramaticala
  - Ataseseaza semantica
  - Adauga expresii / functii care se testeaza

Argument pt semantica

DCG	FOPL	PROLOG
S(sem) $\rightarrow$ NP(sem1) VP(sem2) {compose(sem1, sem2, sem)}	NP(s1, sem1) $\wedge$ VP(s2, sem2) $\Rightarrow$ S(append(s1, s2)), compose(sem1, sem2, sem)	slide urmator

semantica compozitionala

The dog has legs.	(caine *parti* picioare)
The ball is yellow.	(minge *proprietate* galbena)
The ball is red.	(minge *proprietate* rosie)
The dog bites.	(caine *actiune* musca)

```
sentence(S, Sem) :- np(S1, Sem1), vp(S2, Sem2), append(S1, S2, S),
                                                    Sem = [Sem1 | Sem2].
np([S1, S2], Sem) :- article(S1), noun(S2, Sem).
vp([S], Sem) :- verb(S, Sem1), Sem = [actiune, Sem1].
vp([S1, S2], Sem) :- verb(S1,_), adjective(S2, Sem1), Sem = [proprietate, Sem1].
vp([S1, S2], Sem) :- verb(S1,_), noun(S2, Sem1), Sem = [parti, Sem1].
noun(dog,caine).
noun(ball,minge).
noun(legs,picioare).
verb(bytes,musca).
verb(is,este).
verb(has,are).
adjective(yellow,galbena).
adjective(red,rosie).
```

S(sem) → NP(sem1) VP(sem2)
{compose(sem1, sem2, sem)}
VP(sem) → verb(sem1)
{compose(actiune, sem1, sem)}
VP(sem) → verb,adjective(sem1)
{compose(proprietate, sem1, sem)}
VP(sem) → verb,noun(sem1)
{compose(parti, sem1, sem)}

The dog has legs.	(caine *parti* picioare)
The ball is yellow.	(minge *proprietate* galbena)
The ball is red.	(mine *proprietate* rosie)
The dog bites.	(caine *actiune* musca)

```
sentence(S, Sem) :- np(S1, Sem1), vp(S2, Sem2), append(S1, S2, S),
                                                Sem = [Sem1 | Sem2].
np([S1, S2], Sem) :- article(S1), noun(S2, Sem).
vp([S], Sem) :- verb(S, Sem1), Sem = [actiune, Sem1].
vp([S1, S2], Sem) :- verb(S1,_), adjective(S2, Sem1), Sem = [proprietate, Sem1].
vp([S1, S2], Sem) :- verb(S1,_), noun(S2, Sem1), Sem = [parti, Sem1].
noun(dog,caine).
noun(ball,ball).
noun(legs,picioare).
verb(bytes,musca).
verb(is,este).
verb(has,are).
adjective(yellow,galbena).
adjective(red,rosie).
```

```
?- sentence([the,ball,is,yellow],Sem).
Sem = [minge, proprietate, galbena]
Yes
?- sentence([the,dog,bytes],Sem).
Sem = [caine, actiune, musca]
Yes
?- sentence([is,dog,bytes],Sem).
No
?- sentence([the,dog,has,legs],Sem).
Sem = [caine, parti, picioare]
Yes
```

Verificare corectitudine gramaticala

- Cazuri
- Subcategorii verbe: complementul pe care il poate accepta un verb
- Acord subiect predicat
- etc.

- Parametrizarea neterminalelor

<u>Cazuri</u>

Nominativ (subjective)

I take the bus	Eu iau autobuzul
You take the bus	Tu iei autobuzul
He takes the bus	El ia autobuzul

Acuzativ (objective)

He gives me the book	Imi da cartea

S → NP(Subjective) VP

NP(case) → Pronoun (case) | Noun | Article Noun

	//	I
VP → VP NP(Objective)	//	believe him
VP → VP PP	//	turn to the right

VP → VP Adjective

VP → Verb

PP → Preposition NP(Objective)

Pronoun(Subjective) → I | you | he | she

Pronoun(Objective) → me | you | him | her

```
sentence(S) :- np(S1,subjective), vp(S2), append(S1, S2, S).
np([S], Case) :- pronoun(S, Case).
np([S], _ ) :- noun(S).
np([S1, S2], _ ) :- article(S1), noun(S2).
pronoun(i, subjective).
pronoun(you, _ ).
pronoun(he, subjective).
pronoun(she, subjective).
pronoun(me, objective).
pronoun(him, objective).
pronoun(her, objective).
noun(ball).
noun(stick).
article(a).
article(the).
```

S → NP(Subjective) VP
NP(case) → Pronoun (case) | Noun | Article Noun
VP → VP NP(Objective)
VP → VP PP
VP → VP Adjective
VP → Verb
PP → Preposition NP(Objective)
Pronoun(Subjective) → I | you | he | she
Pronoun(Objective) → me | you | him | her

Subcategorii verbe

- Lista de subcategorii: ce complemente accepta verbul; depinde de verb

S → NP(Subjective) VP(subcat)

VP(subcat) → {subcat = np} VP(np) NP(Objective)
| {subcat = adj} VP(adj) Adjective
| {subcat = pp} VP (pp) PP
| Verb

Subcategorii verbe

VP(subcat) → {subcat = np} VP(np) NP(Objective)
| {subcat = adj} VP(adj) Adjective
| {subcat = pp} VP (pp) PP
| Verb

smell	[NP]	smell a wumpus
	[Adjective]	smell awfull
	[PP]	smell like a wumpus
is	[Adjective]	is smelly
	[PP]	is in box
	[NP]	is a pit
give	[NP, PP]	give the gold in box to me
	[NP, NP]	give me the gold
died	[]	died

Combinat cazuri cu subcategorii verbe

S → NP(Subjective) VP(subcat)
NP(case) → Pronoun (case) | Noun | Article Noun

Pronoun(Subjective) → I | you | he | she
Pronoun(Objective) → me | you | him | her

VP(subcat) → {subcat = np} VP(np) NP(Objective)
| {subcat = adj} VP(adj) Adjective
| {subcat = pp} VP (pp) PP
| Verb
| VP(subcat) PP
| VP(subcat) Adverb

S → NP(Subjective) VP(subcat)

VP(subcat) → **{subcat = np} VP(np) NP(Objective)**

| {subcat = adj} VP(adj) Adjective

| {subcat = pp} VP (pp) PP

| Verb

| **VP(subcat) PP**

| VP(subcat) Adverb

Reprezentare in Prolog

```
sentence(S) :- np(S1,subjective), vp(S2,Subcat), append(S1, S2, S).
 … pt NP
 vp([S],np) :- vp(S1,np), np(S2,objective), append(S1, S2, S).
 vp([S],adj) :- vp(S1,adj), adjective(S2),append(S1, S2, S).
 vp([S],pp) :- vp(S1,pp), pp(S2),append(S1, S2, S).
 vp([S],_) :- verb(S).
 vp([S],Subcat) :- vp(S1,Subcat), pp(S2),append(S1, S2, S).
 vp([S],Subcat) :- vp(S1,Subcat), adverb(S2),append(S1, S2, S).
```

Probleme datorita recursivitatii

Trebuie eliminata recursivitatea pentru a evita cicluri infinite

Solutie pentru eliminare recursivitate

vp([S],np) :- vp(S1,np), np(S2,objective), append(S1, S2, S).
vp([S],Subcat) :- vp(S1,Subcat), pp(S2),append(S1, S2, S).

Se transforma in

vp([S], Subcat) :- Subcat = np, vp1(S1, np), np(S2, objective), append(S1, S2, S).
vp([S],Subcat) :- vp1(S1, Subcat), pp(S2), append(S1, S2, S).

vp1([S],np) :- verb(S).
verb(give).
verb(make).

Gramatici semantice

- DCG au inspirat gramaticile semantice
- Gramaticile lingvistice (de ex GICP) utlizeaza neterminale sintactice (noun, verb, etc.)
- Gramaticile semnatice au pe post de neterminale categorii semantice, specifice unui domeniu

Top 10 artists in US for the last 3 weeks

limit entity geo date

6. Analiza semantica

- Elmentele de baza
  - Entitati / instante
  - Concepte / clase
  - Relatii – intre entitati si concepte
  - Predicate – verbe, roluri semantice

Analiza semantica - abordari

- Retele semantice
- Dependente conceptuale
- Gramatici de caz (Case Grammars)
- Grafuri conceptuale

Retele semantice

- Radu i-a trimis Ioanei o scrisoare.
- Radu este student.
- Ioana este eleva.
- Adresa lui Radu este Luterana, 15.

- Ocupatie (radu, student)
- Ocupatie (ioana, eleva)
- Trimite (radu, ioana, scrisoare)
- Adresa (radu, luterana - 15)

(b)

Retele semantice

- **Mostenirea proprietatilor (atributelor)**

  1) Mostenirii proprietatilor de la clasa la instanta:

  Daca un obiect O_1 este o particularizare (legat prin relatia ISA) a unui obiect generic/ clasa O si clasa O are un atribut (proprietate) A, atunci si instanta O_1 are atributul A.

  2) Aplicarea mostenirii proprietatilor intre o clasa si o superclasa, de-a lungul unei relatii sau a unui lant de relatii AKO

  Daca o clasa C_1 este o subclasa a unei clase C (legata prin una sau mai multe relatii AKO) si clasa C are proprietatea A, atunci clasa C_1 are de asemenea proprietatea (atributul) A.

7 Analiza pragmatica

- Interpretare pragmatica – utilizare si efect asupra ascultatorului
- Indexical – refera situatia curenta, contextul (eu, acum, aici, atunci)
- Speech acts (comezi, asertiuni)
- Ambiguitati

Dezambiguizare

- Semantica lexicala
- **Elemente de ambiguitate lexicala**
  - Hiponimie – relatie intre un teren generic si instante ale termenului generic (culoare / rosu galben)
  - Omonimie – aceeasi ortografie dar sensuri diferite (broasca)
  - Polisemie – cuvinte cu sensuri diferite dar corelate (banca)
  - Sinonimie – forme lexicale diferite dar acelasi sens sau sensuri apropiate (vulnerabil, firav, slab, nerezistent)
  - Antinomie – simetrie a intelesului in fct de o axa, de ex proprietate, utilizare etc (agreabil / dezagreabil, adevarat/fals)

Dezambiguizare

- Sintactica – arbori diferiti de analiza
- Referentiala – referire la obiecte anerioare

  (Mihai i-a spus lui Tudor ca a luat examenul)
- Pragmatica – referire la loc, timp
- Ambiguitati intre semnificatia uzuala si figurativa
  - Metonimie (a numi un obiect prin intermediul unui termen desemnând un alt obiect, legat cu primul printr-o relatie logica)
  - Metafora (figura de stil prin care se trece de la sensul obișnuit al unui cuvânt la alt sens, prin intermediul unei comparații subînțelese)

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 11

Prelucrarea limbajului natural

a) Prelucrare LN pentru comunicare (curs 11)

b) Prelucrare LN pt achizitia cunostintelor

Modele ale limbajului

Modele de limbaj bazate pe structura gramaticala

Modele n-Gram

POS tagging

Word embedding

1 Modele N-gram

- **Model N-gram de caractere** – distributie de probabilitate peste secvente de caractere
- **$P(c_{1:N})$** – probabilitatea unei secvente de N caractere, c_1 la c_N

  P("the") = 0.27 P("zgq")=0.00000002
- O secventa de simboluri de lungime n – **n-gram**
  - unigram, bigram, trigram
- Un model N-gram este definit ca un lant Markov de ordin N-1 (probabilitatea unui caracter depinde de caracterele precedente)

Modele N-gram de caractere

- **Trigram**

$$P(c_i|c_{1:i-1}) = P(c_i|c_{i-2:i-1})$$

$$P(c_{1:N}) = \Pi_{i=1,N}\, P(c_i|c_{1:i-1}) = \Pi_{i=1,N}\, P(c_i|c_{i-2:i-1})$$

Un model trigram a unui limbaj de 100 caractere

$$P(c_i|c_{i-2:i-1})$$

are 1 mil intrari

Modele N-gram de caractere

Ce putem face cu un astfel de model?

- **Identificarea limbajului** = fiind dat un text se determina in ce limba este scris (99%)
- Construieste un model trigram caracter pentru fiecare limbaj candidat *l*
- $P(c_i|c_{i-2:i-1},l)$; este nevoie de aprox 100 000 caractere pt fiecare limbaj

$$l^* = argmax_l\, P(l|c_{1:N}) = argmax_l\, P(l) * P(c_{1:N}|l) =$$
$$argmax_l\, P(l) * \Pi_{i=1,N}\, P(c_i|c_{i-2:i-1},l)$$

- **Alte aplicatii:**
  - verificare ortografie, clasificare texte in fct de tipuri, identificarea numelor proprii,…

Modele N-gram de caractere

Omogenizarea modelelor

- *Problema*: pt secvente de caractere comune, cam orice corpus va da o estimare buna (P(" th")=0.15
- Dar P(" ht")=0 http?
- *Solutie*
- Calculam N-gram si pentru secvente cu P=0 sau f mica
- Calculam N-1-gram + interpolare

$$\mathbf{P}(c_i|c_{i-2:i-1},l) = \alpha_1 P(c_i|c_{i-2:i-1},l) + \alpha_2 P(c_i|c_{i-1},l) + \alpha_3 P(c_i)$$

cu $\alpha_1 + \alpha_2 + \alpha_3 = 1$

Evaluarea modelelor – prin validare incrucisata

Modele N-gram de cuvinte

- In acest caz vocabularul este mult mai mare
- Daca max 100 car in cele mai multe limbaje, sute de mii, milioane de cuvinte
- Cuvinte noi
- Putem adauga un cuvant <NEC> in vocabular (sau mai multe)
- Bigramele si trigramele sunt mai bune decat unigramele

Modele N-gram de cuvinte - ex

Fie acest mic corpus

<s> I am Sam </s>

<s> Sam and I </s>

<s> I do not like green eggs and ham </s>

Calculul probabilitatilor unor bigrame pt acesta

P(I|<s>) = 2/3 =0.67 P(Sam|<s>)=1/3=0.33

P(am|I)=1/3=0.33 P(</s>|Sam)=1/2=0.5

P(and|Sam)=1/2=0.5 P(do|I)=1/3=0.33

Daca avem probabilitatile tuturor bigramelor putem calcula

P(<s> Sam and I </s>)=

P(<Sam|<s>)*P(and|Sam)*P(I|and)*P(</s>|I)

2 Clasificarea textelor

- Clasificare sau apartenenta la o clasa
- Identificare limbaj, clasificare tip text, analiza starii induse, detectie spam

Detectie spam

Not-spam (Ham)	Spam
m1, m2,…	n1, n2,…

- "for cheap" "you can buy" – n-gram de cuvinte
- "yo,u d-eserve" – n-gram de caractere

Detectie spam

- Calculez P(Mesaj|Spam) P(Mesaj|Ham)
- Clasific mesaj nou

$$\underset{C\in\{Spam,Ham\}}{argmax}\ P(C|Mesaj) = \underset{C\in\{Spam,Ham\}}{argmax}\ P(Mesaj|C)*P(C)$$

unde P(C) este estimat prin numararea nr de mesaje din Spam si Ham

- Se reprezinta mesajul ca o multime de caracteristici (car_i,val_i) si se poate aplica un algoritm de clasificare (invatare)
- Caracteristici – cuvinte din vocabular
- Valori – nr de aparitii in mesaj
- Alg posibili: K-nearest-neigh, SVM, AD, Bayes naiv

3 Regasirea informatiei

- Scop: Gasirea documentelor (de obicei text nestructurat) care sunt relevante pentru o cerere utilizator, dintr-o colectie de documnte

Un sistem de IR (Information Retrieval) poate fi caracterizat de:

- **Un corpus de documente** – paragrafe, pagini, texte pe mai multe pagini
- **Interogarea (query) in limbajul de interogare** – lista cuvinte, cuvinte adiacente, op logici, op nelogici (near)
- **Multimea rezultat** – multimea de documente relevante pentru query
- **Prezentarea rezultatelor** – lista ordonata, grafic, etc.

Evaluarea rezultatului

- **Precizie (precision) – procentul de documente relevante din multimea obtinuta**
- **Relevanta (recall) – procentul de documente relevante din colectie care sunt in setul rezultat**
- 100 documente cu 1 query pt care obtinem o multime de 40

	In multime	Nu in multime
Relevant	30	20
Nerelevant	10	40

- **Precision** = 30/(30+10) = 0.75
- **Recall** = 30/(30+20) = 0.6
- Recall este mai greu de calculat
- Se pot combina 2PR/(P+R)

Regasirea Booleana

Primele sisteme IR – **Boolean keyword model**

- Fiecare cuvant din text tratat ca un flag
- Limbajul de interogare – expresii logice peste cuvinte
- Ce piese ale lui Shakespeare contin cuvintele ***Brutus*** *AND* ***Caesar*** but *NOT* ***Calpurnia***?

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 daca piesa continue cuvantul, 0 altfel

Regasirea Booleana

Vectori booleni (0/1)

Pentru a raspunde la query: se iau vectorii pt Brutus, Ca esar si Calpurina

- 110100 *AND*
- 110111 *AND*
- 101111 =
- **100100**

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Regasirea Booleana

Pt 1 milion de documente cu aprox 1000 cuvinte fiecare, rezulta o matrice f mare

O reprezentare mai buna: index inversat

Pt fiecare termen se memoreaza lista documentelor care contin t

Constructia indexului inversat

Etape initiale in prelucrarea textului

- Tokenization
  - Determina token (analiza lexicala)
    - ***John's***
- Normalizare
  - Mapeaza textul si termenul din query la aceeasi forma
    - ***U.S.A.*** si ***USA***
- Stemming
  - Forme diferite pt o aceeasi radacina sa se potriveasca
    - ***authorize***, ***authorization***
- Eliminarea cuvintelor comune (stop words)
    - ***the, a, to, of***

Secventa tokens

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Secventa tokens sortata

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Dictionar si postings

term	doc. freq.	→	postings lists
ambitious	1	→	2
be	1	→	2
brutus	2	→	1 → 2
capitol	1	→	1
caesar	2	→	1 → 2
did	1	→	1
enact	1	→	1
hath	1	→	2
i	1	→	1
i'	1	→	1
it	1	→	2
julius	1	→	1
killed	1	→	1
let	1	→	2
me	1	→	1
noble	1	→	2
so	1	→	2
the	2	→	1 → 2
told	1	→	2
you	1	→	2
was	2	→	1 → 2
with	1	→	2

Prelucrare integrogari

Brutus *AND* ***Caesar*** but *NOT* ***Calpurnia***

(***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caesar***

Regasirea pe baza de rang

- In regasirea Booleana documentele fie se potrivesc fie nu
- Utilizatorul trebuie sa introduca un query Boolean (uneori dificil)
- Numarul de documente gasite f mare sau 0
- **Regasirea bazata pe rang (ranked retrieval systems)** – se produce un set ordonat de documente relevante pentru query
- Nu mai folosesc un limbaj de interogare ci interogare in limbaj natural
- Ideea de baza – **frecventa termenilor (Term Frequency)** = numarul de aparitii ale unui termen in document
- Modele statistice bazate pe contoare de cuvinte

Regasirea pe baza de rang

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Term frequency (tf)

- **Frecventa termenilor (Term Frequency)**
- $f_{t,d}$ = numarul de aparitii ale termenului **t** in documentul **d**
- Relevanta unui document nu creste proportional cu frecventa termenilor (un document cu 10 aparitii a lui t este mai relevant decat un document cu 1 aparitie, dar nu de 10 ori mai relevant)
- Se defineste varianta logaritmica a **term frequency**

$$\mathbf{tf_{t,d}} = \mathbf{1 + \log_{10} f_{t,d}} \quad \text{daca } f_{t,d} > 0$$

$$\mathbf{0} \text{ in caz contrar}$$

Term frequency (tf)

- **Frecventa augmentata** – impartita la frecventa cuvantului care apare cel mai des in document

$$\mathbf{tf_{t,d} = 0.5 + 0.5 * f_{t,d} / max(f_{t',d} : t' \in d)}$$

- Scorul pt termenul **t** pt query **q** si document **d**

$$\mathbf{Scor = \sum_{t \in q \cap d} tf_{t,d}}$$

Document frequency si idf

- Aparitia in document a unor cuvinte rare este mai relevanta decat cea a cuvintelor comune
- **Frecventa in documente** = numarul de documente in care apare **t**
- **d_{ft} – document frequency pt t**
- Este o masura inversa a cat de informativ este documentul si $\mathbf{d_{ft}} \leq N$, cu N=|D|, nr total de documente din corpusul D
- **Inverse document frequency** pt **t**

$$\mathbf{idf_{t,D} = \log_{10} (N/df_t)}$$

Inverse document frequency

N = 1 milion

$$\mathbf{idf_{t,D} = log_{10} (N/df_t)}$$

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

idf - *cata informatie ofera un cuvant*
scade ponderea cuvintelor care apar frecvent
in documente si creste ponderea celor care apar rar

tf si idf

$$tf_{t,d} = 1 + \log_{10} t_{f,d}$$

$$idf_{t,D} = \log_{10} (N/df_t)$$

$$\mathbf{tfidf_{t,d,D} = tf_{t,d} * idf_{t,D}}$$

$\mathbf{tfidf_{t,d,D}}$

- Creste odata cu numarul de aparitii in document
- Creste odata cu raritatea cuvintelor in colectie
- O valoare mare a **$\mathbf{tfidf_{t,d,D}}$** se obtine pentru o frecventa mare a cuvantului in document si o valoare mica a frecventei cuvantului in colectia de documente (se incearca eliminarea cuvintelor comune)

Matrice de ponderi

Antony	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Fiecare document este reprezentat printr-un vector cu valori reale – ponderile **tfidf**

tf si idf

Ponderea (weight) a unui termen este

$$w_{t,d} = \log(1 + tf_{t,d}) \times \log_{10}(N / df_t)$$

Cea mai cunoscuta formula in IR

Creste cu nr de aparitii in document

Creste cu raritatea cuventelor in colectie

Exemplu sistemul Lucene

TF(q_i,d_j) pt **N** documente – nr de aparitii $\mathbf{q_i}$ in documentul $\mathbf{d_j}$

DF(q_i) – **Document frequency counts** – nr de documente care contin cuvantul $\mathbf{q_i}$

Fiind dat un document $\mathbf{d_j}$ si un query cu cuvintele $\mathbf{q_{1:N}}$ avem

$$BM25(d_j, q_{1:N}) = \sum_{i=1}^{N} IDF(q_i) * \frac{TF(q_i, d_j) * (k+1)}{TF(q_i, d_j) + k(1 - b + b\frac{|d_j|}{L})}$$

$|d_j|$ - nr cuvinte din documentul d_j

$L = \Sigma_i |d_i| / N$ – lungimea medie a documentelor din corpus

k, b – determinati prin validare incrucisata, valori tipice:

k=2.0, b=0.75

$$IDF(q_i) = \log \frac{N - DF(q_i) + 0.5}{DF(q_i) + 0.5}$$

Regasirea informatiei

- Dificil de aplicat tfidf fiecarui document din corpus
- **Hit list** = index creat anterior care refera pentru fiecare cuvant din vocabular documentele ce contin acel cuvant
- Pt un query, se face intersectia intre *hit list* si cuvintele din query si se face cautarea numai pe aceasta intersectie
- tfidf – model care trateaza cuvintele ca fiind independente
- **Imbunatatiri**
  - Corelatii
  - Cuvinte derivate, omonime

Regasirea informatiei

- Page Rank (Brin and Larry Page 1998 Google)
- In loc sa caute documente cu masuri IR cat mai mari, dau prioritate documentelor de calitate
- Paginile cu multe *in-links* au scor mare
- Page Rank – o distributie de probabilitate – persoana navigheaza pe Web sa ajunga la o anumita pagina

$$PR(p) = \sum_{ini \in Bp} \frac{PR(in_i)}{C(in_i)}$$

- PR(p) – rangul paginii p
- Bp – multimea paginilor care au link la pagina ***p***
- in_i – paginile care au link la p (paginile din Bp)
- $C(in_i)$ – nr de *out-links* in pagina in_i
- Ex: A, B->A,C, C-> A, D-> A,B,C PR(A) = PR(B)/2 + PR(C)/1 + PR(D)/3

Regasirea informatiei

$$PR(p) = \frac{1-d}{N} + d\sum_{ini \in Bp} \frac{PR(in_i)}{C(in_i)}$$

- PR(p) – rangul paginii p
- Bp – multimea paginilor care au link la pagina ***p***
- in_i – paginile care au link la p (paginile din Bp)
- $C(in_i)$ – nr de *out-links* in pagina in_i
- N – nr total de pagini in corpus
- d – damping factor – probabilitatea ca sa ramana pe aceeasi pagina

- Calculat iterativ – se incepe cu paginile cu PR(p,0)=1/N si itereaza actualizand rangurile pana la convergenta

$$PR(p,t+1) = \frac{1-d}{N} + d\sum_{i} \frac{PR(in_i,t)}{C(in_i)}$$

4 Extragerea informatiei

- **Sisteme IE scop:**
- Localizeaza si extrage parti relevante dintr-un text
- Achizitia cunostintelor prin analiza unui text cu focalizare pe aparitia unei clase particulare de obiecte si a relatiilor dintre aceste obiecte
- Produce o reprezentare structurata
  - Relatii (intr-o bd)
  - Baza de cunostinte, ontologii

Extragerea informatiei

- Exemple tipice
  - extragerea din pagini web a adreselor (cu campuri strada, nr, etc.)
  - meteo – temp, vat, precipitatii, etc.

Capitala Iranului este orasul Teheran

capitala(Iran, teheran)

- De ce este dificil?
- Ex: Pretul unui produs (un interval, sau was $100 now $120, offers in the high 700s..)

Extragerea informatiei

- Arhitectura tipica (constantono, 1997)

Named Entity Recognition (NER)

- Gasirea si clasificarea entitatilor nume, de ex nume persoane, organizatii, locatii (orase) sau expresii numerice de ex timp, data, bani, procent etc.
- **Viorel Salvador Caragea**, fost șef al **Inspectoratului Județean de Poliție Gorj**, a demisionat din **USR**, după ce cu numai o zi înainte filiala județeană a **USR** i-a acceptat adeziunea. (sursa Digi24)
- Mazare fina congelata 450gr **5,99** ron

Evaluarea rezultatului

- **Precizie (precision) – procentul de entitati selectate care sunt corecte**
- **Relevanta (recall) – procentul de entitati corecte care sunt selectate**
- 100 documente din care selectez entitate nume companie XX

	Corecte	Incorecte
Selectate	30	20
Neselectatet	10	40

- **Precision** = 30/(30+20) = 0.6
- **Recall** = 30/(30+10) = 0.75

Evaluarea rezultatului

- De obicei o masura combinata (medie armonica)

$$F = \frac{1}{\alpha \frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}$$

- Mai frecvent se utilizeaza cu β=1

F = 2PR/(P+R)

Abordari pentru NER

- Expresii regulate (automate finite)
- ML cu modele secventiale
  - Naive Bayes
  - Discriminative: modele Maxent
- Modele probabiliste
  - HMM
  - CRFs
- Utilizate si in combinatie cu POS tagging, analiza sintactica (NP, VP, PP) sau categorii semantice (din WordNet de ex)

NER cu expresii regulate

- **Atomate finite**

Templates

- Extrage informatii relevante unui obiect – valorile unor atribute predefinite
- Se defineste un template (pattern) pentru fiecare atribut de extras
- Template-ul este definit de un automat finit (expresii regulate)
- Template – prefix regex, target regex, postfix regex
- price prefix regex **[€][0-9]*[.][0-9][0-9]** postfix regex

NER cu expresii regulate

- Daca template-ul se potriveste 1 data – extrage target regex
- Daca nu se potriveste de loc – atribut lipsa
- Daca se potriveste de mai multe ori – prioritate, mai multe versiuni de template (prefix regex de ex)
- Intereseaza in special **Recall**

NER cu reguli de productie

- Ce persoana are ce pozitie intr-o organizatie

  [person], [office] of [org]

  Ian Flex, leader of Geen movement

  [org] (named, appointed etc.) [person] Prep [office]

  NATO appointed Mircea Geoana as Deputy Secretary General

- Determinarea locatiei unei organizatii

  [org] in [loc]

  NATO headquarters in Brussels

  [org] [loc] (division, branch, headquarters, etc.)

  Google branch in Vancouver

ML cu modele secventiale

- Antrenare
  - Colecteaza set de documente reprezentative
  - Eticheteaza fiecare token cu clasa entitatii
  - Extrage atribute semnificative pentru clasa
  - Antreneaza modelul pt a prezice clase

- Testare
  - Colecteaza setul de documente de test
  - Ruleaza recunoasterea
  - Evalueaza reultatele (R, P)

ML cu modele secventiale

- De ex ”Named Entity Recognition and Classification with Scikit-Learn”
- Corpus adnotat cu IOB si POS tags
- IOB tags = Inside, Outside, Beginning
  - I- prefix - inaintea unui tag indica ca tagul este in interiorul unui chunck
  - B- prefix - inaintea unui tag indica ca un tag este la inceputul unui chunk.
  - O tag - indica ca un token nu apartine unui chunk (outside).

ML cu modele secventiale

- **Atribute ale entitatilor**

  geo = Geographical Entity

  org = Organization

  per = Person

  gpe = Geopolitical Entity

  tim = Time indicator

  art = Artifact

  eve = Event

  nat = Natural Phenomenon

Sentence #	Word	POS	Tag
Sentence: 1	Thousands	NNS	O
	of	IN	O
	demonstrators	NNS	O
	have	VBP	O
	marched	VBN	O
	through	IN	O
	London	NNP	B-geo
	to	TO	O
	protest	VB	O
	the	DT	O
	war	NN	O
	in	IN	O
	Iraq	NNP	B-geo
	and	CC	O
	demand	VB	O
	the	DT	O
	withdrawal	NN	O
	of	IN	O
	British	JJ	B-gpe
	troops	NNS	O
	from	IN	O
	that	DT	O
	country	NN	O
	.	.	O
Sentence: 2	Families	NNS	O
	of	IN	O
	soldiers	NNS	O

IR cu IE

- Anumite sisteme combina Information retrieval cu information extraction

5 POS tagging

- **Part-of-speech tagging** – un cuvant sau un grup de cuvinte sunt etichetate cu o categorie gramaticala in functie de semnificatie si de context (cuvinte adiacente, fraza, paragraf).
- Substantiv, adj, etc
- Gen, acord, forme infelxionate etc
- Engleza – de la 50 la 150 PoS
- Metode: HMM, Logistic Regression, NN

POS tagging

Penn

Tree bank

Tag	Word	Description	Tag	Word	Description
CC	*and*	Coordinating conjunction	PRP$	*your*	Possessive pronoun
CD	*three*	Cardinal number	RB	*quickly*	Adverb
DT	*the*	Determiner	RBR	*quicker*	Adverb, comparative
EX	*there*	Existential there	RBS	*quickest*	Adverb, superlative
FW	*per se*	Foreign word	RP	*off*	Particle
IN	*of*	Preposition	SYM	+	Symbol
JJ	*purple*	Adjective	TO	*to*	to
JJR	*better*	Adjective, comparative	UH	*eureka*	Interjection
JJS	*best*	Adjective, superlative	VB	*talk*	Verb, base form
LS	*1*	List item marker	VBD	*talked*	Verb, past tense
MD	*should*	Modal	VBG	*talking*	Verb, gerund
NN	*kitten*	Noun, singular or mass	VBN	*talked*	Verb, past participle
NNS	*kittens*	Noun, plural	VBP	*talk*	Verb, non-3rd-sing
NNP	*Ali*	Proper noun, singular	VBZ	*talks*	Verb, 3rd-sing
NNPS	*Fords*	Proper noun, plural	WDT	*which*	Wh-determiner
PDT	*all*	Predeterminer	WP	*who*	Wh-pronoun
POS	*'s*	Possessive ending	WP$	*whose*	Possessive wh-pronoun
PRP	*you*	Personal pronoun	WRB	*where*	Wh-adverb
$	$	Dollar sign	#	#	Pound sign
“	‘	Left quote	”	’	Right quote
(	[	Left parenthesis	)	]	Right parenthesis
,	,	Comma	.	!	Sentence end
:	;	Mid-sentence punctuation			

POS tagging

HMM - Hidden Markov Models

Observation space	$O_t \in \{x_1, x_2, \ldots, x_K\}$
Hidden states	$S_t \in \{S_1, \ldots, S_M\}$

POS tagging

$P(C_t|C_{t-1})$

$P(W_t|C_t)$

Observation space	$O_t \in \{x_1, x_2, \ldots, x_K\}$
Hidden states	$S_t \in \{S_1, \ldots, S_M\}$

6 Word Embedding

- Reprezentare a vocabularului care surprinde contextul cuvintelor – cuvinte care sunt legate/similare dpv sintactic (mic, frumos), sematic (pisica, pisoi), topica (insorit, ploios), sentiment (interesant, plictisitor)
- **Asocierea sematica** – co-aparitie in acelasi context (albina si miere)
- **Co-aparitie de ordinul I** (relatii sintagmatice – albina miere)
- **Co-aparitei de ordinul II** (relatii paradigmatice sau de similaritate – albina bondar)
- Am putea reprezenta ca one-hot-vector sau ca n-grame de cuvinte – vectori f mari si rari

Word Embedding

- Am dori vectori de dimensiuni mai mici (de ex cateva sute), care pot fi determinati automat

Matrice de co-aparitie

- Cum se calculeaza valorile din vector
- Pointwise Mutual Information

  $$PMI(a,b) = \log (p(a,b) / p(a)\, p(b))$$

- Calcul dificil

Cosine similarity

Se determina numarul de aparitii ale cuvintelor intr-o fereastra glisanta in text

Se calculeaza similaritatea cosina

Cosine similarity

Word Embedding

- Am dori vectori de dimensiuni mai mici (de ex cateva sute), care pot fi invatati automat

$$
\begin{aligned}
\text{“aardvark”} &= [-0.7, +0.2, -3.2, \ldots] \\
\text{“abacus”} &= [+0.5, +0.9, -1.3, \ldots] \\
&\ldots \\
\text{“zyzzyva”} &= [-0.1, +0.8, -0.4, \ldots].
\end{aligned}
$$

Vectori word embedding calculati cu alg. GloVe antrenat pe 6 miliarde de cuvinte din texte (sursa AIMA)

Word Embedding

A	B	C	D = C + (B − A)	Relationship
Athens	Greece	Oslo	Norway	*Capital*
Astana	Kazakhstan	Harare	Zimbabwe	*Capital*
Angola	kwanza	Iran	rial	*Currency*
copper	Cu	gold	Au	*Atomic Symbol*
Microsoft	Windows	Google	Android	*Operating System*
New York	New York Times	Baltimore	Baltimore Sun	*Newspaper*
Berlusconi	Silvio	Obama	Barack	*First name*
Switzerland	Swiss	Cambodia	Cambodian	*Nationality*
Einstein	scientist	Picasso	painter	*Occupation*
brother	sister	grandson	granddaughter	*Family Relation*
Chicago	Illinois	Stockton	California	*State*
possibly	impossibly	ethical	unethical	*Negative*
mouse	mice	dollar	dollars	*Plural*
easy	easiest	lucky	luckiest	*Superlative*
walking	walked	swimming	swam	*Past tense*

Exista dictionare de vectori preantrenati: WORD2VEC, GloVe(Global Vectors), FASTTEXT

Word Embedding

Retea neurala pt word embedding si POS tagging

Slide-uri pct 3 bazate pe
CS 276 / LING 286: Information Retrieval and Web Search

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 12

Modelul logicii simbolice

- Reprezentarea cunostintelor in logica simbolica
- Logica propozitiilor
- Logica cu predicate de ordinul I
- Demonstrarea teoremelor prin respingere rezolutiva
- Legatura cu limbajul Prolog

1. Reprezentarea cunostintelor in LS

- Logica – avantaje
- Puterea de reprezentare a diverselor logici simbolice
- Conceptualizare + exprimarea in limbaj
- Limbaj formal: sintaxa, semantica
- Reguli de inferenta

2. Logica propozitiilor

2.1 Sintaxa

- Alfabet
- O **formula bine formata** in calculul propozitional se defineste recursiv astfel:

(1) Un atom este o formula bine formata

(2) Daca P este formula bine formata, atunci ~P este formula bine formata.

(3) Daca P si Q sint formule bine formate atunci $P \wedge Q$, $P \vee Q$, $P \rightarrow Q$ si $P \leftrightarrow Q$ sint formule bine formate.

(4) Multimea formulelor bine formate este generata prin aplicarea repetata a regulilor (1)..(3) de un numar finit de ori.

2.2 Semantica

- Interpretare
- *Functia de evaluare a unei formule*
- Proprietatile fbf
  - Valida/tautologie
  - Realizabila
  - Inconsistenta
  - Formule echivalente

Semantica - cont

- O formula F este o *consecinta logica a unei formule* P daca F are valoarea adevarat in toate interpretarile in care P are valoarea adevarat.
- O formula F este *consecinta logica a unei multimi de formule* $P_1,\ldots P_n$ daca formula F este adevarata in toate interpretarile in care $P_1,\ldots P_n$ sunt adevarate.
- Consecinta logica se noteaza $P_1,\ldots P_n \Rightarrow F$.
- **Teorema**. Formula F este consecinta logica a unei multimi de formule $P_1,\ldots P_n$ daca formula $P_1 \wedge \ldots \wedge P_n \rightarrow F$ este valida.
- **Teorema**. Formula F este consecinta logica a unei multimi de formule $P_1,\ldots P_n$ daca formula $P_1 \wedge \ldots \wedge P_n \wedge \sim F$ este inconsistenta.

Legi de echivalenta

Idempotenta	$P \vee P \equiv P$	$P \wedge P \equiv P$	
Asociativitate	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	$(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$	
Comutativitate	$P \vee Q \equiv Q \vee P$	$P \wedge Q \equiv Q \wedge P$	$P \leftrightarrow Q \equiv Q \leftrightarrow P$
Distributivitate	$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	
De Morgan	$\sim (P \vee Q) \equiv \sim P \wedge \sim Q$	$\sim (P \wedge Q) \equiv \sim P \vee \sim Q$	
Eliminarea implicatiei	$P \rightarrow Q \equiv \sim P \vee Q$		
Eliminarea implicatiei duble	$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$		

2.3 Obtinerea de noi cunostinte

- Conceptualizare
- Reprezentare in limbaj
- Teoria modelului

  $KB \models_S x$
- Teoria demonstratiei

  $KB \vdash_{\mathcal{R}} x$
- Logici monotone
- Logici nemonotone

3.4 Reguli de inferenta

- *Modus Ponens* $\dfrac{\begin{array}{c} P \\ P \rightarrow Q \end{array}}{Q}$
- *Substitutia*
- *Regula inlantuirii* $\dfrac{\begin{array}{c} P \rightarrow Q \\ Q \rightarrow R \end{array}}{P \rightarrow R}$
- *Regula introducerii conjunctiei* $\dfrac{\begin{array}{c} P \\ Q \end{array}}{P \wedge Q}$
- *Regula transpozitiei* $\dfrac{P \rightarrow Q}{\sim Q \rightarrow \sim P}$

Exemplu

- *Mihai are bani*
- *Masina este alba*
- *Masina este frumoasa*
- *Daca masina este alba sau masina este frumoasa si Mihai are bani atunci Mihai pleaca in vacanta*
- B
- A
- F
- $(A \vee F) \wedge B \rightarrow C$

3. Logica cu predicate de ordinul I

3.1 Sintaxa

Fie *D* un domeniu de valori. Un *termen* se defineste astfel:

- (1) O constanta este un termen cu valoare fixa apartinand domeniului *D*.
- (2) O variabila este un termen ce poate primi valori diferite din domeniul *D*.
- (3) Daca *f* este o functie de *n* argumente si $t_1,..t_n$ sint termeni, atunci $f(t_1,..t_n)$ este termen.
- (4) Toti termenii sunt generati prin aplicarea regulilor (1)…(3).

Sintaxa LP - cont

- Predicat de aritate n
- Atom sau formula atomica.
- Literal

O *formula bine formata* in logica cu predicate de ordinul I se defineste astfel:

(1) Un atom este o formula bine formata

(2) Daca P[x] este fbf, atunci $\sim$P[x] este fbf.

(3) Daca P[x] si Q [x] sunt fbf atunci P[x]$\wedge$Q[x], P[x] $\vee$Q[x], P$\rightarrow$Q si P$\leftrightarrow$Q sunt fbf.

(4) Daca P[x] este fbf atunci $\forall$x P[x], $\exists$x P[x] sunt fbf.

(5) Multimea formulelor bine formate este generata prin aplicarea repetata a regulilor (1)..(4) de un numar finit de ori.

Sintaxa pe scurt

FNC, FND

- O formula bine formata este in *forma normala conjunctiva*, pe scurt FNC, daca formula are forma

$$F_1 \wedge \ldots \wedge F_n,$$

unde este F_i , i=1,n sunt formule formate dintr-o disjunctie de literali ($L_{i1} \vee \ldots \vee L_{im}$).

- O formula bine formata este in *forma normala disjunctiva*, pe scurt FND, daca formula are forma ,

$$F_1 \vee \ldots \vee F_n,$$

unde F_i , i=1,n sunt formule formate dintr-o conjunctie de literali ($L_{i1} \wedge \ldots \wedge L_{im}$)

3.2 Semantica LP

- *Interpretarea* unei formule F in logica cu predicate de ordinul I consta in fixarea unui domeniu de valori nevid D si a unei asignari de valori pentru fiecare constanta, functie si predicat ce apar in F astfel:
- (1) Fiecarei constante i se asociaza un element din D.
- (2) Fiecarei functii f, de aritate n, i se asociaza o corespondenta $D^n \rightarrow D$, unde

$$D^n = \{(x_1,\ldots,x_n) | x_1 \in D,\ldots,x_n \in D\}$$

- (3) Fiecarui predicat de aritate n, i se asociaza o corespondenta $P: D^n \rightarrow \{\mathbf{a},\mathbf{f}\}$

Interpretare I

$$(\forall x)(((A(a,x) \vee B(f(x))) \wedge C(x)) \rightarrow D(x))$$

$D=\{1,2\}$

a
2

f(1)	f(2)
2	1

A(2,1)	A(2,2)	B(1)	B(2)	C(1)	C(2)	D(1)	D(2)
a	**f**	**a**	**f**	**a**	**f**	**f**	**a**

X=1 $\quad ((\mathbf{a} \vee \mathbf{f}) \wedge \mathbf{a}) \rightarrow \mathbf{f}$

X=2 $\quad ((\mathbf{f} \vee \mathbf{a}) \wedge \mathbf{f}) \rightarrow \mathbf{a}$

3.3 Proprietatile fbf in LP

- Valida/tautologie
- Realizabila
- Inconsistenta
- Echivalente

- F - *consecinta logica a unei formule* P
- F - *consecinta logica a unei multimi de formule* $P_1,\ldots P_n$
- **Teorema**. Formula F este consecinta logica a unei multimi de formule $P_1,\ldots P_n$ daca formula $P_1 \wedge \ldots \wedge P_n \rightarrow F$ este valida.
- **Teorema**. Formula F este consecinta logica a unei multimi de formule $P_1,\ldots P_n$ daca formula $P_1 \wedge \ldots \wedge P_n \wedge \sim F$ este inconsistenta.

Echivalenta cuantificatorilor

$(Qx)F[x] \vee G \equiv (Qx)(F[x] \vee G)$	$(Qx)F[x] \wedge G \equiv (Qx)(F[x] \wedge G)$
$\sim ((\forall x)F[x]) \equiv (\exists x)(\sim F[x])$	$\sim ((\exists x)F[x]) \equiv (\forall x)(\sim F[x])$
$(\forall x)F[x] \wedge (\forall x)H[x] \equiv (\forall x)(F[x] \wedge H[x])$	$(\exists x)F[x] \vee (\exists x)H[x] \equiv (\exists x)(F[x] \vee H[x])$
$(Q_1x)F[x] \wedge (Q_2x)H[x] \equiv (Q_1x)(Q_2z)(F[x] \wedge H[z])$	$(Q_1x)F[x] \vee (Q_2x)H[x] \equiv (Q_1x)(Q_2z)(F[x] \vee H[z])$

Exemple

- Exista un mar rosu
- Toate merele sunt rosii
- Toate obiectele sunt mere rosii

Toate ciupercile purpurii sunt otravitoare

- $\forall x\ (\text{Purpuriu}(x) \wedge \text{Ciuperca}(x)) \Rightarrow \text{Otravitor}(x)$
- $\forall x\ \text{Purpuriu}(x) \Rightarrow (\text{Ciuperca}(x) \Rightarrow \text{Otravitor}(x))$
- $\forall x\ \text{Ciuperca}\ (x) \Rightarrow (\text{Purpuriu}\ (x) \Rightarrow \text{Otravitor}(x))$

$(\forall x)(\exists y)\ \text{iubeste}(x,y)$

$(\exists y)(\forall x)\ \text{iubeste}(x,y)$

3.4. Reguli de inferenta in LP

- Modus Ponens (MP)

$$\frac{P(a) \quad (\forall x)(P(x) \rightarrow Q(x))}{Q(a)}$$

- Substitutia
- Regula inlantuirii
- Transpozitia
- Eliminarea conjunctiei (ElimC)
- Introducerea conjunctiei (IntrC)
- Instantierea universala (InstU)
- Instantierea existentiala (InstE)
- Rezolutia

Exemplu

- Caii sunt mai rapizi decat cainii si exista un ogar care este mai rapid decat orice iepure. Se stie ca Harry este un cal si ca Ralph este un iepure. Sa se demonstreze faptul ca Harry este mai rapid decat Ralph.
- Cal(x) Ogar(y)
- Caine(y) Iepure(z)
- MaiRapid(y,z)

$$\forall x\ \forall y\ \mathbf{Cal(x)} \wedge \mathbf{Caine(y)} \rightarrow \mathbf{MaiRapid(x,y)}$$

$$\exists y\ \mathbf{Ogar(y)} \wedge (\forall z\ \mathbf{Iepure(z)} \rightarrow \mathbf{MaiRapid(y,z)})$$

$$\mathbf{Cal(Harry)}$$

$$\mathbf{Iepure(Ralph)}$$

$$\forall y\ \mathbf{Ogar(y)} \rightarrow \mathbf{Caine(y)}$$

$$\forall x\ \forall y\ \forall z\ \mathbf{MaiRapid(x,y)} \wedge \mathbf{MaiRapid(y,z)} \rightarrow \mathbf{MaiRapid(x,z)}$$

Exemplu de demonstrare

- **Teorema: MaiRapid(Harry, Ralph) ?**
- **Demonstrare folosind reguli de inferenta**

1. $\forall x\ \forall y\ Cal(x) \wedge Caine(y) \rightarrow MaiRapid(x,y)$
2. $\exists y\ Ogar(y) \wedge (\forall z\ Iepure(z) \rightarrow MaiRapid(y,z))$
3. $\forall y\ Ogar(y) \rightarrow Caine(y)$
4. $\forall x \forall y \forall z\ MaiRapid(x,y) \wedge MaiRapid(y,z) \rightarrow MaiRapid(x,z)$
5. $Cal(Harry)$
6. $Iepure(Ralph)$

---

7. $Ogar(Greg) \wedge (\forall z\ Iepure(z) \Rightarrow MaiRapid(Greg,z))$ — 2, InstE
8. $Ogar(Greg)$ — 7, ElimC
9. $\forall z\ Iepure(z) \Rightarrow MaiRapid(Greg,z))$ — 7, ElimC

Exemplu de demonstrare - cont

10.	Iepure(Ralph) $\rightarrow$ MaiRapid(Greg,Ralph)	9, InstU
11.	MaiRapid(Greg,Ralph)	6,10, MP
12.	Ogar(Greg) $\rightarrow$ Caine(Greg)	3, InstU
13.	Caine(Greg)	12, 8, MP
14.	Cal(Harry) $\wedge$ Caine(Greg) $\rightarrow$ MaiRapid(Harry, Greg)	1, InstU
15.	Cal(Harry) $\wedge$ Caine(Greg)	5, 13, IntrC
16.	MaiRapid(Harry, Greg)	14, 15, MP
17.	MaiRapid(Harry, Greg) $\wedge$ MaiRapid(Greg, Ralph) $\rightarrow$ MaiRapid(Harry,Ralph)	4, InstU
18.	MaiRapid(Harry, Greg) $\wedge$ MaiRapid(Greg, Ralph)	16, 11, IntrC
19.	MaiRapid(Harry,Ralph)	17, 18, MP

4. Demonstrarea teoremelor prin respingere rezolutiva

4.1 Forma standard (forma clauzala)

- Clauza
- Clauza Horn
- Clauza Horn definita
- Clauza vida
- Forme de notare
  - Forma clauzala $\sim A(x) \vee B(x)$
  - Forma clauzala multime $\{\sim A(x), B(x)\}$
  - Forma consecinta (Gentzer) $A(x) \Rightarrow B(x)$
  - Forma Prolog B(x) :- A(x)

Transformare in forma clauzala

Elimina implicatiile (simple sau duble)

Aduce negatiile in fata atomilor

Muta toti cuantificatorii in fata formulei (redenumire variabile daca este necesar)

Skolemizare

Transforma in FNC

Elimina cuantificatorii universali (se considera implicit in clauze)

Elimina conjunctii – set de clauze

Normalizeaza variabilele (daca este cazul)

Transformare in forma clauzala - exemplu

$$\forall x\ \forall y\ (\mathrm{Horse}(x) \wedge \mathrm{Dog}(y) \ \rightarrow\ \mathrm{Faster}(x, y))$$

$$\forall x\ \forall y\ (\sim(\mathrm{Horse}(x) \wedge \mathrm{Dog}(y)) \vee \mathrm{Faster}(x, y))$$

$$\forall x\ \forall y\ (\sim\mathrm{Horse}(x) \vee \sim\mathrm{Dog}(y) \vee \mathrm{Faster}(x, y))$$

$$\sim\mathrm{Horse}(x) \vee \sim\mathrm{Dog}(y) \vee \mathrm{Faster}(x, y)$$

$$R = \{\sim\mathrm{Horse}(x) \vee \sim\mathrm{Dog}(y) \vee \mathrm{Faster}(x, y)\}$$

Transformare in forma clauzala - exemplu

$$\exists y\ (\mathrm{Greyhound}(y) \wedge (\forall z\ (\mathrm{Rabbit}(z) \rightarrow \mathrm{Faster}(y, z))))$$

$$\exists y\ (\mathrm{Greyhound}(y) \wedge (\forall z(\sim \mathrm{Rabbit}(z) \vee \mathrm{Faster}(y, z)))$$

$$\exists y \forall z\ (\mathrm{Greyhound}(y) \wedge (\sim \mathrm{Rabbit}(z) \vee \mathrm{Faster}(y, z)))$$

$$\forall z\ (\mathrm{Greyhound}(\mathrm{Greg}) \wedge (\sim \mathrm{Rabbit}(z) \vee \mathrm{Faster}(\mathrm{Greg}, z)))$$

$$\mathrm{Greyhound}(\mathrm{Greg}) \wedge (\sim \mathrm{Rabbit}(z) \vee \mathrm{Faster}(\mathrm{Greg}, z))$$

$$R = \{\mathrm{Greyhound}(\mathrm{Greg}), \sim \mathrm{Rabbit}(z) \vee \mathrm{Faster}(\mathrm{Greg}, z)\}$$

4.2 Unificarea expresiilor

- Substitutie α
- Unificator
- Cel mai general unificator β
- Expresie
- Algoritm de unificare

Algoritm Unifica(E1,E2): Unificarea expresiilor

1. **daca** E1 si E2 sunt constante
   atunci
   1.1. **daca** E1=E2
   atunci intoarce { }
   1.2. **intoarce** INSUCCES
2. **daca** E1 este variabila **atunci**
   2.1 **daca** E1 apare in E2 **atunci intoarce** INSUCCES
   altfel intoarce {E1/E2}
3. **daca** E2 este variabila **atunci**
   3.1 **daca** E2 apare in E1 **atunci intoarce** INSUCCES
   altfel intoarce {E2/E1}
4. **daca** E1= { } si E2 = { } **atunci intoarce** SUCCES
5. **daca** E1= { } sau E2 = { } **atunci intoarce** INSUCCES

6. **daca** E1=simb(t_{11},…,t_{1n}) **si** E2=simb(t_{21},…t_{2n}) **atunci**

   6.1 $X \leftarrow t_{11}$, $Y \leftarrow t_{21}$

   6.2 $Rest_1 \leftarrow t_{12},\ldots,t_{1n}$, $Rest_2 \leftarrow t_{22},\ldots t_{2n}$

   6.3 $\alpha 1 \leftarrow$ Unifica(X,Y)

   6.4 **daca** $\alpha 1$= INSUCCES

   atunci intoarce INSUCCES

   6.5 $G_1 \leftarrow$ aplica($\alpha 1$,$Rest_1$), $G_2 \leftarrow$ aplica($\alpha 1$,$Rest_2$)

   6.6 $\alpha 2 \leftarrow$ Unifica(G_1, $G2_2$)

   6.7 **daca** $\alpha 2$= INSUCCES

   atunci intoarce INSUCCES

   altfel intoarce ($\alpha 1$, $\alpha 2$)

7. **intoarce** INSUCCES

sfarsit.

4.3 Rezolutia in logica propozitiilor

- Rezolvent
- Clauze care rezolva
- Principiul demonstratiei
- Arbore/graf de demonstrare

Algoritm: Respingerea prin rezolutie in logica propozitionala.

1. Converteste setul de axiome A in forma clauzala si obtine multimea de clauze S
2. Neaga teorema, transforma teorema negata in forma clauzala si adauga rezultatul la S

3. **repeta**

3.1. Selecteaza o pereche de clauze C_1 si C_2 din S

3.2. Determina $R = Res(C_1,C_2)$

3.3. **daca** $R \neq \square$

atunci adauga R la S

pana $R = \square$ **sau** nu mai exista nici o pereche de clauze care rezolva

4. **daca** s-a obtinut clauza vida

atunci teorema este adevarata (este demonstrata)

5. **altfel** teorema este falsa

sfarsit.

- Comentarii
  - strategie
  - decidabilitate

4.4 Rezolutia in logica predicatelor

- Rezolvent binar
- Clauze care rezolva
- Factor al unei clauze
- Rezolvent

Algoritm: Respingerea prin rezolutie in logica predicatelor.

1. Converteste setul de axiome A in forma clauzala si obtine multimea de clauze S
2. Neaga teorema, transforma teorema negata in forma clauzala si adauga rezultatul la S
3. **repeta**
   3.1. Selecteaza o pereche de clauze C_1 si C_2 din S
   3.2. Determina R = {Res(C_1,C_2)}
   3.3. **daca** $\square \notin R$
   atunci reuneste R cu S

   pana s-a obtinut $\square$ **sau** nu mai exista nici o pereche de clauze care rezolva **sau** un efort a fost epuizat
4. **daca** s-a obtinut clauza vida

   atunci teorema este adevarata (este demonstrata)
5. **altfel daca** nu mai exista nici o pereche de clauze care rezolva

   atunci teorema este falsa

   altfel nu se stie

sfarsit.

- Comentarii
  - strategie
  - decidabilitate
  - completitudine

Exemplu

- Horses are faster than dogs and there is a greyhound that is faster than every rabbit. We know that Harry is a horse and that Ralph is a rabbit. Derive that Harry is faster than Ralph.

- Horse(x) Greyhound(y)
- Dog(y) Rabbit(z)
- Faster(y,z))

$$\forall x\ \forall y\ \mathbf{Horse(x)} \wedge \mathbf{Dog(y)} \rightarrow \mathbf{Faster(x,y)}$$

$$\exists y\ \mathbf{Greyhound(y)} \wedge (\forall z\ \mathbf{Rabbit(z)} \rightarrow \mathbf{Faster(y,z)})$$

$$\mathbf{Horse(Harry)}$$

$$\mathbf{Rabbit(Ralph)}$$

$$\forall y\ \mathbf{Greyhound(y)} \rightarrow \mathbf{Dog(y)}$$

$$\forall x\ \forall y\ \forall z\ \mathbf{Faster(x,y)} \wedge \mathbf{Faster(y,z)} \rightarrow \mathbf{Faster(x,z)}$$

A1. $\forall x\ \forall y\ (Horse(x) \wedge Dog(y) \rightarrow Faster(x,y))$

A2. $\exists y\ Greyhound(y) \wedge (\forall z\ Rabbit(z) \rightarrow Faster(y,z))$

A3. $Horse(Harry)$

A4. $Rabbit(Ralph)$

A5. $\forall y\ (Greyhound(y) \rightarrow Dog(y))$

A6. $\forall x\ \forall y\ \forall z\ (Faster(x,y) \wedge Faster(y,z) \rightarrow Faster(x,z))$

T $Faster(Harry,Ralph)$

C1. $\sim Horse(x) \vee \sim Dog(y) \vee Faster(x,y)$

C2. $Greyhound(Greg)$

C2' $\sim Rabbit(z) \vee Faster(Greg,z)$

C3. $Horse(Harry)$

C4. $Rabbit(Ralph)$

C5. $\sim Greyhound(y) \vee Dog(y)$

C6. $\sim Faster(x,y) \vee \sim Faster(y,z) \vee Faster(x,z)$

C7. $\sim Faster(Harry,Ralph)$

4.5 Strategii rezolutive

- *Strategia dezvoltarii pe latime*
- *Strategia multimii suport*
- *Strategia rezolutiei liniare*
- *Strategia rezolutiei liniare de intrare*
- *Strategia rezolutiei unitare*
- *Strategia eliminarii*

Strategia dezvoltarii pe latime

Strategia dezvoltarii pe latime

- S_0, L_0
- $L_{k+1} \leftarrow \{Res(C_i, C_j) | C_i \in L_k, C_j \in S_k\}$
- $S_{k+1} \leftarrow S_k \cup L_{k+1}$

  $k = 0,1,2,\ldots$

Strategia multimii suport

- S, T $\qquad S_0 = S \cup T, \quad S \cap T = \phi$
- $S \leftarrow S \cup \{Res(C_i, C_j) | C_i \in S, C_j \in T\}$

Strategia rezolutiei liniare

- $S, \qquad C_0 \in S$
- $C_1 \leftarrow \{Res(C_0, C_i) | C_0, C_i \in S\}$
- $C_{k+1} \leftarrow \{Res(C_k, C_i) | C_i \in \{C_{k-1}, C_{k-2}, ..\} \cup S\}$

  $k=1, 2, 3, \ldots$

Strategia rezolutiei liniare de intrare

- $S, \qquad C_0 \in S$
- $C_1 \leftarrow \{Res(C_0, C_i) | C_0, C_i \in S\}$
- $C_{k+1} \leftarrow \{Res(C_k, C_i) | C_i \in S\}$

  $k=1, 2, 3, \ldots$

Strategia rezolutiei unitare

Clauze subsumate

- O clauza C subsumeaza o clauza D daca si numai daca exista o substitutie α astfel incat $C\alpha \subseteq D$. D se numeste clauza subsumata.
- $C=P(x)$ $\quad\quad$ $D=P(a) \vee Q(a)$

Strategia eliminarii

Verificare C subsumeaza D

$D = L_1 \vee \ldots \vee L_m$, $\alpha = \{x_1/a_1, .. x_n/a_n\}$

$D\alpha = L_1\alpha \vee \ldots \vee L_m\alpha$, $\sim D\alpha = \sim L_1\alpha \wedge \ldots \wedge \sim L_m\alpha$

1. $W \leftarrow \{\sim L_1\alpha, \ldots, \sim L_m\alpha\}$
2. $k \leftarrow 0$, $U^0 \leftarrow \{C\}$
3. **daca** $\square \in U^k$ **atunci intoarce** SUCCES
4. $U^{k+1} \leftarrow \{rez(C_1, C_2) \mid C_1 \in U^k, C_2 \in W\}$
5. **daca** $U^{k+1} = \phi$ **atunci intoarce** INSUCCES
6. $k \leftarrow k+1$
7. **repeta** de la 3

sfarsit

5. Legatura cu limbajul Prolog

- R. Kowalski, A. Colmerauer - începutul anilor '70
- Corespondenta Logica cu Predicate si limbajul Prolog – ***clauze Horn definite***
- Structura logica vs structura de control si executie

5.1 Structura logica a limbajului Prolog

- Un scop **S** este **adevarat** intr-un program Prolog, adica *poate fi satisfacut* sau *derivă logic din program*, daca si numai daca:

1. exista o clauza **C** a programului;
2. exista o instanta **I** a clauzei **C** astfel incat:
   - antetul lui **I** sa fie identic cu cel al lui **S**;
   - toate scopurile din corpul lui **I** sunt adevarate, deci pot fi satisfacute

5.2 Structura de control si executie a limbajului Prolog

- Rezolutie liniara de intrare
- Ordinea faptelor si clauzelor
- Ordinea scopurilor in corpul clauzelor
- Backtracking

5.3 Prolog – exemplu

% parinte(IndividX, IndividY)

% stramos(IndividX, IndividZ)

parinte(vali, gelu).

parinte(ada, gelu).

parinte(ada, mia).

parinte(gelu, lina).

parinte(gelu, misu).

parinte(misu, roco).

str1(X, Z) :- parinte(X, Z).

str1(X, Z) :- parinte(X, Y), str1(Y, Z).

Prolog – exemplu

% Se schimba ordinea regulilor:

str2(X, Z) :- parinte(X, Y), str2(Y, Z).

str2(X, Z) :- parinte(X, Z).

% Se schimba ordinea scopurilor in prima varianta:

str3(X, Z) :- parinte(X, Z).

str3(X, Z) :- str3(X, Y), parinte(Y, Z).

% Se schimba atat ordinea regulilor, cat si ordinea scopurilor:

str4(X, Z) **:-** str4(X, Y), parinte(Y, Z).

str4(X, Z) **:-** parinte(X,Z).

Prolog - exemplu

?- str1(ada, misu).

yes

?- str2(ada, misu).

yes

?- str3(ada, misu).

yes

Prolog – exemplu

?- str4(ada, misu).

?- str3(mia, roco).

Inteligență Artificială

Universitatea Politehnica Bucuresti
Anul universitar 2021-2022

Adina Magda Florea

Curs nr. 13

Sisteme bazate pe reguli

- Reprezentarea prin reguli
- Structura SBR
- Ciclul de inferenta al unui sistem bazat pe reguli
- Strategia de control
- SBR cu inlantuire inainte
- SBR cu inlantuire inapoi
- Modelul cunoștințelor incerte din MYCIN – factori de certitudine

1. Reprezentarea prin reguli

- Reprezentare modulara a cunostintelor procedurale
- Cunostinte procedurale in maniera declarativa
- Permit atasare de actiuni (functii, proceduri)
- Se pot combina cu reprezentari structurate ale cunostintelor, de ex. ontologii
- Multiple limbaje bazate pe reguli

Limbaje bazate pe reguli

Precursori

- **OPS5 -** Official Production System
- **MYCIN**

Limbaje bazate pe reguli

Noi, moderne

- **CLIPS - C Language Integrated Production System**
- **Jess** – **Java Expert System Shell** - written in Java, superset of CLIPS
- **RuleML** – **Rule Markup Language** (Rule Markup Initiative)
- **SWRL - Semantic Web Rule Language**
- **Drools** - **Business Rules Management System** **https://www.drools.org/**
- **Legal XML** **http://www.legalxml.org/**
- **Declarative AI 2020** - Rules, Reasoning, Decisions, and Explanations https://2020.declarativeai.net/

Forme de reguli

- **Reguli de inferenta**
  - **if** conditie(i) **then** concluzie/actiune
- Utilizare
  - obtinerea de noi cunostinte
  - executarea actiunilor in functie de conditii
- **Reguli reactive**
  - **on** eveniment **if** conditie(i) **then** (executa) actiune
- Utilizare
  - triggere in SMBD
  - detectarea exceptiilor
  - reguli de integritate
- Se pot combina

2. Structura SBR

3. Ciclul de inferenta al unui sistem bazat pe reguli

- Identificare
- Selectie
- Executie

Ciclul de inferenta al unui sistem bazat pe reguli - cont

Algoritm: Functionarea unui sistem bazat pe reguli

1. ML ← Date de caz
2. **repeta**

   2.1. Executa identificare intre ML si BC si construieste multimea de conflicte a regulilor aplicabile (MC)

   2.2. Selecteaza o regula dupa un criteriu de selectie

   2.3. Aplica regula prin executia partii drepte a regulii

   pana nu mai sunt reguli aplicabile **sau**

   memoria de lucru satisface conditia de stare scop **sau**

   o cantitate predefinita de efort a fost epuizata

sfarsit.

4. Strategia de control

- Criteriile de selectie din MC
- Directia de aplicare a regulilor

4.1 Criteriile de selectie din MC

- Selectia primei reguli aplicabile
- Alegerea unei reguli din multimea de conflicte
  - Preferinte bazate pe natura regulilor

    Specificitate

    Momentului folosirii anterioare
  - Preferinte bazate pe obiectele identificate
  - Preferinte bazate pe natura starilor

Criteriile de selectie din MC - cont

- Utilizarea metaregulilor

daca o regula are conditiile A si B **si**
regula refera {nu refera} X
{ de loc/
numai in partea stinga/
numai in partea dreapta }
atunci regula va fi in special utila
{ probabil utila/
probabil inutila/
sigur inutila }

- Aplicarea tuturor regulilor din multimea de conflicte

4.2 Directia de aplicare a regulilor

(a) Inlantuire inainte

Continut initial al memoriei de lucru: A,E
Stare scop: D

$$A,E \xrightarrow{R3} A,E,B \xrightarrow{R1} A,E,B,C \xrightarrow{R2} A,E,B,C,D$$

(a)

R1: daca A&B atunci C

R2: daca C atunci D

R3: daca A atunci B

R4: daca A&E atunci C

(b) Inlantuire inapoi

Continut initial al memoriei de lucru: A,E
Stare scop: D

$$D? \xrightarrow{R2} C? \xrightarrow{R1} A?, B? \xrightarrow{R3} A?, E?$$

(a)

(b)

R1: daca A&B atunci C

R2: daca C atunci D

R3: daca A atunci B

R4: daca A&E atunci C

Algoritm: **Determinarea unei valori cu inlantuire inainte a regulilor (a)**

DetValoareD(A)

1. **daca** A are valoare
   atunci intoarce SUCCES
2. Construieste multimea de conflicte, MC, si initializeaza Efort
3. **daca** MC = {}
   atunci intoarce INSUCCES
4. Alege o regula R$\in$MC dupa un criteriu de selectie
5. Aplica regula R
   Adauga faptul din concluzia regulii R la ML (daca nu exista deja)
   sau executa actiunea din concluzie
6. **daca** A are valoare (A $\in$ML)
   atunci intoarce SUCCES
7. MC $\leftarrow$ MC - R
8. MC $\leftarrow$ MC $\cup$ Noile reguli aplicabile
9. Actualizeaza Efort
10. Daca Efort = {} atunci **intoarce** INSUCCES
11. **repeta** de la 3

sfarsit.

Algoritm: Determinarea unei valori cu inlantuirea inapoi a regulilor (b)

DetValoare(A)

1. Construieste multimea regulilor care refera A in concluzie, MC
2. **daca** MC = {}
   atunci
   2.1. Intreaba utilizatorul valoarea lui A
   2.2. **daca** se cunoaste valoarea lui A
   atunci depune in ML aceasta valoare
   altfel intoarce INSUCCES
3. **altfel**
   3.1. Alege o regula dupa un criteriu de selectie
   3.2. **pentru** fiecare ipoteza Ij, j=1,N, a premisei lui R **executa**
      3.2.1. Fie Aj atributul referit de ipoteza Ij
      3.2.2. **daca** Aj are valoare
      atunci atunci evalueaza adevarul ipotezei Ij
      3.2.3. **altfel**
         i. **daca** DetValoare(Aj) = SUCCES
         atunci evalueaza adevarul ipotezei Ij
         ii. **altfel** considera ipoteza Ij nesatisfacuta

3.3. **daca** toate ipotezele Ij, j=1,N sunt satisfacute

atunci depune in ML valoarea lui A dedusa pe baza concluziei R

4. **daca** A are valoare (in ML)

atunci intoarce SUCCES

5. **altfel**

5.1 MC ← MC – R

5.2 **repeta** de la 2

sfarsit.

1'. **daca** A este data primara

atunci

1'.1. Intreaba utilizatorul valoarea lui A

1'.2. **daca** se cunoaste valoarea lui A

atunci **-** depune in ML valoarea lui A

- intoarce SUCCES

1’.3. **altfel intoarce** INSUCCES

5. SBR cu inlantuire inainte

Familia OPS5

- OPS5
- ART
- CLIPS
- Jess
- Familia Web Rule languages
  - In special RuleML si SWRL (OWL + Rule language)
  - Interoperabilitatea regulilor

- Drools

SBR cu inlantuire inainte

- Ciclul recunoastere-actiune:
  - Match
  - Select
  - Act
- WME = working memory element
  - Identificat printr-un "time tag"
- Instantiere: multime de WME care satisface o regula
- Multime de conflicte (MC)
- Rezolvarea conflictelor

SBR cu inlantuire inainte

- Verifica la fiecare ciclu care reguli sunt aplicabile
- Match – cam 80% din timpul ciclului recunoastere-actiune
- WME
- Fiecare WME este comparat cu fiecare conditie din fiecare regula

SBR cu inlantuire inainte - eficienta

Cresterea eficientei implementarii

Partitionarea regulilor in functie de o conditie

- Elimin necesitatea sa verific la fiecare ciclu care reguli sunt aplicabile

Forward chaining incremental

- Algoritmul RETE

SBR cu inlantuire inainte - RETE

- RETE match algorithm (Forgy)
  - Salveaza starea de match intre 2 cicluri recunoastere-actiune
  - La crearea unui WME, acesta este comparat cu toate elementele conditie din program si este memorat impreuna cu fiecare element conditie cu care a identificat =>
  - Numai schimbarile incrementale din ML sunt identificate in fiecare ciclu

Exemplu JESS

```
(deftemplate student
    (slot name)
    (slot sex)
    (slot placed_in)
    (slot special_considerations (default no))
```

```
(deftemplate room
        (slot number)
        (slot capacity (type INTEGER)(default 4))
        (slot sexes_are)
        (slot vacancies (type INTEGER))
        (multislot occupants))
```

Exemplu JESS

```
(assert (room (number 112)  (capacity 4) (vacancies 4)
                                    (sexes_are nil) (occupants nil)))
(assert (student (name Mihai) (placed_in nil) (sex M)))
```

WMEs

(room 112 4 4 nil nil)	time tag1
(student Mihai nil M)	time tag 2

Stud/cam in Jess

```
(defrule  assign-student-empty-room
    ?unplaced_student <- (student (name  ?stud_name)
                                  (placed_in   nil)
                                  (sex  ?gender))
    ?empty_room <- (room (number ?room_no)
                         (capacity ?room_cap)
                         (vacancies  ?max_size))
    (test  (=   ?room_cap    ?max_size))
=>
    (modify ?unplaced_student (placed_in ?room_no))
    (modify ?empty_room (occupants  ?stud_name) (sexes_are ?gender)
                        (vacancies (-- ?max_size))
)
```

Multimea de conflicte

```
(defrule  assign-student-empty-room
    (student (name  ?stud_name)
             (placed_in nil)
             (sex  ?gender))
    (room (number ?room_no)
          (capacity ?room_cap)
          (vacancies  ?max_size))

   (test  (=   ?room_cap    ?max_size))
=> …
```

Instantieri (MC)
assign-student-empty-room 41 9
assign-student-empty-room 41 17
assign-student-empty-room 52 9
assign-student-empty-room 52 17

WMEs
41 (student name Mary sex F placed_in nil)
52 (student name Peter sex M placed_in nil)
9 (room number 221 capacity 1 vacancies 1)
12 (room number 346 capacity 2 vacancies 1)
17 (room number 761 capacity 1 vacancies 1)

Rezolvarea conflictelor

- Diferite strategii
  - Refractie = o aceeasi instantiere nu este executata de mai multe ori (2 instantieri sunt la fel daca au acelasi nume de regula si aceleasi time tags, in aceeasi ordine)
  - Momentul utilizarii = Se prefera instantierile care au identificat cu WMEs cu cele mai recente time tags (sau invers)
  - Specificitate = Au prioritate instantierile cu LHS specifice = nr de valori testate in LHS
    - teste asupra: nume clasa, predicat cu 1 arg constanta, predicat cu un operator variabila legata

Rezolvarea conflictelor

- **Strategia LEX**
  - Elimina din MC instantierile care au fost deja executate
  - Ordoneaza instantierile pe baza momentului utilizarii
  - Daca mai multe instantieri au aceleasi time tags, utilizeaza specificitate
  - Daca mai multe instantieri au aceeasi specificitate alege arbitrar
- **Strategia MEA** – aceeasi dar utilizeaza primul time tag

Construirea MC

Match

- O regula se poate aplica daca conditiile din antecedent sunt satisfacute de WMEs din ML
- Procesul are 2 aspecte:
  - match intra-element
  - match inter-element

```
(defrule  assign-student-room
    (student (name  ?stud_name) (placed_in nil)
             (sex  ?gender))
    (room (number ?room_no) (sexes_are ?gender)
          (vacancies  ?size))

   (test  (>   ? size  0))

=> ...
```

RETE - Compilarea regulilor

- Se compileaza conditiile regulilor intr-o **retea de noduri de test** – retea de flux de date
- Exsita noduri pt teste intra-element (**Alpha nodes**) si inter-element (**Beta nodes**)
- Intrarea – schimbari in ML
- Iesirea – schimbari in MC

RETE - Compilarea regulilor

- Un pointer la noul WME este trecut prin retea, incepand de la nodul radacina
- Fiecare nod actioneaza ca un switch
- Cand un nod primeste un pointer la un WME, testeaza WME asociat. Daca testul reuseste, nodul se deschide si WME trece mai departe
- Altfel nu se intampla nimic
- Daca un WME va trece prin retea, va fi combinat cu alte WME care trec pentru a forma o instantiere in MC
- Pointerii la WME sunt trimisi prin reteaua RETE ca tokens = pointer + stare (assert sau retract)

```
(defrule  cat-job-size
      (message (jobs  ?x) (status  new) (size ?y) )
      (job-def  (size-def  medium) (size ?y) )
=>
     (assert (job (job-name ?x) (job-size ?y))))
```

Retea RETE pt regula

```
(defrule show-act
    (a ?x)
    (b ?x ?y)
    (c ?y)
=> …
```

ML

```
(a 1)
(b 1 2) (b 2 3) (b 2 4)
(c 2) (c 3)
```

Stare initiala a retelei

Adaugare WME (assert)

```
(defrule  show-act
      (a  ?x)
      (b  ?x  ?y)
      (c  ?y)
=> …
```

Eliminare WME (retract)

```
(defrule show-act
      (a ?x)
      (b ?x ?y)
      (c ?y)
=> …
```

RuleML – pt Web

- RuleML Initiative – ruleml.org (2013)
- RuleML Initiative - dezvolta limbaje bazate pe reguli deschise XML/RDF
- Limbajul RuleML – sintaxa XML pentru reprezentarea cunostintelor sub forma de reguli
- Aceasta permite schimbul de reguli intre diferite sisteme, inclusiv componete software distribuite pe Web si sisteme client-server eterogene
- Integrarea cu ontologii: sistemul de reguli poate sa deriveze/utilizeze cunostinte din ontologie

RuleML

- **Reguli reactive** = Observa/verifica anumite evenimente/conditii si executa o actiune – numai forward
- **Constrangeri de integritate** = reguli speciale care semnaleaza inconistente cand se indeplinesc anumite conditii – numai forward
- **Reguli de inferenta** (derivare) = reguli reactive speciale cu actiuni care adauga o concluzie daca conditiile (premisele sunt adevarate) - Se pot aplica atat forward cat si backward
- **Fapte** = reguli de inferenta particulare

RuleML

- **Reguli reactive**

<rule> **<_body>** **<and>** *prem1 ... premN* **</and>** **</_body>**
<_head> *action* **</_head>**
</rule>

- **Constrangeri de integritate**

<ic> **<_body>** **<and>** *prem1 ... premN* **</and>** **</_body>**
<_head> *inconsistency* **</_head>**
</ic>

implementate ca

<rule> **<_body>** **<and>** *prem1* **...** *premN* **</and>** **</_body>**
<_head> **<signal>** *inconsistency* **</signal>** **</_head>**
</rule>

RuleML

- **Reguli de inferenta/derivare**

<imp> **<_head>** *conc* **</_head>**
<_body> **<and>** *prem1 ... premN* **</and>** **</_body>**
</imp >

implementate prin
<rule> **<_head>** **<assert>** *conc* **</assert>** **</_head>**
<_body> **<and>** *prem1 ... premN* **</and>** **</_body>**
</rule>

- **Fapte**

<atom> **<_head>** *conc* **</_head>** **</atom>**
implementate prin
<imp> **<_head>** *conc* **</_head>**
<_body> **<and>** **</and>** **</_body>** **</imp>**

RuleML

Fapte

```
<atom> <rel>spending</rel>
        <ind>Peter Miller</ind>
        <ind>min 5000 euro</ind>
        <ind>previous year</ind>
</atom>
```

spending(petterMiller, min5000euro, previousYear).

Reguli de inferenta/derivare

```
<imp>
  <head> <atom>
           <rel>discount</rel>
           <var>customer</var>
           <var>product</var>
           <ind>7.5 percent</ind>
         </atom>
  </head>
  <body>
    <and>
         <atom>
           <rel>premium</rel>
           <var>customer</var>
         </atom>
         <atom>
           <rel>luxury</rel>
           <var>product</var>
         </atom>
    </and>
  </body>
</imp>
```

discount(Customer, Product, 7.5_percent):-
premium(Customer), luxury(Product).

6. SBR cu inlantuire inapoi

Multe sisteme sunt combinate cu o forma de rationament incert

MYCIN – sistem expert pentru diagnosticarea infectiilor bacteriene ale sangelui

- Shortliffe, Davis, Buchanan, van Melle, si altii
  - Stanford Heuristic Programming Project
  - Infectious Disease Group, Stanford Medical

MYCIN - arhitectura

MYCIN – Context tree

MYCIN

Inlantuire inapoi bazat pe cautare in adancime cu acumulare de probe

Control la nivel inalt

- Daca pacientul are o infectie semnificativa
- Identitatea organismului
- Care sunt medicamentele adecvate
- Alege pe cel mai bun

MYCIN - Explicatii

32) Was penicillinase added to this blood culture (CULTURE-1)?
****WHY**
[i.e. WHY is it important to determine whether penicillinase was added to CULTURE-1?]

[3.0] This will aid in determining whether ORGANISM-1 is a contaminant. It has already been established that
[3.1] the site of CULTURE-1 is blood, and
[3.2] the gram stain of ORGANISM-1 is grampos
Therefore, if
[3.3] penicillinase was added to this blood culture then there is weakly suggestive evidence...

7. Factori de certitudine

- Modelul MYCIN - euristic
- Factori de certitudine / Coeficienti de incredere (**CF**)
- Model euristic al reprezentarii cunostintelor incerte

MYCIN - Exemplu de utilizare CF

- Regula in sistemul MYCIN

  daca tipul organismului este gram-pozitiv, si
  morfologia organismului este coc, si
  conformatia cresterii organismului este lant
  atunci exista o incredere puternica (**0.7**)
  ca identitatea organismului este streptococ.

- Exemple de fapte in sistemul MYCIN :

  (identitate organism-1 pseudomonas **0.8**)
  (identitate organism-2 e.coli **0.15**)
  (loc cultura-2 git **1.0**)

Functii de combinare a incertitudinii

(1) Probe adunate incremental

- Aceeasi valoare de atribut, h, este obtinuta pe doua cai de deductie distincte, cu doua perechi diferite de valori pentru CF, CF[h,s1] si CF[h,s2]
- Cele doua cai de deductie distincte, corespunzatoare probelor sau ipotezelor s1 si s2 pot fi ramuri diferite ale arborelui de cautare generat prin aplicarea regulilor sau probe indicate explicit sistemului de medic.
- **CF[h, s1&s2] = CF[h,s1] + CF[h,s2] – CF[h,s1]*CF[h,s2]**
- (identitate organism-1 pseudomonas 0.8)
- (identitate organism-1 pseudomonas 0.7)

0.8+0.7-0.8

Functii de combinare a incertitudinii

(2) Conjunctie de ipoteze

- Se aplica pentru calculul CF asociat unei premise de regula care contine mai multe conditii

  daca (A = h1) si (B = h2) **atunci** … - **s**

  ML: (A h1 CF1) (B h2 CF2)

- **CF[h1&h2, s] = min(CF1[h1,s], CF2[h2,s])**

- Se generalizeaza pt mai multe conditii

Functii de combinare a incertitudinii

Daca A=a si B=b at
(A a 0.8) (B b 0.5)
(C 10 CF=0.5 x 0.7

(3) Combinarea increderii

- O valoare incerta este dedusa pe baza unei reguli care are drept conditie de intrare alte valori incerte (deduse eventual prin aplicarea altor reguli).
- Permite calculul factorului de certitudine asociat valorii deduse pe baza aplicarii unei reguli care refera valoarea in concluzie, tinind cont de CF-ul asociat premisei regulii.
- **CF[s,e]** - increderea intr-o ipoteza **s** pe baza unor probe anterioare **e**
- **CF[h,s]** - CF in **h** in cazul in care **s** este sigura (C 10 CF=0.8)
- **CF'[h,s] = CF[h,s] * CF [s,e]**

Functii de combinare a incertitudinii

(3) Combinarea increderii – cont

daca A = h1 si B = h2 **atunci** C = c1 0.7

ML: (A h1 0.9) (B h2 0.6)

CF(premisa) = min(0.9, 0.6) = 0.6

CF (concluzie) = CF(premisa) * CF(regula) = 0.6 * 0.7

ML: (C c1 0.42)

Exemplu SBR cu inlantuire inapoi si calcul incert (CF)

- Exemplu: o baza de cunostinte pentru alegerea vinului adecvat unui meniu
- Valoarea fiecarui atribut este memorata impreuna cu coeficientul de certitudine asociat.
- Coeficientii de certitudine sunt valori pozitive in intervalul [0,1].

  (vin chardonney 0.8 riesling 0.6)
- O regula poate avea asociat un coeficient de certitudine

  daca sos-meniu = sos-alb

  atunci culoare-vin = alba 0.6

R11: daca componenta-meniu = curcan
atunci culoare-vin = rosie 0.7
si culoare-vin = alba 0.2

R12: daca componenta-meniu = peste
atunci culoare-vin = alba

R13: daca sos-meniu = sos-alb
atunci culoare-vin = alba 0.6

R14: daca componenta-meniu = porc
atunci culoare-vin = rosie

R21: daca sos-meniu = sos-alb
atunci tip-vin = sec 0.8
si tip-vin = demisec 0.6

R22: daca sos-meniu = sos-tomat
atunci tip-vin = dulce 0.8
si tip-vin = demisec 0.5

R23: daca sos-meniu = necunoscut
atunci tip-vin = demisec

R24: daca componenta-meniu = curcan
atunci tip-vin = dulce 0.6
si tip-vin = demisec 0.4

(meniu curcan) (sos-meniu sos-tomat)
R11 (c-vin rosie 0.7) (c-vin alba 0.2)
R22 (tip-vin dulce 0.8) (tip-vin demisec 0.6)
R24 (tip-vin dulce 0.6) (tip-vin demisec 0.4)
(tip-vin dulce 0.92) (tip-vin demisec 0.76)

R31: daca culoare-vin = rosie
si tip-vin = dulce
atunci vin = gamay

R32: daca culoare-vin = rosie
si tip-vin = sec
atunci vin = cabernet-sauvignon

R33: daca culoare-vin = rosie
si tip-vin = demisec
atunci vin = pinot-noir

R34: daca culoare-vin = alba
si tip-vin = dulce
atunci vin = chenin-blanc

R35: daca culoare-vin = alba
si tip-vin = sec
atunci vin = chardonnay

R36: daca culoare-vin = alba
si tip-vin = demisec
atunci vin = riesling

(meniu curcan) (sos-meniu sos-tomat)
(c-vin rosie 0.7) (c-vin alba 0.2)
(tip-vin dulce 0.92) (tip-vin demisec 0.76) (vin gamay 0.7 riesling

2.3 Limitari ale modelului CF

- Modelul coeficientilor de certitudine din MYCIN presupune ca ipotezele sustinute de probe sunt independente.
- Un exemplu care arata ce se intimpla in cazul in care aceasta conditie este violata.

Fie urmatoarele fapte:

A: Aspersorul a functionat noaptea trecuta.

U: Iarba este uda dimineata.

P: Noaptea trecuta a plouat.

Limitari ale modelului CF - cont

A: Aspersorul a functionat noaptea trecuta.

U: Iarba este uda dimineata.

P: Noaptea trecuta a plouat.

si urmatoarele doua reguli care leaga intre ele aceste fapte:

R1: **daca** aspersorul a functionat noaptea trecuta

atunci exista o incredere puternica (0.9) ca iarba este uda dimineata

R2: **daca** iarba este uda dimineata

atunci exista o incredere puternica (0.8) ca noaptea trecuta a plouat

Limitari ale modelului CF - cont

- CF[U,A] = 0.9
- deci proba aspersor sustine iarba uda cu 0.9

A: Aspersorul a functionat noaptea trecuta.
U: Iarba este uda dimineata.
P: Noaptea trecuta a plouat.
si urmatoarele doua reguli care leaga intre ele aceste fapte:
R1: **daca** aspersorul a functionat noaptea trecuta
atunci exista o incredere puternica (0.9
R2: **daca** iarba este uda dimineata
atunci exista o incredere puternica (0.8

- CF[P,U] = 0.8
- deci iarba uda sustine ploaie cu 0.8

- CF[P,A] = 0.8 * 0.9 = 0.72
- deci aspersorul sustine ploaia cu 0.72
- **Solutii**

Inteligenta Artificiala

Universitatea Politehnica Bucuresti
Anul universitar 2017-2018

Adina Magda Florea

Curs nr. 10

Agenti inteligenti

1. Definitii ale agentilor in stiinta calculatoarelor

- Nu exista o definitie unanim acceptata
- IA, agenti inteligenti, sisteme multi-agent
- Aparent agentii sunt dotati cu inteligenta
- Sunt toti agentii inteligenti?
- **Agent** = definit mai mult prin caracteristici, unele pot fi considerate ca manifestari ale unui comportament inteligent

Definitii agenti

- “De cele mai multe ori, oamenii folosesc termenul agent pentru a referi o etitate care functioneaza **permanent** si **autonom** intr-un mediu in care exsita alte procese si/sau alti agenti” (Shoham, 1993)
- “Un agent este o entitate care **percepe** mediul in care se afla si **actioneaza** asupra acestuia” (Russell, 1997)

- "**Agent** = un sistem (software sau hardware) cu urmatoarele proprietati:

↓ **autonomie** – agentii opereaza fara interventai directa a utilizatorui si au un anumit control asupra actiunilor si starilor lor;

Actiune autonoma flexibila

↓ **reactivitate**: agentii percep mediul si reactioneaza corespunzator al schimbarile din acesta;

↓ **pro-activitate**: agentii, pe langa reactia la schimbarile din mediu, sunt capabili sa urmareasca executia scopurilor si sa actioneze independent;

↓ **abilitati sociale** – agentii interactioneaza cu alti agenti sau cu utilizatorul pe baza unui limbaj de comunicare.

(Wooldridge and Jennings, 1995)

2. Caracteristici agenti

2 directii de definitie

- Definirea unui agent izolat
- Definirea agentilor in colectivitate ➔ **dimensiune sociala** ➔ **SMA**

2 tipuri de definitii

- Nu neaparat agenti inteligenti
- Include o comportare tipica IA ➔ **agenti inteligenti**

Caracteristici agenti

- Actioneaza pentru un utilizator sau un program
- Autonomie
- Percepe mediul si actioneaza asupra lui reactiv
- Actiuni pro-active
- Caracter social
- Functionare continua (persistent software)
- Mobilitate

inteligenta?

- **Scopuri, rationalitate**
- **Rationament, luarea deciziilor** ***cognitiv***
- **Invatare/adaptare**
- **Interactiune cu alti agenti – dimensiune sociala**

Alte moduri de a realiza inteligenta?

SMA – mai multi agenti in acelasi mediu

- **Interactiuni intre agenti**

  - nivel inalt

- Interactiuni pentru- coordonare
  - comunicare
  - organizare

- **Coordonare**

  ➔ motivati colectiv

  ➔ motivati individual

- scopuri proprii / indiferenta
- scopuri proprii / competitie pentru resurse
- scopuri proprii si contradictorii / competitie pentru resurse
- scopuri proprii / coalitii

- **Comunicare**

  ➔ protocol

  ➔ limbaj

- negociere
- ontologii

- **Structuri organizationale**

  ➔ centralizate vs decentralizate

  ➔ ierarhie/ piata

  abordare *"agent cognitiv"*

3 Tipuri de agenti

3.1 Agenti cognitivi

Modelul uman al perspectivei asupra lumii ➔ caracterizare agent utilizand reprezentari simbolice si ***notiuni mentale***

- knowledge - cunostinte
- beliefs - convingeri
- desires, goals – dorinte, scopuri
- intentions - intentii
- commitments - angajamente
- obligations - obligatii

3.2 Agenti reactivi

- Unitati simple de prelucrare care percep mediul si reactioneaza la schimbarile din mediu
- Nu folosesc reprezentari simbolice sau rationament.
- Inteligenta nu este situata la nivel individual ci distribuita in sistem, rezulta din interactiunea entitatilor cu mediu – “emergence”

3.3 Exemple de probleme tipice

Problema inteleptilor

Regele picteaza cate o pata alba si spune ca cel putin o pata este alba

Dilema prizonierului

Rezultatele pentru A si B (in puncte ipotetice) in functie de combinatiile de actiuni executate

Player A / **Player B**	*Defect*	*Cooperate*
Defect	**2 , 2**	**5 , 0**
Cooperate	**0 , 5**	**3 , 3**

Problema prazilor si vanatorilor

- **Abordare cognitiva**
  - Vanatorii au scopuri, prazile nu
  - Detectia prazilor
  - Echipa vanatori, roluri
  - Comunicare/cooperare

Abordare reactiva

- Prazile emit semnale a caror intensitate scade pe masura cresterii distantei de vanatori
- Vanatorii emit semnale care pot fi percepute de alti vanatori
- Fiecare vanator este atras de o prada si respins de alt semnal de la un vanator

- **Cautare distribuita**
- **Planificare distribuita**

4. Modele arhitecturale de agenti

- Structura conceptuala a agentilor
- Arhitecturi de agenti cognitivi
- Arhitecturi de agenti reactivi

4.1 Structura conceptuala a agentilor

Rationalitatea unui agent

- Ce inseamna rationalitatea unui agent
- *Cum putem masura rationalitatea unui agent?*
- O masura a performantei
- Arhitectura simpla -> se detaliaza treptat

Modelare agent reactiv

$E = \{e_1, .., e, ..\}$
$P = \{p_1, .., p, ..\}$
$A = \{a_1, .., a, ..\}$

Agent reactiv

$\mathbf{\mathit{see} : E \rightarrow P}$

$\mathbf{\mathit{action} : P \rightarrow A}$

$\mathbf{\mathit{env} : E \times A \rightarrow E}$

$\mathbf{(\mathit{env} : E \times A \rightarrow \mathit{P}(E))}$

Modelare agenti reactivi

Mai multi agenti reactivi

$$see_i : E \rightarrow P_i$$
$$action_i : P_i \rightarrow A_i$$
$$env : E \times A_1 \times \ldots A_n \rightarrow P(E)$$

$A_1, \ldots, A_i, ..$

$P_1, \ldots, P_i, ..$

(de obicei identice)

Modelare agent cognitiv

$E = \{e_1, .., e, ..\}$
$P = \{p_1, .., p, ..\}$
$A = \{a_1, .., a, ..\}$
$S = \{s_1, .., s, ..\}$

Agent cu stare

$\mathbf{\mathit{see} : E \rightarrow P}$
$\mathbf{\mathit{next} : S \times P \rightarrow S}$
$\mathbf{\mathit{action} : S \rightarrow A}$
$\mathbf{\mathit{env} : E \times A \rightarrow \mathit{P}(E)}$

Modelare agenti cognitivi

S1,…, Si,..

A_1,…, A_i,..

P_1,…, P_i,..

(nu intotdeauna identice)

$I = \{i_1, .., i_k, \ldots\}$

Mai multi agenti cognitivi

$see_i : E \rightarrow P_i$

$next_i : S_i \times P \rightarrow S_i$

$action_i : S_i \times I \rightarrow A_i$

$inter_i : S_i \rightarrow I$

$env : E \times A_1 \times \ldots A_n \rightarrow P(E)$

Modelare agent cognitiv

Agenti cu stare si scopuri

***goal* : E → {0, 1}**

Agenti cu utilitate

utility : E → R

Mediu nedeterminist

***env* : E x A → *P*(E)**

Probabilitatea estimata de un agent ca rezultatul unei actiuni (***a***) executata in *e* sa fie noua stare ***e'***

$$\sum_{e' \in env(e,a)} prob(ex(a, e) = e') = 1$$

Modelare agent cognitiv

Agenti cu utilitate

Utilitatea estimata (*expected utility)* a unei actiuni ***a*** intr-o stare ***e***, dpv al agentului

$$U(a,e) = \sum_{e' \in env(e,a)} prob(ex(a,e) = e') * utility(e')$$

Principiul utilitatii estimate maxime

Maximum Expected Utility (**MEU**)

Masura a performantei

4.2 Arhitecturi de agenti cognitivi

Arhitecturi BDI

- Specificatii de nivel inalt
- **Beliefs (convingeri) = informatii pe care agentul le are despre lume**
- **Desires (dorinte) = stari pe care agentul ar vrea sa le vada realizate**
- **Intentions (intentii) = dorinte (sau actiuni) pe care agentul s-a angajat sa le indeplineasca**

Arhitectura BDI

Bucla de control a agentului

B = B_0 **I** = I_0 **D** = D_0

while true **do**

get next perceipt p

B = **brf**(**B**,p)

D = **options**(**B**, **D**, **I**)

I = **filter**(**B**, **D**, **I**)

π = **plan**(**B**, **I**)

execute(π)

end while

Strategii de angajare (Commitment strategies)

- Optiune aleasa de agent ca intentie – **agentul s-a angajat pentru acea optiune**
- Persistenta intentiilor

Interbare: Cat timp se angajeaza un agent fata de o inetntie?

- **Angajare oarba (Blind commitment)**
- **Angajare limitata (Single minded commitment)**
- **Angajare deschisa (Open minded commitment)**

Bucla de control BDI

angajare oarba

```
B = B0
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B)) do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if not sound(π, I, B) then
                        π = plan(B, I)   ←——— Reactivity, replan
        end while
end while
```

Bucla de control BDI

angajare limitata

```
B = B0
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B)  or impossible(I, B))  do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if not sound(π, I, B) then
                        π = plan(B, I)
        end while
end while
```

Dropping intentions that are impossible or have succeeded

Reactivity, replan

Bucla de control BDI

angajare deschisa

```
B = B0
I = I0  D = D0
while true do
        get next perceipt p
        B = brf(B,p)
        D = options(B, D, I)
        I = filter(B, D, I)
        π = plan(B, I)
        while not (empty(π) or succeeded (I, B) or impossible(I, B)) do
                α = head(π)
                execute(α)
                π = tail(π)
                get next perceipt p
                B = brf(B,p)
                if reconsider(I, B) then
                        D = options(B, D, I)
                        I = filter(B, D, I)
                π = plan(B, I)        ←——— Replan
        end while
end while
```

- Nu exista o unica arhitectura BDI

- PRS - Procedural Reasoning System (Georgeff)
- dMARS
- UMPRS si JAM – C++
- JACK – Java
- JADE - Java
- JADEX – XML si Java,
- JASON – Java

4.3 Arhitecturi de agenti reactivi

Arhitectura de subsumare - Brooks, 1986

- (1) Luarea deciziilor = {**Task Accomplishing Behaviours**}
  - Fiecare comportare (behaviour) = o functie ce realizeaza o actiune
  - Implementare: ***situation → action***
- (2) Mai multe comportari pot fi activate in paralel

Arhitectura de subsumare

- Un TAB este reprezentat de un **modul de competenta** (c.m.)
- Fiecarte c.m. executa un task simplu
- c.m. opereaza in paralel
- Nivele inferioare au prioritate fata de cele superioare
- c.m. la nivel inferior monitorizeaza si influenteaza intrarile si iesirile c.m. la nivel superior

➔ ***subsumtion architecture***

M1 = **move around** while *avoiding obstacles* ⊃ M0

M2 = **explores the environment looking for distant objects of interests** while *moving around* ⊃ M1

- Incoroprarea functionalitatii unui c.m. subordonat de catre un c.m. superior se face prin noduri **supresoare** (modifica semnalul de intrare) si noduri **inhibitoare** (inhiba iesirea)

Comportare

(c, a) – conditie-actiune; descrie comportarea

R = { (c, a) | c $\subseteq$ P, a $\in$ A} - multimea reguli de comportare

$\angle \subseteq$ R x R – relatie binara totala de inhibare

function action(p: P)
var fired: P(R), selected: A
begin
fired = {(c, a) | (c, a) $\in$ R and p $\in$ c}
for each (c, a) $\in$ fired **do**
if $\neg \exists$ (c', a') $\in$ fired such that (c', a') $\angle$ (c, a) **then return** a
return null
end

Ne aflam pe o planeta necunoscuta care contine aur. Mostre de teren trebuie aduse la nava. Nu se stie daca sunt aur sau nu. Exsita mai multi agenti autonomi care nu pot comunica intre ei. Nava transmite semnale radio: gradient al campului

Comportare

(1) **Daca** detectez obstacol **atunci** schimb directia

(2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre

(3) **Daca** am mostre **si** nu sunt la baza **atunci** urmez campul de gradient

(4) **Daca** gasesc mostre **atunci** le iau

(5) **Daca** adevarat **atunci** ma misc in mediu

$$(1) \angle (2) \angle (3) \angle (4) \angle (5)$$

Agentii pot comunica indirect:

- Depun si culeg boabe radiocative
- Pot seziza aceste boabe radioactive

(1) **Daca** detectez obstacol **atunci** schimb directia

(2) **Daca** am mostre **si** sunt la baza **atunci** depune mostre

(3) **Daca** am mostre **si** nu sunt la baza **atunci** depun boaba radioactiva **si** urmez campul de gradient

(4) **Daca** gasesc mostre **atunci** le iau

(5) **Daca** gasesc boabe radioactive **atunci** iau una **si** urmez campul de gradient

(6) **Daca** adevarat **atunci** ma misc in mediu

(1) $\angle$ (2) $\angle$ (3) $\angle$ (4) $\angle$ (5) $\angle$ (6)

5. Negociere

5.1 Despre negociere

Agenti motivati colectiv = cooperare

Agenti motivati individual = competitie

Negociere = interactiune -> contract

- **Negocierea include:**
  - un limbaj de comunicare
  - un protocol de negociere
  - un proces de decizie: concesii, criterii de acceptare/refuzare
- Single party **or** multi-party negotiation: one to many or many to many (**eBay** **http://www.ebay.com**)
- **Tehnici de negociere**
  - **Negociere bazata pe teoria jocurilor**
  - **Negociere euristica**
  - **Negociere bazata pe argumentare**

5.2 Negociere bazata pe teoria jocurilor

- Criterii de evaluare protocol negociere
- Agentii se comporta rational
- **Comportare rationala** = un agent prefera o utilitate / plata (***utility*** / ***payoff***) mai mare fata de una mai mica
- Functa de utilitate
  - $\mathbf{u_i: \Omega \rightarrow R}$
  - $\Omega = \{s1, s2, \ldots\}$
  - $\mathbf{u_i(s) \geq u_i(s') \ (s \geq s')}$**–** ordonarea preferintelor asupra rezultatelor

- Doi agenti au 2 actiuni posibile: D si C (Ac={C,D})
- Mediul se comporta astfel:

$$t: Ac \times Ac \rightarrow \Omega$$

t(D,D)=s1 t(D,C)=s2 t(C,D)=s3 t(C,C)=s4

sau

t(D,D)=s1 t(D,C)=s1 t(C,D)=s1 t(C,C)=s1

u1(s1)=4, u1(s2)=4, u1(s3)=1, u1(s4)=1

u2(s1)=4, u2(s2)=1, u2(s3)=4, u2(s4)=1

u1(D,D)=4, u1(D,C)=4, u1(C,D)=1, u1(C,C)=1

u2(D,D)=4, u2(D,C)=1, u2(C,D)=4, u2(C,C)=1

Agent1 D,D ≥ D,C ≥ C,D ≥ C,C

- Agentii pot utiliza functii de utilitate pe care doresc sa le maximizeze
- Agentii trebuie sa aleaga acele actiuni care maximizeaza utilitatea (MEU)
- Daca sunt 2 sau mai multi agenti care fac actiuni care se conditioneaza reciproc – **teoria jocurilor**

Jocuri in forma normala

Forma normala a jocurilor

Un joc in forma normala cu **n** persoane este un tuplu (N,A,U) unde:

- **N** este o multime finita de **n** jucatori (indexati cu ***i***)
- $\mathbf{A = A_1 \times \ldots \times A_n}$ unde $\mathbf{A_i}$ este multimea de actiuni a jucatorului ***i***.
- Fiecare vector $\mathbf{a=(a_1, .., a_n) \in A}$ se numeste **profil de actiune**
- $\mathbf{U=(u_1,..,u_n)}$ unde u_i:A $\rightarrow$R este functia de utilitate (payoff) a jucatorului ***i***.

- Un mod natural de reprezentare este printr-o ***matrice de n dimensiuni***

Exemple de jocuri in forma normala

Prisoner's dilema

Player A / Player B	*Defect*	*Cooperate*
Defect	2 , 2	5 , 0
Cooperate	0 , 5	3 , 3

- Cooperate = nu marturiseste
- Defect = marturiseste

Jocuri cu castig comun

Jocul coordonarii

		Column player	
		Left	Right
Row player	Left	1, 1	0, 0
	Right	0, 0	1, 1

- Un joc cu castig comun **common payoff game** este un joc in care pentru toate actiunile $\mathbf{a} \in A_1 x \ldots x\ A_n$ si pentru orice pereche de agenti ***i***, ***j*** exista egalitate $\mathbf{u_i(a)=u_j(a)}$
- Agentii nu au interese conflictuale, scopul lor este sa se coordoneze in actiuni pentru beneficiul maxim comun

Jocuri cu suma nula

Matching pennies

	Heads	Tails
Heads	1, -1	-1, 1
Tails	-1, 1	1, -1

- Jocuri constante sau cu suma nula
- Un joc este un joc cu suma constanta (**constant sum game**) daca exista o constanta **c** atfel incat pentru fiecare strategie $a \in A_1 \times A_2$ exista relatia $\mathbf{u_1(a)+u_2(a)=c}$
- Daca c=0 atunci **joc cu suma nula**
- Reprezinta competitie pura

Battle of sexes

	Football	Movie
Football	2, 1	0, 0
Movie	0, 0	1, 2

- Jocuri care au atat elemente de cooperare cat si de competitie

Criterii in negociere

- **Comportare rationala** = utilitate (payoff) mai mare preferata fata de una mai mica
- **Maximizarea platii**: plata individuala, plata de grup, sau bunastare sociala
- **Bunastare sociala**
  - Suma utilitatii agentilor pentru o anumita situatie/solutie
  - Masoara binele general
  - Problema: cum compar utilitatile

Eficienta Pareto

- **Eficienta Pareto**
  - O solutie **x**, i.e., un **vector de plata p(x_1, …, x_n)**, este **eficient Pareto**, i.e., Pareto optimal, daca nu exista alta solutie **x'** a.i. cel putin un agent are o utilitate mai mare in **x'** decat in **x** si nici un agent nu are o utilitate mai mica in **x'** decat in **x**.
  - Masoara bunastarea globala darnu necesita compararea utilitatilor
  - Bunastarea sociala $\subset$ eficienta Pareto
- **Rationalitate individuala (IR)**
  - **IR a participarii unui agent** = Plata agentului in urma participarii la negociere nu este mai mica decat plata lui daca nu ar participa la negociere
  - **O negociere este IR** daca este IR pentru toti agentii

Strategie dominanta

- **Stabilitate**
  - un protocol este **stabil** daca o data ce agentii au ajuns la o solutie ei nu deviaza de la aceasta
- **Strategie dominanta** = agentul are o utilitate mai mare folosind aceasta strategie indiferent de ce strategii folosesc ceilati agenti

$s_{ij} = t(Act_i, Act_j)$ rezultatul (starea) actiunilor Act_i a agentului i si Act_j a agentului j.

- O strategie $S_1 = \{s_{11}, s_{12}, \ldots, s_{1n}\}$ ***domina*** o alta strategie $S_2 = \{s_{21}, s_{22}, \ldots, s_{2n}\}$ daca orice rezultat $s_{ij} \in S_1$ este preferat (este mai bun) oricarui rezultat $s_{ij}' \in S_2$.

Echilibru Nash

Echilibru Nash

- Doua strategii, S_1 a agentului A si S_2 a agentului B sunt in **echilibru Nash** :
  - daca agentul A urmeaza S_1 atunci agentul B nu poate obtine un castig mai mare decat acela obtinut daca urmeaza S_2 **si**
  - daca agent B urmeaza S_2 atunci agentul A nu poate obtine un castig mai mare decat acela obtinut daca urmeaza S_1.
- Multime de strategii $\{S_1, \ldots, S_k\}$ folosite de agentii $A_1, \ldots, A_k$ sunt in **echilibru Nash** daca, pentru orice agent A_i, strategia S_i este cea mai buna strategie a lui A_i daca ceilalti agenti folosesc $\{ S_1, S_2, \ldots, S_{i-1}, S_{i+1}, \ldots, S_k.\}$.

Probleme:

- nici un echilibru Nash
- multiple echilibre Nash

Prisoner's dilema

Player A / Player B	*Defect*	*Cooperate*
Defect	2 , 2	5 , 0
Cooperate	0 , 5	3 , 3

Jocuri cu castig comun

Jocul coordonarii

		Column player	
		Left	Right
Row player	Left	1, 1	0, 0
	Right	0, 0	1, 1

Jocuri care au atat elemente de cooperare cat si de competitie

Battle of sexes

	Football	Movie
Football	2, 1	0, 0
Movie	0, 0	1, 2

Jocuri cu suma nula

Matching pennies

	Heads	Tails
Heads	1, -1	-1, 1
Tails	-1, 1	1, -1

Game of Chicken

		Column player	
		Swerve	Strait
Row player	Swerve	Tie, Tie	Lose, Win
	Strait	Win, Lose	Crash, Crash

		P2	
		A	B
P1	A	0, 0	-1, +1
	B	+1, -1	-10, -10

Strategii in jocuri repetate

Turneul Axelrod

Strategii

- **ALL-D** – D tot timpul
- **RANDOM** –C sau D cu probabilitate egala
- **TIT-FOR-TAT**
  - C in primul tur
  - In turul t>1 ce a ales oponentul in t-1
- **TESTER**
  - D in primul tur
  - Daca oponentul a ales D atunci TIT-FOR-TAT
  - Altfel joaca 2 tururi C si 1 tur D
- **JOSS**
  - TIT-FOR-TAT – dar cu 10% D

6.3 Licitatii

- Protocoale simple
- Centralizate
- Licitatii cu valoare privata
- Liciatii cu valoare comuna
- Licitatii cu valoare corelata

Protocoale de licitatii

- **English (first-price open cry) auction** – fiecare participant anunta deschis pretul pe care il liciteaza. Cel mai mare pret castiga
  - Strategie dominanta: putin mai mult decat ultimul pret, ma opresc cand ajung la valoarea privata – in licitatii cu valoare privata
  - In licitatii cu valoare corelata; creste cosntant pretul pana decizie stop
- **First-price sealed-bid auction –** fiecare participant anunta pretul in plic inchis. Castiga cel cu pret maxim
  - Nu exista stragie dominanta; ofera cel mult pana la valoare lui privata

- **Dutch (descending) auction** - the auctioneer continuously lowers the price until one of the bidders takes the item at the current price.
  - Echivalenta cu licitatia *first-price sealed-bid*
- **Vickrey (second-price sealed-bid) auction** – trimite oferta in plic inchis. castiga cel care a facut cea mai mare oferta dar plateste al doilea pret
  - Strategia dominanta: valoarea lui privata

Probleme in licitatii

6.4. Alocarea taskurilor prin negociere

- Alocare prin retea de contracte

  - Contract Net

  - Iterated Contract Net

Se afla intre negociere teoretica si euristica

FIPA - Contract net

• **Initiatorul solicita propuneri** –
cfp: specifica taskul si conditii asupra lui, de la n participanti (contractori)
• Cei n participanti **genreaza raspunsuri**:
- **i** refuza (**refuse**)
- **j**=n-i propun (**propose**), inclusiv conditii
• cfp include un deadline pt raspunsuri (apoi sunt automat excluse)
• Initiatorul **evalueaza propunerile** si selecteaza **l** (intre 1 si j) agenti –
li se trimite mesaj de **accept-proposal** si mesaj de **reject-proposal** la **k**=j-l agenti
• Propunerile **angajeaza** participantii
• Un participant trebuie sa **raspunda** cu:
- inform-done sau
-inform-result (daca a executat taskul) sau
-failure.
Interactiunea este identificata printr-un unic parametru *conversation-id*

Exemplu

Agentul *j* ii cere lui *i* propuneri de vanzare a 50 cutii de prune si conditii de pret

```
(cfp
  :sender (agent-identifier :name j)
  :receiver (set (agent-identifier :name i))
  :content
    "((action (agent-identifier :name i)
      (sell plum 50))
     (any ?x (and (= (price plum) ?x) (< ?x 10))))"
  :ontology fruit-market
  :language fipa-sl)
```

Exemplu

Agentul *j* propune lui *i* sa-i vanda 50 cutii prune la pret de 5

```
(propose
  :sender (agent-identifier :name j)
  :receiver (set (agent-identifier :name i))
  :content
    "((action j (sell plum 50))
     (= (any ?x (and (= (price plum) ?x) (< ?x 10))) 5)"
  :ontology fruit-market
  :in-reply-to proposal2
  :language fipa-sl)
```

Exemplu

Agentul *i* accepta pe j

```
(accept-proposal
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :in-reply-to bid089
  :content
   "   ((action (agent-identifier :name j)
                (sell plum 50))
                (= (price plum) 5))) "
  :language fipa-sl)
```

Exemplu

Agentul *i* il refuza pe k

```
(reject-proposal
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name k))
  :content
    "((action (agent-identifier :name k)
        (sell plum 50))
      (= (price plum) 20)
      (price-too-high 20))"
  :in-reply-to bid080)
```

FIPA – Iterated Contract net